Provable Security and Real-World Protocols: Theory and Practice

A dissertation submitted towards the degree
Doctor of Natural Sciences
of the Faculty of Mathematics and Computer Science
of Saarland University

by

Mang Zhao

Saarbrücken, 2023

Date of Colloquium:	March 18, 2024
Dean of the Faculty:	Prof. Dr. Jürgen Steimle
Chair of the Committee:	Prof. Dr. Jan Reinike
Reviewers:	Prof. Dr. Cas Cremers
	Prof. Dr. Antoine Joux
	Dr. Nico Döttling
Academic Assistant:	Dr. Aleksi Peltonen

以一生一世来铸就的事业，非旁人能理解，寂寞必然，浪漫也必然，因为爱足以遮蔽一切艰辛。如果仍觉艰辛，那就是爱得不够。

——小思《一瓦之缘》

A career built for a lifetime can be hardly comprehended by others. Loneliness is inevitable, but so is romance, for love has the power to conceal arduousness completely. If arduousness is still palpable, then love is not enough.

— Xiao Si, *Once Seeing a Tile*

Abstract

In our modern life, network communication has become one of the primary mediums for information transmission, e.g., instant messaging, online shopping, and video conferencing. In order to protect the security of information transmitted over networks, real-world applications are often equipped with cryptographic communication protocols. While some of these protocols are built from small cryptographic primitives and expected to offer strong security guarantees based on some intuition or informal arguments, their comprehensive formal analyses are however missing. A natural question arises: whether these protocols really satisfy their expected security guarantees?

This thesis presents the provable security analysis of five cryptographic primitives and large-scale communication protocols: (1) one of the most efficient digital signature implementations Ed25519, (2) the generic authenticated encryption with associated data scheme, (3) the latest version of passwordless authentication standard FIDO2, (4) the popular video conferencing library Zoom, and (5) our own proposal of extended secure messaging protocol eSM. Moreover, this thesis highlights their essential security features in theory and provides suggestions for their practical deployments. In the end, this thesis addresses common obstacles to (large-scale) protocol designs and provable security analyses, provides intuition on the feasibility, and leaves a complete and provable solution as my future work.

Zusammenfassung

In unserer modernen Welt ist die Kommunikation über Netzwerke ein wichitiges Mittel zur Übertragung von Informationen. Um die Sicherheit der über Netzwerke übertragenen Informationen zu gewährleisten, werden reale Anwendungen oft mit kryptografischen Kommunikationsprotokollen ausgestattet. Einige dieser Protokolle werden aus kleinen kryptografischen Grundelementen entwickelt und sollen aufgrund von informellen Argumenten starke Sicherheitsgarantien bieten. Dennoch fehlen umfassende formale Analysen für diese Protokolle. Eine naheliegende Frage lautet: Erfüllen diese Protokolle tatsächlich ihre Sicherheitsgarantien?

Diese Arbeit präsentiert die nachweisbare Sicherheitsanalyse von fünf kryptografischen Grundlagen und Kommunikationsprotokollen: (1) eine der effizientesten digitalen Signaturen Ed25519, (2) das generische Authenticated Encryption with Associated Data Schema, (3) die neueste Version des kennwortlosen Authentifizierungsstandards FIDO2, (4) die beliebte Videokonferenzbibliothek Zoom, und (5) unseren eigenen Vorschlag für ein erweitertes sicheres Nachrichtenprotokoll, eSM. Zudem hebt diese Arbeit ihre wesentlichen Sicherheitsmerkmale in der Theorie hervor und gibt Empfehlungen für ihre praktische Implementierung. Abschließend behandelt diese Arbeit häufig auftretende Hindernisse bei groß Protokolldesigns sowie deren nachweisbare Sicherheitsanalysen, bietet Einblicke in die Durchführbarkeit und lässt eine vollständige und nachweisbare Lösung als zukünftige Arbeit offen.

Acknowledgements

First and foremost, I would like to express my endless gratitude to Cas. It has been around five years since we first met for the research immersion lab in 2018. Since that time, you've equipped me with invaluable scientific skills, teaching me how to think comprehensively about problems, conduct rigorous academic research, write articles with logical coherence, and deliver clear and concise presentations. Moreover, I'm immensely grateful for your solid support. You've encouraged me to explore topics I'm passionate about, guided me through challenges when I encountered roadblocks, and imparted the wisdom of tackling unfair criticisms with peace&love. It's been my great honor and fortunate to have you as my supervisor and to work with you. Thank you sincerely!

I would also like to extend my thanks to every member in our research group at CISPA: Jacqueline Brendel, Alexander Dax, Charlie Jacomme, Benjamin Kiesl, Philip Lukert, Niklas Medinger, and Aurora Naska. Having you as colleagues has been an incredibly enriching experience. Our shared moments, whether working diligently in the office or enjoying leisure activities like board games, billiards, bouldering, and unwinding over drinks, are treasured memories that I will cherish forever.

I want to give my special thanks to my co-authors: Nina Bindel, Jacqueline Brendel, Alexander Dax, Dennis Jackson, Charlie Jacomme, Eyal Ronen, and of course, Cas Cremers. Scientific research is undoubtedly rewarding but can also be challenging, filled with moments of uncertainty. During such times, your motivation, discussions, inspiration, and encouragement have been my guiding light. Collaborating with each of you has been an absolute pleasure.

I feel incredibly lucky to meet many people over these years and have them in my life. In particular, with all my heart, a big thank to Xiaoyi, for your unwavering support, patience, and understanding. To Yang Zhang, thank you for your care and your guide on social skills over the years. Dingfan, thank you for your assistance, your care, and your delicious food when I forget to buy groceries. To Sihang, thank you for our crypto discussions and shared drinks.

Finally, I cannot thank my family enough. Love you!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In our modern life, network communication has become one of the primary mediums for information transmission and provides people great convenience, e.g., private messages can be delivered over messaging apps instantly; video/audio conferencing can be held remotely; business services can be provided online. However, the sensitive information transmitted over insecure network channels suffers from the risk of exposure. In order to protect the security of information transmitted over networks, real-world applications are often equipped with cryptographic communication protocols. While some of these protocols are built from small cryptographic primitives and expected to offer strong security guarantees based on some intuition or informal arguments, their comprehensive formal analyses are however missing. A natural question arises: do these protocols really satisfy their expected security guarantees?

This thesis presents five projects during my doctoral studies, within a wide scope of applied cryptography that can be roughly categorized into (1) the security analysis of fundamental cryptographic primitives, including one of the most efficient digital signature implementations Ed25519 and the generic authenticated encryption with associated data scheme, and (2) the analysis and design of large-scale communication protocols, including the latest version of passwordless authentication standard FIDO2, the popular video conferencing library Zoom, and our own proposal of extended secure messaging protocol eSM.

(1) The Security Analysis of Fundamental Cryptographic Primitives. In the first project in my doctoral studies, we analyze the security of Ed25519, a popular instantiation of a novel digital signature scheme EdDSA. A standard requirement for a signature scheme is existential unforgeability (EUF-CMA), alongside other properties of interest such as strong unforgeability (SUF-CMA) and resilience against key substitution attacks. While Ed25519 is one of the most efficient signature schemes, and different instantiations of Ed25519 are widely used in protocols such as SSH, Tor, ZCash, and WhatsApp/Signal, no detailed proofs have ever been given for it or any of its variants. We observe that the differences between these instantiations are subtle and that several proofs of protocol security simply assume the EUF-CMA or SUF-CMA security of Ed25519, based on some informal arguments. In this project, we provide the first detailed analysis and security proofs of Ed25519 signature schemes. While the designs of the schemes follow the well-established Fiat-Shamir paradigm, which should guarantee EUF-CMA security, we find that the security guarantees provided by different instantiations are diverse, e.g., while Ed25519-IETF and -libsodium versions both additionally offer SUF-CMA security, only Ed25519-libsodium version provides strong message- and key-binding properties. Our work provides the scientific rationale for choosing among several Ed25519 variants and understanding their properties, fills a much-needed proof gap in modern protocol proofs that use these signatures, and supports further standardisation efforts.

In my second project, we analyze the security of Authenticated Encryption (optionally with Authenticated Data, AEAD for short) that is used in many modern security protocols such as TLS, WPA2, WireGuard, and Signal. AEADs are usually built from symmetric encryption schemes and additionally provide authentication. While this additional requirement may seem to be straightforward, it has in fact turned out to be complex: many different security notions for AEADs are still being proposed, and many recent protocol-level attacks exploit subtle AEAD behaviours and complicate the edge cases of AEAD guarantees and assumptions. This further causes the divergent landscape of AEAD definitions and their mismatch to real-world attack scenarios. To address this, we revisit several recent cryptographic AEAD definitions, extract collision resistance as a new core requirement for a generic computational AEAD model, and prove new results about their relations. Our generic computational AEAD model enables us to develop a family of symbolic AEAD models that can be used with symbolic protocol analysis tools, e.g., the Tamarin prover, for further case studies or independent research interests.

(2) The Analysis and Design of Large-scale Communication Protocols. Supported by the Microsoft identity project research grant "Developing Post-Quantum (PQ) Secure Identity Services" from Microsoft Security Response Center (MSRC), in my third project we focus on a two-party synchronous authentication protocol: FIDO2. The FIDO2 protocol is a globally used standard for passwordless authentication, building on an alliance between major players in the online authentication space. While already widely deployed, the standard is still under active development. Since version 2.1 of its CTAP sub-protocol, FIDO2 can potentially be instantiated with PQ secure primitives. We provide the first formal security analysis of FIDO2 with the CTAP 2.1 and WebAuthn 2 sub-protocols. Our security models build on work by Barbosa et al. for their analysis of FIDO2 with CTAP 2.0 and WebAuthn 1, which we extend in several ways. First, we provide a more fine-grained security model that allows us to prove more relevant protocol properties, such as guarantees about token binding agreement, the None attestation mode, and user verification. Second, we prove PQ security for FIDO2 under certain conditions and minor protocol extensions. Finally, we show that for some threat models, the downgrade resilience of FIDO2 can be improved, and show how to achieve this with a simple modification.

During the Covid-19 pandemic, video conferencing apps like Zoom enabled multi-party synchronous communication and facilitated people's work in home office. So far, video conferencing apps have hundreds of millions of daily users, making them a high-value target for surveillance and subversion. While such apps claim to achieve some forms of End-to-End Encryption (E2EE), they usually assume an incorruptible server that is able to identify and authenticate all the parties in a meeting. Concretely this means that, e.g., even when using the "end-to-end encrypted" setting, malicious Zoom servers could eavesdrop or impersonate in arbitrary groups. In my fourth project, we show how security against malicious servers can be improved by changing the way in which such protocols use passwords and integrating a password-authenticated key exchange (PAKE) protocol. To formally prove that our approach achieves its goals, we formalize a class of cryptographic protocols suitable for this setting, and define a basic security notion for them, in which group security can be achieved assuming the server is trusted to correctly authorize the group members. We prove that Zoom indeed meets this notion. We then propose a stronger security notion that can provide security against malicious servers, and propose a transformation that can achieve this notion. We show how we can apply our transformation to Zoom to provably achieve stronger security against malicious servers, notably without introducing new security elements.

E2EE is a modern requirement for communication in not only synchronous but also asynchronous settings. In my fifth project, we explore two-party secure messaging, an essential topic in the asynchronous communication domain. Recent years have seen many advances in designing secure messaging protocols, aiming at provably strong security properties in theory or high efficiency for real-world practical deployment. However, important trade-off areas of the design space inbetween these elements have not yet been explored. In this work, we design the first provably secure protocol eSM that at the same time achieves (i) strong resilience against fine-grained compromise, (ii) temporal privacy, and (iii) immediate decryption with constant-size overhead, notably, in the PQ setting. Besides these main design goals, we introduce a novel definition of offline deniability suitable for our setting, and prove that our protocol meets it, notably when combined with a PQ offline deniable initial key exchange.

The extended abstracts of these projects appear in the proceedings of IEEE Symposium on Security and Privacy and Usenix Security Symposium, as follows[1].

- Jacqueline Brendel, Cas Cremers, Dennis Jackson, and ***Mang Zhao***. "The Provable Security of Ed25519: Theory and Practice," 2021 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2021.
- Cas Cremers, Alexander Dax, Charlie Jacomme, and ***Mang Zhao***. Automated Analysis of Protocols that use Authenticated Encryption: Analysing the Impact of the Subtle Differences between AEADs on Protocol Security. In USENIX Security 2023 (*Distinguished Paper Award* winner).
- Nina Bindel, Cas Cremers and ***Mang Zhao***, "FIDO2, CTAP 2.1, and WebAuthn 2: Provable Security and Post-Quantum Instantiation," in 2023 2023 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, US, 2023 .
- Cas Cremers, Eyal Ronen, and ***Mang Zhao***, "Multi-Stage Group Key Distribution and PAKEs: Securing Zoom Groups against Malicious Servers without New Security Elements", in 2024 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, US, 2024.
- Cas Cremers and ***Mang Zhao***, "Secure Messaging with Strong Compromise Resilience, Temporal Privacy, and Immediate Decryption", in 2024 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, US, 2024.

The provable security analyses in these projects are conducted in a conventional three-steps methodology: (1) first formally model the desired security requirements, (2) then propose novel or recall existing protocols, and (3) finally claim and prove that these protocols are secure in the threat models. However, such a conventional methodology lacks of generality and often requires high efforts on the similar or repeated proofs. This thesis further explains how this observation motivates my invention of a novel and more efficient methodology for security analyses and design, and leaves a complete and provable solution as my future work.

[1]In the research domain of these works, alphabetic author order is common, and the position of an author is not indicative of their contribution.

1.1 Overview

Chapter 2 introduces the notations and necessary cryptographic primitives as well as their associated security definitions.

Chapter 3 presents our formal analysis of three popular implementations of digital signature Ed25519, figures out the subtle differences between their designs, and clarifies the huge impacts on the final security guarantees.

Chapter 4 proves various missing or previously conjectured relations for authenticated encryption with associated data (AEAD) schemes with respect to privacy, integrity, and collision resistance, completing the picture in this domain.

Chapter 5 formally proves the security of the latest version of passwordless authentication protocol FIDO2, provides instantiations for its post-quantum security, and proposes improvements against a novel type of downgrade attack.

Chapter 6 develops a solution to improve the security of Zoom-like apps against malicious servers without introducing new security elements, formally proves our solution, and illustrates how to efficiently apply it to the Zoom version 4.0 protocol.

Chapter 7 proposes the first provably secure messaging protocol that simultaneously satisfies (1) immediate decryption with constant-size overhead, (2) temporal privacy, and (3) resilience against fine-grained compromise. Our proposal additionally achieves post-quantum security and a novel flavor of offline deniability, becoming a suitable PQ-secure candidate for Double Ratchet in Signal.

Chapter 8 completes this thesis by a brief summary of contributions and an open challenges.

Chapter 2

Background

2.1 Notations

We use PPT to denote the **P**robabilistic **P**olynomial **T**ime and QPT to denote **Q**uantum **P**olynomial **T**ime. In this thesis, all algorithms are executed in PPT. We restrict the attackers' complexity Compl to be PPT in Chapter 3, Chapter 4, and Chapter 6. In Chapter 5 and Chapter 7, we assume all attackers' complexity to be $\mathsf{Compl} = \mathsf{QPT}$, unless stated otherwise. We write PQ in place of "post-quantum", meaning that the attackers have QPT complexity while the target protocols and network transmission have only PPT complexity.

We assume that each algorithm A has a security parameter λ and a public parameter pp as implicit inputs. We say a function is "negligible" a shorthand for "negligible with respect to the security parameter", unless stated otherwise.

We write $y \leftarrow A(x)$ for running a deterministic algorithm A with input x and assigning the output to y. We write $y \xleftarrow{\$} A(x; r)$ for a probabilistic algorithm A using randomness r, which is sometimes omitted when it is irrelevant. Furthermore, $\mathcal{A}^{\mathcal{O}}(x)$ denotes that $\mathcal{A}$ has access to the oracle $\mathcal{O}$ during its execution on input x. For variables x, y, we denote by $y \leftarrow x$, the assignment of value x to y and by $s \xleftarrow{\$} \mathcal{D}$ we denote the sampling of an element x from the probability distribution $\mathcal{D}$; for simplicity, we denote the uniform sampling of an element x from a set X by $x \xleftarrow{\$} X$. Security experiments $\mathrm{Expr}_{\Pi}^{\mathsf{sec\text{-}prop}}(\mathcal{A})$ describe the run of an Compl attacker $\mathcal{A}$ against the security property $\mathsf{sec\text{-}prop}$ of a cryptographic scheme Π parametrized by an implicit input pp, if the complexity $\mathsf{Compl} \in \{\mathsf{PPT}, \mathsf{QPT}\}$ is unambiguous from the context. We use $\epsilon_{\Pi}^{\mathsf{sec}}$ to denote the advantage of any Compl attacker that breaks sec security of Π protocol.

Let $(\cdot)$ and $\{\cdot\}$ respectively denote an ordered tuple and an unordered set. We use $\{0,1\}^\star$ to denote the set of all strings with finite length. For any positive integer n, let $[n]$ denote the set of integers from 1 to n, i.e., $[n] = \{1, ..., n\}$. We use $\mathcal{D}$ to denote a dictionary that stores values for each index and $\mathcal{D}[\cdot] \leftarrow \perp$ for the dictionary initialization. The functionality of symbol $|\cdot|$ is diverse in this thesis: For a set X, $|X|$ denotes the cardinality of X; For a number x, $|x|$ denotes the absolute value of x; For a string s, $|s|$

denotes the bit-length of s. To avoid duplicating some longer variable names, we use some update-based variants of common operators: For a number x, we write x++ as a shorthand for $x \leftarrow x + 1$. For a set S and an element x, we write $S \overset{+}{\leftarrow} x$ for $S \leftarrow S \cup \{x\}$, and $S \overset{-}{\leftarrow} x$ for $S \leftarrow S \setminus \{x\}$.

Let $\perp$ denote an special reserved symbol that is not included in any set in this paper. All undefined variables are initialized with $\perp$. We write $\top$ to denote the empty string, and write $x \parallel y$ to denote the concatenation of strings x and y. We use $\_$ to denote a variable that is irrelevant. We use 1 to represent the Boolean value *True*, and 0 for *False*. By $[\![statement]\!]$, we denote the Boolean evaluation of *statement*. Within algorithms or oracles, we write "**require** C" to denote that C is a requirement: if the condition C is not met, then the algorithm or oracle containing this keyword is exited with output $\perp$ and all actions in this invocation are undone.

2.2 Cryptographic Primitives

In this section, we recall the necessary definitions of cryptographic primitives and security models. Our presentation of some related security definitions follows the style of the textbook by Katz and Lindell [124] with security defined in the *concrete setting* which explicitly specifies the amount of time and resources needed (cf. [124, Sec. 3.1]). For simplicity, we define other related security notions in a rough setting without specifying the amount of time and resources needed, but only expect them to be in polynomial with respect to the implicit security parameter.

2.2.1 Security Assumptions on Cyclic Groups

Definition 1 (sCDH). *Let $\mathbb{G} = \langle g \rangle$ denotes a cyclic group of prime order q with generator g. We say the* computational Diffie-Hellman (CDH) *problem is $\epsilon^{\mathsf{CDH}}_{\mathbb{G},g}$ hard if for all* PPT *attackers $\mathcal{A}$ it holds that*

$$\Pr[g^{ab} \leftarrow \mathcal{A}(\mathbb{G}, g, g^a, g^b) : a, b \overset{\$}{\leftarrow} \mathbb{Z}_q] \leq \epsilon^{\mathsf{CDH}}_{\mathbb{G},g}$$

We say the strong CDH (sCDH) *problem is $\epsilon^{\mathsf{sCDH}}_{\mathbb{G},g}$ hard if for all* PPT *attackers $\mathcal{A}$ the CDH problem is $\epsilon^{\mathsf{CDH}}_{\mathbb{G},g} = \epsilon^{\mathsf{sCDH}}_{\mathbb{G},g}$ hard even when $\mathcal{A}$ has access to an oracle $\mathcal{O}_a(\cdot,\cdot)$ that inputs $Y, Z \in \mathbb{G}$ and outputs whether $Y^a = Z$.*

2.2.2 Pseudorandom and Hash Functions

Definition 2. *Let $\mathsf{H} : \mathcal{M} \to \mathcal{O}$ denote a function that maps from a message space $\mathcal{M}$ to an output space $\mathcal{O}$. We say H is ϵ-*collision resistant*, if for any attacker $\mathcal{A}$ it holds that,*

$$\mathsf{Adv}^{\mathsf{coll\text{-}res}}_{\mathsf{H}}(\mathcal{A}) := \Pr[(m_1, m_2) \overset{\$}{\leftarrow} \mathcal{A} \text{ such that } m_1, m_2 \in \mathcal{M}, m_1 \neq m_2, \mathsf{H}(m_1) = \mathsf{H}(m_2)] \leq \epsilon$$

Definition 3. *Let* $\mathsf{F} : \mathcal{R} \to \mathcal{O}$ *denote a function that maps a random string* $r \in \mathcal{R}$ *to an output* $y \in \mathcal{O}$*. We say* F *is* ϵ-prg *secure if for any variable* X *that follows uniform distribution over* $\mathcal{R}$ *and any variable* Y *that follows uniform distribution over* $\mathcal{O}$*, we have*

$$\mathsf{Adv}^{\mathsf{prg}}_{\mathsf{F}}(\mathcal{D}) := \Big| \Pr[\mathcal{D}(\mathsf{F}(X)) = 1] - \Pr[\mathcal{D}(Y) = 1] \Big| \leq \epsilon$$

Definition 4. *Let* $\mathsf{F} : \mathcal{K} \times \mathcal{M} \to \mathcal{O}$ *be a function that maps a key* $k \in \mathcal{K}$ *and a message* $m \in \mathcal{M}$ *to an output* $y \in \mathcal{O}$*. We say* F *is* ϵ-prf *secure, if for any attacker* $\mathcal{A}$ *and any* $k \xleftarrow{\$} \mathcal{K}$*, there exists a truly random function* $\mathsf{R} : \mathcal{M} \to \mathcal{O}$*, it holds that,*

$$\mathsf{Adv}^{\mathsf{prf}}_{\mathsf{F}}(\mathcal{A}) := \big| \Pr[\mathcal{A}^{\mathsf{F}(k,\cdot)} = 1] - \Pr[\mathcal{A}^{\mathsf{R}(\cdot)} = 1] \big| \leq \epsilon$$

Moreover, we say F *is* ϵ-swap *secure, if the function* $\bar{\mathsf{F}}$ *defined below is* ϵ-prf *secure.*

$$\bar{\mathsf{F}} : \mathcal{M} \times \mathcal{K} \to \mathcal{O}, \bar{\mathsf{F}}(m, k) := \mathsf{F}(k, m)$$

We say F *is* ϵ-dual *secure, if* F *is both* ϵ-prf *and* ϵ-swap *secure.*

Definition 5. *Let* $m \geq 2$*. Let* $\mathsf{F} : \mathcal{K}_1 \times ... \times \mathcal{K}_m \to \mathcal{O}$ *be a function that maps* m *keys* $k_i \in \mathcal{K}_i$ *for* $1 \leq i \leq m$ *to an output* $y \in \mathcal{O}$*. We say* F *is* ϵ-mprf*-secure if all of the functions* $\bar{\mathsf{F}}_i(k_i, (k_1, ..., k_{i-1}, k_{i+1}, ..., k_m)) := \mathsf{F}(k_1, ..., k_m)$ *is* prf*-secure.*

The mprf secure function can be easily construction from dual-secure functions. In this thesis, we only makes use of a mprf-secure KDF for $m = 3$. Below, we present the instantiation and prove the security.

Theorem 1. *Let* $\mathsf{F}_1 : \mathcal{K}_1 \times \mathcal{K}_2 \to \mathcal{O}_1$ *and* $\mathsf{F}_2 : \mathcal{O}_1 \times \mathcal{K}_3 \to \mathcal{O}_2$ *be two functions. If* F_1 *and* F_2 *both are* ϵ-dual*-secure, then the function* $\mathsf{F}'(k_1, k_2, k_3) := \mathsf{F}_2(\mathsf{F}_1(k_1, k_2), k_3)$ *is* ϵ'-3prf*-secure such that* $\epsilon' \leq q\epsilon$*, where* q *denotes the number of queries by any attacker against* 3prf*-security of* F'*.*

Proof. We first show that $\bar{\mathsf{F}}_1(k_1, (k_2, k_3)) := \mathsf{F}'(k_1, k_2, k_3) = \mathsf{F}_2(\mathsf{F}_1(k_1, k_2), k_3)$ is prf-secure. We prove this by game hopping. Let q denote the number of queries that an attacker $\mathcal{A}$ makes. Let Adv_i denote the advantage of $\mathcal{A}$ in winning game i.

Game 0. This game is identical to the experiment. And we have that $\mathsf{Adv}_0 := \epsilon'$

Game 1. In this game, whenever $\mathcal{A}$ queries (k_2, k_3), the challenger samples a random y_1 and replaces $\bar{\mathsf{F}}_1(k_1, (k_2, k_3)) = \mathsf{F}_2(\mathsf{F}_1(k_1, k_2), k_3)$ by $\bar{\mathsf{F}}_1(k_1, (k_2, k_3)) = \mathsf{F}_2(y_1, k_3)$. If the attacker $\mathcal{A}$ can distinguish **Game** 0 and **Game** 1, then we can easily construct an attacker that breaks the prf security of F_1. Thus, $\mathsf{Adv}_0 - \mathsf{Adv}_1 \leq \epsilon$.

Game 2. In this game, whenever $\mathcal{A}$ queries (k_2, k_3), the challenger samples a random y_1 and replaces $\bar{\mathsf{F}}_1(k_1, (k_2, k_3)) = \mathsf{F}_2(y_1, k_3)$ by $\bar{\mathsf{F}}_1(k_1, (k_2, k_3)) = y_2$.

If the attacker $\mathcal{A}$ can distinguish **Game** 0 and **Game** 1, then we can easily construct an attacker that breaks the prf security of at least one of q F_2. Thus, $\mathsf{Adv}_0 - \mathsf{Adv}_1 \leq q\epsilon$.

```
$\mathrm{Expr}^{\mathsf{bind}}_{\mathsf{cPRF}}$:
$(k_1, m_1, k_2, m_2) \xleftarrow{\$} \mathcal{A}()$
$(r_1, t_1) \leftarrow \mathsf{cPRF}(k_1, m_1)$, $(r_2, t_2) \leftarrow \mathsf{cPRF}(k_2, m_2)$
return $[\![t_1 = t_2]\!]$
```

Figure 2.1: bind experiment for a cPRF function.

Now, in **Game** 2 the challenger always simulates the random function. Thus, $\mathcal{A}$ cannot distinguish it, and we have that $\epsilon \leq (q+1)\epsilon$.

The analysis for the prf-security of $\bar{\mathsf{F}}_2(k_2, (k_1, k_3)) := \mathsf{F}'(k_1, k_2, k_3) = \mathsf{F}_2(\mathsf{F}_1(k_1, k_2), k_3)$ and $\bar{\mathsf{F}}_3(k_3, (k_1, k_2)) := \mathsf{F}'(k_1, k_2, k_3) = \mathsf{F}_2(\mathsf{F}_1(k_1, k_2), k_3)$ is similar. $\square$

Definition 6. *Let $\mathsf{F} : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{O}$ be a function that maps a key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$ to an output $y \in \mathcal{O}$. We say F is ϵ-prp secure, if*

1. *for any $k \in \mathcal{K}$, F is bijective from $\mathcal{M}$ to $\mathcal{O}$, this indicates that $\mathcal{O} = \mathcal{M}$*
2. *for any $k \in \mathcal{K}$, $\mathsf{F}(k, m)$ can be evaluated in polynomial time for any $m \in \mathcal{M}$*
3. *for any $k \in \mathcal{K}$, the inversion $\mathsf{F}^{-1}(k, y)$ can be evaluated in polynomial time for any $y \in \mathcal{O}$*
4. *for any attacker $\mathcal{A}$ and any $k \xleftarrow{\$} \mathcal{K}$, there exists a truly random invertible permutation $f : \mathcal{M} \rightarrow \mathcal{O}$ with inversion $f^{-1} : \mathcal{O} \rightarrow \mathcal{M}$, it holds that*

$$\mathsf{Adv}^{\mathsf{prp}}_{\mathsf{F}}(\mathcal{A}) := \left| \Pr[\mathcal{A}^{\mathsf{F}(k,\cdot), \mathsf{F}^{-1}(k,\cdot)} = 1] - \Pr[\mathcal{A}^{f(\cdot), f^{-1}(\cdot)} = 1] \right| \leq \epsilon$$

We recall a primitive called committing PRF and its simplified binding security, which was first defined in [27].

Definition 7. *Let $\mathsf{cPRF} : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{R} \times \mathcal{T}$ denote a deterministic function inputs a key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$ and outputs $r \in \mathcal{R}$ and $t \in \mathcal{T}$. We say cPRF is ϵ-binding (or ϵ-bind) secure, if the below defined advantage of any attacker $\mathcal{A}$ against $\mathrm{Expr}^{\mathsf{bind}}_{\mathsf{cPRF}}$ experiment in Figure 2.1 is bounded by:*

$$\mathsf{Adv}^{\mathsf{bind}}_{\mathsf{cPRF}}(\mathcal{A}) := \Pr[\mathrm{Expr}^{\mathsf{bind}}_{\mathsf{cPRF}}(\mathcal{A}) = 1] \leq \epsilon$$

2.2.3 Symmetric Key Encryption

Definition 8. *A symmetric key encryption scheme over key space $\mathcal{K}$, message space $\mathcal{M}$, randomness space $\mathcal{R}$, and ciphertext space $\mathcal{CT}$, is a tuple of algorithms $\mathsf{SKE} = (\mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ as defined below.*

- ***Encryption*** *$c \xleftarrow{\$} \mathsf{SKE.Enc}(k, m)$: takes as input a symmetric key k and a message m and outputs a ciphertext c. We write $c \xleftarrow{\$} \mathsf{SKE.Enc}(k, m; r^{\mathsf{Enc}})$ if the random coins $r^{\mathsf{Enc}} \in \mathcal{R}$ is specified.*

```
$\mathrm{Expr}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SKE}}(\mathcal{A})$:
    $\mathsf{b} \xleftarrow{\$} \{0,1\}$
    $K \xleftarrow{\$} \mathsf{SKE.KGen}()$
    $\mathsf{b}' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Enc}}}()$
    return $[\![\mathsf{b} = \mathsf{b}']\!]$

$\mathcal{O}_{\mathsf{Enc}}(m_0, m_1)$:
    $c \xleftarrow{\$} \mathsf{SKE.Enc}(k, m_{\mathsf{b}})$
    return $c$
```

Figure 2.2: IND-CPA experiment for a $\mathsf{SKE} = (\mathsf{SKE.KGen}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ scheme.

- ***Decryption*** *$m \leftarrow \mathsf{SKE.Dec}(k, c)$: takes as input a symmetric key k and a ciphertext c and outputs either a message m or an error symbol $\perp$.*

We say a SKE is δ-correct if for every $k \xleftarrow{\$} \mathcal{K}$ and every message $m \in \mathcal{M}$, we have

$$\Pr[m \neq \mathsf{SKE.Dec}(k, \mathsf{SKE.Enc}(k, m))] \leq \delta$$

In particular, we call a SKE *(perfectly) correct* if $\delta = 0$.

In terms of the security, we first recall the standard *indistinguishability under chosen plaintext attacks (*IND-CPA*).*

We say a SKE is δ-strongly correct if for every $k \xleftarrow{\$} \mathcal{K}$, every message $m \in \mathcal{M}$, and every $r^{\mathsf{Enc}} \in \mathcal{R}$, we have

$$\Pr[m \neq \mathsf{SKE.Dec}(k, \mathsf{SKE.Enc}(k, m; r^{\mathsf{Enc}}))] \leq \delta$$

Compared to the conventional correctness, the strong correctness requires that the encrypted message can be correctly recovered for every randomness coins involved during the encryption. In particular, we call a SKE *(perfectly) strongly correct* if $\delta = 0$.

Definition 9. *Let* $\mathsf{SKE} = (\mathsf{SKE.KGen}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ *be a symmetric key encryption scheme with symmetric key space $\mathcal{K}$. We say* SKE *is ϵ-*IND-CPA *secure, if the blow defined advantage of every (potential quantum) attacker $\mathcal{A}$ against* $\mathrm{Expr}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SKE}}$ *experiment in Figure 2.2 is bounded by,*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SKE}}(\mathcal{A}) := \left| \Pr[\mathrm{Expr}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SKE}}(\mathcal{A}) = 1] - \frac{1}{2} \right| \leq \epsilon$$

Then, we recall two notions, the *one time* IND-CPA *(*IND-1CPA*)* security [76] and *indistinguishability under one-time chosen and then random plaintext attack (*IND-1\$PA*)* security [20].

Definition 10. *Let* $\mathsf{SKE} = (\mathsf{SKE.KGen}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ *be a symmetric key encryption scheme with symmetric key space $\mathcal{K}$. We say* SKE *is ϵ-*IND-1CPA *secure, if the blow defined advantage of every (potential quantum) attacker $\mathcal{A}$ against* $\mathrm{Expr}^{\mathsf{IND\text{-}1CPA}}_{\mathsf{SKE}}$ *experiment in Figure 2.3 is bounded by,*

$$\mathsf{Adv}^{\mathsf{IND\text{-}1CPA}}_{\mathsf{SKE}}(\mathcal{A}) := \left| \Pr[\mathrm{Expr}^{\mathsf{IND\text{-}1CPA}}_{\mathsf{SKE}}(\mathcal{A}) = 1] - \frac{1}{2} \right| \leq \epsilon$$

```
$\mathrm{Expr}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CPA}}(\mathcal{A})$:
$\mathsf{b} \xleftarrow{\$} \{0,1\}$
$K \xleftarrow{\$} \mathsf{SKE.KGen}()$
$(m_0^\star, m_1^\star) \xleftarrow{\$} \mathcal{A}()$
if $|m_0^\star| \neq |m_1^\star|$
    return $0$
$c^\star \xleftarrow{\$} \mathsf{SKE.Enc}(K, m_{\mathsf{b}}^\star)$
$\mathsf{b}' \xleftarrow{\$} \mathcal{A}(c^\star)$
return $[\![\mathsf{b} = \mathsf{b}']\!]$
```

```
$\mathrm{Expr}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA}}(\mathcal{A})$:
$\mathsf{b} \xleftarrow{\$} \{0,1\}$
$K \xleftarrow{\$} \mathsf{SKE.KGen}()$
$(m_0^\star, m_1^\star) \xleftarrow{\$} \mathcal{A}()$
if $|m_0^\star| \neq |m_1^\star|$
    return $0$
$c^\star \xleftarrow{\$} \mathsf{SKE.Enc}(K, m_{\mathsf{b}}^\star)$
$\mathsf{b}' \xleftarrow{\$} \mathcal{A}^{\textsc{Rand}}(c^\star)$
return $[\![\mathsf{b} = \mathsf{b}']\!]$
```

```
$\textsc{Rand}(l)$:
$m_0', m_1' \xleftarrow{\$} \{0,1\}^l$
$c' \xleftarrow{\$} \mathsf{SKE.Enc}(K, m_{\mathsf{b}}')$
return $(m_0', m_1', c')$
```

Figure 2.3: IND-1CPA and IND-1\$PA experiments for a $\mathsf{SKE} = (\mathsf{SKE.KGen}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ scheme.

Definition 11. *Let* $\mathsf{SKE} = (\mathsf{SKE.KGen}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ *be a symmetric key encryption scheme with symmetric key space* $\mathcal{K}$. *We say* SKE *is* ϵ-IND-1\$PA *secure, if the blow defined advantage of every (potential quantum) attacker* $\mathcal{A}$ *against* $\mathrm{Expr}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA}}$ *experiment in Figure 2.3 is bounded by,*

$$\mathsf{Adv}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA}}(\mathcal{A}) := \left| \Pr[\mathrm{Expr}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA}}(\mathcal{A}) = 1] - \frac{1}{2} \right| \leq \epsilon$$

Finally, we recall the *indistinguishability under one-time chosen ciphertext attacks* (IND-1CCA) in [12]. In this security notion, the attacker is allowed to query the encryption oracle $\mathcal{O}_{\mathsf{Enc}}$ at most once. However, the attacker can have access to the decryption oracle $\mathcal{O}_{\mathsf{Dec}}$ with arbitrary times. We in particular define the IND-1CCA experiment with simplified SKE definition, i.e., $\mathsf{SKE} = (\mathsf{SKE.Enc}, \mathsf{SKE.Dec})$.

We particularly stress that the difference between IND-1CPA and IND-1CCA security. On the one hand, while the goal of attackers in IND-1CPA experiment is to distinguish two plaintexts (this is defined in a so-called "left-or-right" manner), the goal of attackers in IND-1CCA experiment is to distinguish a real ciphertext from a random ciphertext (this is defined in a so-called "real-or-random" manner). On the other hand, while the attackers in IND-1CCA experiment have access to the decryption oracle, the ones in IND-1CPA have none.

Definition 12. *Let* $\mathsf{SKE} = (\mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ *be a symmetric key encryption scheme with symmetric key space* $\mathcal{K}$ *and ciphertext space* $\mathcal{C}$. *We say* SKE *is* ϵ-IND-1CCA *secure, if the blow defined advantage of every (potential quantum) attacker* $\mathcal{A}$ *against* $\mathrm{Expr}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}}$ *experiment in Figure 2.4 is bounded by,*

$$\mathsf{Adv}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}}(\mathcal{A}) := \left| \Pr[\mathrm{Expr}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}}(\mathcal{A}) = 1] - \frac{1}{2} \right| \leq \epsilon$$

```
Expr^{IND-1CCA}_{SKE}(A):         O_Enc(m):                     O_Dec(c):
b ←$ {0,1}                        require c* = ⊥                if c = c* or b = 1
k ←$ K                            if b = 0                          return ⊥
c* ← ⊥                                c* ←$ SKE.Enc(k, m)       return SKE.Dec(k, c)
b' ←$ A^{O_Enc, O_Dec}()          else
return [[b = b']]                     c* ←$ C
                                  return c
```

Figure 2.4: $\mathsf{IND\text{-}1CCA}$ experiment for a $\mathsf{SKE} = (\mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ scheme.

2.2.4 Authenticated Encryption with Associated Data

Definition 13 ([163]). *Let $\mathcal{K}$, $\mathcal{N}$, $\mathcal{H}$, $\mathcal{M}$, $\mathcal{CT}$ respectively denote the space of keys, nonces, headers (aka. header), messages, and ciphertexts. An authenticated encryption with associated data scheme* $\mathsf{AEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$[1] *is a tuple of algorithms where*

- $\mathsf{AEAD.KGen}$ *the key generation algorithm outputs a symmetric key $k \in \mathcal{K}$, i.e., $k \xleftarrow{\$} \mathsf{AEAD.KGen}()$.*
- $\mathsf{AEAD.Enc}$ *the encryption algorithm inputs a key $k \in \mathcal{K}$, a nonce $n \in \mathcal{N}$, a header $h \in \mathcal{H}$, and a message m and (deterministically) outputs a ciphertext c, i.e., $c \leftarrow \mathsf{AEAD.Enc}(k, n, h, m)$.*
- $\mathsf{AEAD.Dec}$ *the decryption algorithm inputs a key $k \in \mathcal{K}$, a nonce $n \in \mathcal{N}$, a header $h \in \mathcal{H}$, and a ciphertext $c \in \mathcal{CT}$ and deterministically outputs a message $m \in \mathcal{M} \cup \{\perp\}$, i.e., $m \leftarrow \mathsf{AEAD.Dec}(k, n, h, c)$.*

Each AEAD scheme is assumed to be defined with a length function ℓ such that $|\mathsf{AEAD.Enc}(k, n, h, m)| = \ell(|m|)$ for all $(k, n, h, m) \in \mathcal{K} \times \mathcal{N} \times \mathcal{H} \times \mathcal{M}$.

We say an AEAD scheme is ϵ-correct if for all $(n, h, m) \in \mathcal{N} \times \mathcal{H} \times \mathcal{M}$ and $k \xleftarrow{\$} \mathsf{AEAD.KGen}()$ it holds that

$$\Pr[m' \leftarrow \mathsf{AEAD.Dec}(k, n, h, \mathsf{AEAD.Enc}(k, n, h, m)) : m \neq m'] \leq \epsilon$$

In particular, we say AEAD is perfect correct if $\epsilon = 0$.

We say an AEAD scheme is *tidy* if for each $(k, n, h, c) \in \mathcal{K} \times \mathcal{N} \times \mathcal{H} \times \mathcal{CT}$ it holds that

$$\perp \neq m \leftarrow \mathsf{AEAD.Dec}(k, n, h, c) \implies c \leftarrow \mathsf{AEAD.Enc}(k, n, h, m)$$

Over such schemes, the n, h and ciphertext c need to be sent over the network[2], and the correctness of the scheme requires that the decryption of a ciphertext with the same

[1]Note the fact that most popular AEAD constructions generate keys by simply sampling bit strings from key space uniformly at random. We sometimes also omit the key generation algorithm and simply write $\mathsf{AEAD} = (\mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$, in particular, in Chapter 6.

[2]We stress that AEAD schemes can be used offline in practice, where nonces and headers both are hidden from attackers' view. However, this paper focuses on a more common case where the attackers might have access to the nonce and headers, e.g., which are transmitted over network.

```
$\mathrm{Expr}^{\mathsf{IND\$\text{-}CPA}}_{\mathsf{AEAD}}$:
$\mathsf{b} \xleftarrow{\$} \{0,1\}$
$\mathcal{L}_{\mathcal{O}_{\mathsf{Enc}}} \leftarrow \emptyset$
$k \xleftarrow{\$} \mathsf{AEAD.KGen}()$
$\mathsf{b}' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Enc}},\mathcal{O}_{\mathsf{Dec}}}()$
**return** $[\![\mathsf{b} = \mathsf{b}']\!]$

$\mathcal{O}_{\mathsf{Enc}}(n, h, m)$:
**require** $(n, \_, \_, \_) \notin \mathcal{L}_{\mathcal{O}_{\mathsf{Enc}}}$
**if** $\mathsf{b} = 0$
    $c \leftarrow \mathsf{AEAD.Enc}(k, n, h, m)$
**else** $c \xleftarrow{\$} \{0,1\}^{\ell(|m|)}$
$\mathcal{L}_{\mathcal{O}_{\mathsf{Enc}}} \xleftarrow{+} (n, h, m, c)$
**return** $c$
```

Figure 2.5: IND\$-CPA experiment for an $\mathsf{AEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$ scheme.

```
$\mathrm{Expr}^{\mathsf{CTI\text{-}CPA}}_{\mathsf{AEAD}}$:
$\mathcal{L}_c \leftarrow \emptyset$
$k \xleftarrow{\$} \mathsf{AEAD.KGen}$
$(n, h, c) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Enc}}}()$
**require** $c \notin \mathcal{L}_c$
**return** $[\![\mathsf{AEAD.Dec}(k, n, h, c) \neq \perp]\!]$

$\mathcal{O}_{\mathsf{Enc}}(n, h, m)$:
$c \leftarrow \mathsf{AEAD.Enc}(k, n, h, m)$
$\mathcal{L}_c \xleftarrow{+} c$
**return** $c$
```

Figure 2.6: CTI-CPA experiment for an $\mathsf{AEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$ scheme.

parameters n, h, k indeed returns the plaintext. We assume that the decryption with inputs outside the corresponding spaces must output $\perp$.

The two core security guarantees are integrity and privacy.

Definition 14 (Privacy [163]). *We say an* $\mathsf{AEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$ *is* ϵ-IND\$-CPA *secure, if the below defined advantage of any attacker* $\mathcal{A}$ *against* $\mathrm{Expr}^{\mathsf{IND\$\text{-}CPA}}_{\mathsf{AEAD}}$ *experiment in Figure 2.5 is bounded by:*

$$\mathsf{Adv}^{\mathsf{IND\$\text{-}CPA}}_{\mathsf{AEAD}} := |\Pr[\mathrm{Expr}^{\mathsf{IND\$\text{-}CPA}}_{\mathsf{AEAD}}(\mathcal{A}) = 1] - \frac{1}{2}| \leq \epsilon$$

Definition 15 (Integrity [163]). *We say an* $\mathsf{AEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$ *is* ϵ-CTI-CPA *secure, if the below defined advantage of any attacker* $\mathcal{A}$ *against* $\mathrm{Expr}^{\mathsf{CTI\text{-}CPA}}_{\mathsf{AEAD}}$ *experiment in Figure 2.6 is bounded by:*

$$\mathsf{Adv}^{\mathsf{CTI\text{-}CPA}}_{\mathsf{AEAD}} := \Pr[\mathrm{Expr}^{\mathsf{CTI\text{-}CPA}}_{\mathsf{AEAD}}(\mathcal{A}) = 1] \leq \epsilon$$

Both for integrity and privacy, we can define two security variants, CTI-CCA and IND\$-CCA, based on whether the attacker also has access to a decryption oracle during the experiment, see e.g., the definition of the experiment for CTI-CCA in Figure 2.6.

We start with the confidentiality notion IND\$-CPA and extend it to IND\$-CCA in a natural way.

Definition 16. *We say an* $\mathsf{AEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$ *is* ϵ-IND\$-CCA *secure, if the below defined advantage of any attacker* $\mathcal{A}$ *against* $\mathrm{Expr}^{\mathsf{IND\$\text{-}CCA}}_{\mathsf{AEAD}}$ *experiment in Figure 2.7 is bounded by:*

$$\mathsf{Adv}^{\mathsf{IND\$\text{-}CCA}}_{\mathsf{AEAD}} := |\Pr[\mathrm{Expr}^{\mathsf{IND\$\text{-}CCA}}_{\mathsf{AEAD}}(\mathcal{A}) = 1] - \frac{1}{2}| \leq \epsilon$$

```
$\mathrm{Expr}^{\mathsf{IND\$\text{-}CCA}}_{\mathsf{AEAD}}$:
    $\mathsf{b} \xleftarrow{\$} \{0,1\}$
    $\mathcal{L}_{\mathcal{O}_{\mathsf{Enc}}}, \mathcal{L}_{\mathcal{O}_{\mathsf{Dec}}} \leftarrow \emptyset$
    $k \xleftarrow{\$} \mathsf{AEAD.KGen}$
    $\mathsf{b}' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Enc}}, \mathcal{O}_{\mathsf{Dec}}}()$
    return $[\![\mathsf{b} = \mathsf{b}']\!]$

$\mathcal{O}_{\mathsf{Enc}}(n, h, m)$:
    require $(n, \_, \_, \_) \notin \mathcal{L}_{\mathcal{O}_{\mathsf{Enc}}}$
    require $(n, h, m, \_) \notin \mathcal{L}_{\mathcal{O}_{\mathsf{Dec}}}$
    if $\mathsf{b} = 0$
        $c \leftarrow \mathsf{AEAD.Enc}(k, n, h, m)$
    else $c \xleftarrow{\$} \{0,1\}^{\ell(|m|)}$
    $\mathcal{L}_{\mathcal{O}_{\mathsf{Enc}}} \xleftarrow{+} (n, h, m, c)$
    return $c$

$\mathcal{O}_{\mathsf{Dec}}(n, h, c)$:
    require $(n, h, \_, c) \notin \mathcal{L}_{\mathcal{O}_{\mathsf{Enc}}}$
    $m \leftarrow \mathsf{AEAD.Dec}(k, n, h, c)$
    if $m \neq \bot$
        $\mathcal{L}_{\mathcal{O}_{\mathsf{Dec}}} \xleftarrow{+} (n, h, m, c)$
    return $m$
```

Figure 2.7: $\mathsf{IND\$\text{-}CCA}$ experiment for an $\mathsf{AEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$ scheme. Note that we assume "nonce-respecting", which means the encryption oracle cannot be repeatedly queried with same nonces.

```
$\mathrm{Expr}^{\mathsf{CTI\text{-}CCA}}_{\mathsf{AEAD}}$:
    $\mathcal{L}_c \leftarrow \emptyset$
    $k \xleftarrow{\$} \mathsf{AEAD.KGen}()$
    $(n, h, c) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Enc}}, \mathcal{O}_{\mathsf{Dec}}}()$
    require $c \notin \mathcal{L}_c$
    return $[\![\mathsf{AEAD.Dec}(k, n, h, c) \neq \bot]\!]$

$\mathcal{O}_{\mathsf{Enc}}(n, h, m)$:
    $c \leftarrow \mathsf{AEAD.Enc}(k, n, h, m)$
    $\mathcal{L}_c \xleftarrow{+} c$
    return $c$
$\mathcal{O}_{\mathsf{Dec}}(n, h, c)$:
    return $\mathsf{AEAD.Dec}(n, h, c)$
```

Figure 2.8: $\mathsf{CTI\text{-}CCA}$ experiment for an $\mathsf{AEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$ scheme. Note that we assume "nonce-respecting", which means the encryption oracle cannot be repeatedly queried with same nonces.

Definition 17. *We say an* $\mathsf{AEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$ *is* ϵ-$\mathsf{CTI\text{-}CCA}$ *secure, if the below defined advantage of any attacker* $\mathcal{A}$ *against* $\mathrm{Expr}^{\mathsf{CTI\text{-}CCA}}_{\mathsf{AEAD}}$ *experiment in Figure 2.8 is bounded by:*

$$\mathsf{Adv}^{\mathsf{CTI\text{-}CCA}}_{\mathsf{AEAD}} := \Pr[\mathrm{Expr}^{\mathsf{CTI\text{-}CCA}}_{\mathsf{AEAD}}(\mathcal{A}) = 1] \leq \epsilon$$

Other than privacy and integrity, Bellare and Hoang [27] observe that the commitment is an essential security property for AEAD.

Definition 18. *We say an* $\mathsf{AEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$ *is* ϵ-$\mathsf{CMT}\text{-}l$ *secure for* $l \in \{1, 3, 4\}$*, if the below defined advantage of any attacker* $\mathcal{A}$ *against* $\mathrm{Expr}^{\mathsf{CMT}\text{-}l}_{\mathsf{AEAD}}$ *experiment in Figure 2.9 is bounded by:*

$$\mathsf{Adv}^{\mathsf{CMT}\text{-}l}_{\mathsf{AEAD}} := \Pr[\mathrm{Expr}^{\mathsf{CMT}\text{-}l}_{\mathsf{AEAD}}(\mathcal{A}) = 1] \leq \epsilon$$

Definition 19. *We say an* $\mathsf{AEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$ *is* ϵ-$\mathsf{CMTD}\text{-}l$ *secure for* $l \in \{1, 3, 4\}$*, if the below defined advantage of any attacker* $\mathcal{A}$ *against* $\mathrm{Expr}^{\mathsf{CMT}\text{-}l}_{\mathsf{AEAD}}$ *experiment in Figure 2.9 is bounded by:*

$$\mathsf{Adv}^{\mathsf{CMTD}\text{-}l}_{\mathsf{AEAD}} := \Pr[\mathrm{Expr}^{\mathsf{CMTD}\text{-}l}_{\mathsf{AEAD}}(\mathcal{A}) = 1] \leq \epsilon$$

```
$\mathrm{Expr}^{\mathsf{CMT}\text{-}l}_{\mathsf{AEAD}}$:
$\big((k_1, n_1, h_1, m_1), (k_2, n_2, h_2, m_2)\big) \xleftarrow{\$} \mathcal{A}()$
if $\perp \in \{k_1, n_1, h_1, m_1, k_2, n_2, h_2, m_2\}$
    return 0
if $\mathsf{WiC}_l(k_1, n_1, h_1, m_1) = \mathsf{WiC}_l(k_2, n_2, h_2, m_2)$
    return 0
$c_1 \leftarrow \mathsf{AEAD.Enc}(k_1, n_1, h_1, m_1)$
$c_2 \leftarrow \mathsf{AEAD.Enc}(k_2, n_2, h_2, m_2)$
return $[\![c_1 = c_2]\!]$
```

```
$\mathrm{Expr}^{\mathsf{CMTD}\text{-}l}_{\mathsf{AEAD}}$:
$\big(c, (k_1, n_1, h_1, m_1), (k_2, n_2, h_2, m_2)\big) \xleftarrow{\$} \mathcal{A}()$
if $\perp \in \{k_1, n_1, h_1, m_1, k_2, n_2, h_2, m_2\}$
    return 0
if $\mathsf{WiC}_l(k_1, n_1, h_1, m_1) = \mathsf{WiC}_l(k_2, n_2, h_2, m_2)$
    return 0
$m_1' \leftarrow \mathsf{AEAD.Dec}(k_1, n_1, h_1, c)$
$m_2' \leftarrow \mathsf{AEAD.Dec}(k_2, n_2, h_2, c)$
return $[\![m_1 = m_1']\!]$ and $[\![m_2 = m_2']\!]$
```

l	1	3	4
$\mathsf{WiC}_l(k, n, h, m)$	k	(k, n, h)	(k, n, h, m)

Figure 2.9: CMT-l and CMTD-l experiments for an AEAD = (AEAD.KGen, AEAD.Enc, AEAD.Dec) scheme, where $l \in \{1, 3, 4\}$.

In the literature, there are a number of other related strong security notions. The *key committing* KC security [10] says that different keys but same nonce indicate the different ciphertexts. The multi-key collision resistance (MKCR) [135] says that no attacker can forge any nonce-header-ciphertext tuple (n, h, c) that can be decrypted to a valid message under any key from a attacker-chosen key space with certain minimal cardinality.

Definition 20. *We say an* AEAD = (AEAD.KGen, AEAD.Enc, AEAD.Dec) *has* (q, ϵ)*-key commitment (or is* (q, ϵ)-KC *secure), if the below defined advantage of any attacker* $\mathcal{A}$ *against* $\mathrm{Expr}^{\mathsf{KC}}_{\mathsf{AEAD},q}$ *experiment in Figure 2.10 is bounded by:*

$$\mathsf{Adv}^{\mathsf{KC}}_{\mathsf{AEAD},q} := \Pr[\mathrm{Expr}^{\mathsf{KC}}_{\mathsf{AEAD},q}(\mathcal{A}) = 1] \leq \epsilon$$

In particular, we say AEAD *is* ϵ-KC *for short, if* AEAD *is* (q, ϵ)-KC *secure for any* $q \geq 2$.

Definition 21. *We say an* AEAD = (AEAD.KGen, AEAD.Enc, AEAD.Dec) *with key space* $\mathcal{K}$ *has* (κ, ϵ)*-multi-key collision resistance (or is* (κ, ϵ)-MKCR *secure) for some parameter* $\kappa > 1$*, if the below defined advantage of any attacker* $\mathcal{A}$ *against* $\mathrm{Expr}^{\mathsf{MKCR}}_{\mathsf{AEAD},\kappa}$ *experiment in Figure 2.11 is bounded by:*

$$\mathsf{Adv}^{\mathsf{MKCR}}_{\mathsf{AEAD},\kappa} := \Pr[\mathrm{Expr}^{\mathsf{MKCR}}_{\mathsf{AEAD},\kappa}(\mathcal{A}) = 1] \leq \epsilon$$

In particular, we say AEAD *is* ϵ-MKCR *for short, if* AEAD *is* (κ, ϵ)-MKCR *secure for* $\kappa = 2$.

Definition 22. *An* AEAD *can be extended to (compactly) commiting* AEAD (ccAEAD) *if two additional algorithms are defined. Let* $\mathcal{VK}$ *denote the space of verification key.*

- AEAD.openCommit *the open commitment algorithm inputs a key* $k \in \mathcal{K}$*, a nonce* $n \in \mathcal{N}$*, a header* $h \in \mathcal{H}$*, and a ciphertext* $c \in \mathcal{CT}$ *and (deterministically) outputs a verification key* $k_f \in \mathcal{VK} \cup \{\perp\}$*, i.e.,* $k_f \leftarrow \mathsf{AEAD.openCommit}(k, n, h, c)$

```
$\mathrm{Expr}^{\mathsf{KC}}_{\mathsf{AEAD},q}$:
$\mathcal{L} \leftarrow \emptyset,\ n \leftarrow 0$
$() \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Enc}},\mathcal{O}_{\mathsf{Dec}}}()$
**foreach** $(k_1, n_1, h_1, m_1, c_1), (k_2, n_2, h_2, m_2, c_2) \in \mathcal{L}$
  **if** $k_1 \neq k_2$ **and** $n_1 = n_2$ **and** $c_1 = c_2 \neq \bot$ **and** $m_1 \neq \bot$ **and** $m_2 \neq \bot$
    **return** $1$
**return** $0$

$\mathcal{O}_{\mathsf{Enc}}(k, n, h, m)$:
$c \leftarrow \mathsf{AEAD.Enc}(k, n, h, m)$
**if** $n < q$
  $\mathcal{L} \leftarrow \mathcal{L} \cup (k, n, h, m, c)$
  $n \leftarrow n + 1$
**return** $c$

$\mathcal{O}_{\mathsf{Dec}}(k, n, h, c)$:
$m \leftarrow \mathsf{AEAD.Dec}(k, n, h, c)$
**if** $n < q$
  $\mathcal{L} \leftarrow \mathcal{L} \cup (k, n, h, m, c)$
  $n \leftarrow n + 1$
**return** $m$
```

Figure 2.10: KC experiment for an AEAD = (AEAD.KGen, AEAD.Enc, AEAD.Dec) scheme.

```
$\mathrm{Expr}^{\mathsf{MKCR}}_{\mathsf{AEAD},\kappa}$:
$(\mathcal{K}^\star, n^\star, h^\star, c^\star) \xleftarrow{\$} \mathcal{A}()$
**if** $|\mathcal{K}^\star| < \kappa$
  **return** $0$
**foreach** $k \in \mathcal{K}^\star$
  **if** $\mathsf{AEAD.Dec}(k, n^\star, h^\star, c^\star) = \bot$
    **return** $0$
**return** $1$
```

Figure 2.11: MKCR experiment for an AEAD = (AEAD.KGen, AEAD.Enc, AEAD.Dec) scheme.

- AEAD.vrfyCommit *the commitment verification algorithm inputs a verification key* $k_f \in \mathcal{VK}$ *, a nonce* $n \in \mathcal{N}$*, a header* $h \in \mathcal{H}$*, a message* m*, and a ciphertext* $c \in \mathcal{CT}$ *and (deterministically) outputs a boolean value* $v \in \{\mathsf{true}, \mathsf{false}\}$*, i.e.,* $v \leftarrow \mathsf{AEAD.vrfyCommit}(k_f, n, h, m, c)$*.*

In the realm of ccAEAD, there are two important notions: sender binding s-BIND and receiver binding r-BIND [97]. While the s-BIND property ensures that the attacker cannot forge any $(k,\ h,\ c)$ tuple such that the decrypted message can be verified using the opened verification key. The r-BIND ensures that each ciphertext is bound to the same nonce-header-message tuple.

Definition 23. *We say an* ccAEAD = (AEAD.KGen, AEAD.Enc, AEAD.Dec, AEAD.openCommit, AEAD.vrfyCommit) *has* ϵ*-sender binding (or is* ϵ-s-BIND *secure), if the below defined advantage of any attacker* $\mathcal{A}$ *against* $\mathrm{Expr}^{\mathsf{s\text{-}BIND}}_{\mathsf{AEAD}}$ *experiment in Figure 2.12 is bounded by:*

$$\mathsf{Adv}^{\mathsf{s\text{-}BIND}}_{\mathsf{ccAEAD}} := \Pr[\mathrm{Expr}^{\mathsf{s\text{-}BIND}}_{\mathsf{ccAEAD}}(\mathcal{A}) = 1] \leq \epsilon$$

Definition 24. *We say an* $\mathsf{ccAEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec}, \mathsf{AEAD.openCommit}, \mathsf{AEAD.vrfyCommit})$ *has* ϵ*-receiver binding (or is* ϵ*-*r-BIND *secure), if the below defined advantage of any attacker* $\mathcal{A}$ *against* $\mathrm{Expr}^{\mathsf{r\text{-}BIND}}_{\mathsf{AEAD}}$ *experiment in Figure 2.13 is bounded by:*

$$\mathsf{Adv}^{\mathsf{r\text{-}BIND}}_{\mathsf{ccAEAD}} := \Pr[\mathrm{Expr}^{\mathsf{r\text{-}BIND}}_{\mathsf{ccAEAD}}(\mathcal{A}) = 1] \leq \epsilon$$

```
$\mathrm{Expr}^{\mathsf{s\text{-}BIND}}_{\mathsf{ccAEAD}}$:
$(k, n, h, c) \xleftarrow{\$} \mathcal{A}()$
$m \leftarrow \mathsf{AEAD.Dec}(k, n, h, c)$
$k_f \leftarrow \mathsf{AEAD.openCommit}(k, n, h, c)$
if $m = \perp$
  return $0$
if $\mathsf{AEAD.vrfyCommit}(k_f, n, h, m, c)$
  return $0$
return $1$
```

Figure 2.12: s-BIND experiment for an $\mathsf{ccAEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec}, \mathsf{AEAD.openCommit}, \mathsf{AEAD.vrfyCommit})$ scheme.

```
$\mathrm{Expr}^{\mathsf{r\text{-}BIND}}_{\mathsf{ccAEAD}}$:
$\Big(c, (k_{f1}, n_1, h_1, m_1), (k_{f2}, n_2, h_2, m_2)\Big) \xleftarrow{\$} \mathcal{A}()$
if $\perp \in \{k_1, n_1, h_1, m_1, k_2, n_2, h_2, m_2\}$
  return $0$
if $(h_1, m_1) = (h_2, m_2)$
  return $0$
if $\mathsf{AEAD.vrfyCommit}(k_{f1}, n_1, h_1, m_1, c)$ and $\mathsf{AEAD.vrfyCommit}(k_{f2}, n_2, h_2, m_2, c)$
  return $1$
return $0$
```

Figure 2.13: r-BIND experiment for an $\mathsf{ccAEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec}, \mathsf{AEAD.openCommit}, \mathsf{AEAD.vrfyCommit})$ scheme.

From an $\mathsf{AEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$, we can easily extend to an $\mathsf{ccAEAD}[\mathsf{AEAD}] = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec}, \mathsf{AEAD.openCommit}, \mathsf{AEAD.vrfyCommit})$ using "traditionally committing encryption" approach [97], such that

$$\mathsf{AEAD.openCommit}(k, n, h, c) := k$$
$$\mathsf{AEAD.vrfyCommit}(k, n, h, m, c) := [\![m = \mathsf{AEAD.Dec}(k, n, h, c)]\!]$$

2.2.5 Digital Signature Schemes

Definition 25 (Digital Signature scheme)**.** *A digital signature (DS) scheme* $\mathsf{DS} = (\mathsf{DS.KGen}, \mathsf{DS.Sign}, \mathsf{DS.Vrfy})$ *over message space* $\mathcal{M}$ *and randomness space* $\mathcal{R}$ *is a tuple of algorithms as defined below.*

Key Generation $(vk, sk) \xleftarrow{\$} \mathsf{DS.KGen}(\mathsf{pp})$*: inputs the public parameter* pp *and outputs a public verification and private signing key pair* (vk, sk)*.*

Signing $\sigma \xleftarrow{\$} \mathsf{DS.Sign}(sk, m)$*: inputs a signing key sk and a message* $m \in \mathcal{M}$ *and outputs a signature* σ*. We write* $\sigma \xleftarrow{\$} \mathsf{DS.Sign}(sk, m; r^{\mathsf{Sign}})$*, if the random coins* $r^{\mathsf{Sign}} \in \mathcal{R}$ *is specified.*

Verification $\mathsf{true}/\mathsf{false} \leftarrow \mathsf{DS.Vrfy}(vk, m, \sigma)$*: inputs a verification key vk, a message m, and a signature* σ *and outputs a boolean value either true* true *or* false*. Note that we sometimes also denote* true *by* 1 *and* false *by* 0*.*

We say a DS is δ-correct if for every $(vk, sk) \xleftarrow{\$} \mathsf{DS.KGen}()$ and every message $m \in \mathcal{M}$, we have

$$\Pr[\mathsf{false} \leftarrow \mathsf{DS.Vrfy}(vk, m, \mathsf{DS.Sign}(sk, m))] \leq \delta$$

In particular, we call a DS *(perfectly) correct* if $\delta = 0$.

We say a DS is δ-strongly correct if for every $(vk, sk) \xleftarrow{\$} \mathsf{DS.KGen}()$, every message $m \in \mathcal{M}$, and every $r^{\mathsf{Sign}} \in \mathcal{R}$ we have

$$\Pr[\mathsf{false} \leftarrow \mathsf{DS.Vrfy}(vk, m, \mathsf{DS.Sign}(sk, m; r^{\mathsf{Sign}}))] \leq \delta$$

Compared to the conventional correctness, the strong correctness requires that the signed message-signature pair be correctly verified for every randomness coins involved during the signing. In particular, we call a DS *(perfectly) strongly correct* if $\delta = 0$.

The standard notion for security of signature schemes is that of (single-user) *existential unforgeability under chosen message attacks.* Intuitively, this guarantees that for a fixed public verification key, an attacker $\mathcal{A}$ cannot generate a valid signature on a new message, for which it has not seen a valid signature before. A stronger definition of security is that of (single-user) *strong unforgeability*, which will also play a role later. Here, the attacker is not restricted to forging signatures on new messages for a fixed public key but may also generate a signature on a message on which it has seen (other) signatures. Both of these notions can then be transferred to the multi-user setting, where there is not just a single public key generated by the challenger but multiple honestly generated keys. The attacker's goal is then to (existentially or strongly) forge a signature under *any* of these keys.

Definition 26 ($\mathsf{EUF\text{-}CMA}$ and $\mathsf{SUF\text{-}CMA}$ security)**.** *Let* $\mathsf{DS} = (\mathsf{DS.KGen}, \mathsf{DS.Sign}, \mathsf{DS.Vrfy})$ *be a digital signature scheme. Consider the security experiments* $\mathrm{Expr}_{\mathsf{DS}}^{\mathsf{euf\text{-}cma}}$ *and* $\mathrm{Expr}_{\mathsf{DS}}^{\mathsf{suf\text{-}cma}}$ *as defined in Figure 2.14. We say that a digital signature scheme* DS *is* (t, ϵ, Q_S)-$\mathsf{EUF\text{-}CMA}$-*secure or* existentially unforgeable under chosen message attacks, *if for any attacker* $\mathcal{A}$ *running in time at most t, making at most* Q_S *queries to the signing oracle, the advantage*

$$\mathsf{Adv}_{\mathsf{DS}}^{\mathsf{EUF\text{-}CMA}}(\mathcal{A}) := \Pr[\mathrm{Expr}_{\mathsf{DS}}^{\mathsf{EUF\text{-}CMA}}(\mathcal{A}) = 1] \leq \epsilon$$

```
$\mathrm{Expr}_{\mathsf{DS}}^{\mathsf{euf\text{-}cma}}(\mathcal{A})$:
    $\mathcal{L}_{\mathsf{Sign}} \leftarrow \emptyset$
    $(vk, sk) \xleftarrow{\$} \mathsf{DS.KGen}(\mathsf{pp})$
    $(m^\star, \sigma^\star) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}}(vk)$
    return $[\![\mathsf{DS.Vrfy}(vk, \sigma^\star, m^\star) \wedge m^\star \notin \mathcal{L}_{\mathsf{Sign}}]\!]$
$\mathcal{O}_{\mathsf{Sign}}(m)$:
    $\sigma \xleftarrow{\$} \mathsf{DS.Sign}(sk, m)$
    $\mathcal{L}_{\mathsf{Sign}} \leftarrow \mathcal{L}_{\mathsf{Sign}} \cup \{m\}$
    return $\sigma$
```

```
$\mathrm{Expr}_{\mathsf{DS}}^{\mathsf{suf\text{-}cma}}(\mathcal{A})$:
    $\mathcal{L}_{\mathsf{Sign}} \leftarrow \emptyset$
    $(vk, sk) \xleftarrow{\$} \mathsf{DS.KGen}(\mathsf{pp})$
    $(m^\star, \sigma^\star) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}}(vk)$
    return $[\![\mathsf{DS.Vrfy}(vk, \sigma^\star, m^\star) \wedge (m^\star, \sigma^\star) \notin \mathcal{L}_{\mathsf{Sign}}]\!]$
$\mathcal{O}_{\mathsf{Sign}}(m)$:
    $\sigma \xleftarrow{\$} \mathsf{DS.Sign}(sk, m)$
    $\mathcal{L}_{\mathsf{Sign}} \leftarrow \mathcal{L}_{\mathsf{Sign}} \cup \{(m, \sigma)\}$
    return $\sigma$
```

Figure 2.14: EUF-CMA experiment (left) and SUF-CMA experiment (right) of a $\mathsf{DS} = (\mathsf{DS.KGen}, \mathsf{DS.Sign}, \mathsf{DS.Vrfy})$ scheme with differences highlighted in gray.

Analogously, we say that DS *is* (t, ϵ, Q_S)-SUF-CMA-*secure or* strongly unforgeable under chosen message attacks, *if the advantage*

$$\mathsf{Adv}_{\mathsf{DS}}^{\mathsf{SUF\text{-}CMA}}(\mathcal{A}) := \Pr[\mathrm{Expr}_{\mathsf{DS}}^{\mathsf{SUF\text{-}CMA}}(\mathcal{A}) = 1] \le \epsilon$$

For simplicity, we sometimes omit the measure on the running time and the total number of queries, if they are only required to be in polynomial with respect to the implicit security parameter pp.

Definition 27 (Multi-User MU-EUF-CMA and MU-SUF-CMA security)**.** *Let* $\mathsf{DS} = (\mathsf{DS.KGen}, \mathsf{DS.Sign}, \mathsf{DS.Vrfy})$ *be a digital signature scheme. Consider the security games* $\mathrm{Expr}_{\mathsf{DS}}^{\mathsf{mu\text{-}euf\text{-}cma}}$ *and* $\mathrm{Expr}_{\mathsf{DS}}^{\mathsf{mu\text{-}suf\text{-}cma}}$ *as defined in Figure 2.15. We say that a digital signature scheme* DS *is* (t, ϵ, N, Q_S)-MU-EUF-CMA-*secure or* multi-user existentially unforgeable under chosen message attacks, *if for any* PPT *attacker* $\mathcal{A}$ *running in time at most* t*, making at most* Q_S *queries to the signing oracle, given* N *public keys, the probability*

$$\Pr\big[\mathrm{Expr}_{\mathsf{DS}}^{\mathsf{mu\text{-}euf\text{-}cma}}(\mathcal{A}, N) = 1\big] \le \epsilon.$$

Analogously, we say that a signature scheme is (t, ϵ, N, Q_S)-MU-SUF-CMA-*secure or* multi-user strongly unforgeable under chosen message attacks.

For simplicity, we sometimes omit the measure on the running time and the total number of queries, if they are only required to be in polynomial with respect to the implicit security parameter pp.

2.2.6 Key Encapsulation Mechanisms

Definition 28. *A key encapsulation mechanism scheme over (secret) decapsulation key space* $\mathcal{DK}$*, (public) encapsulation key space* $\mathcal{EK}$*, randomness space* $\mathcal{R}$*, symmetric key space* $\mathcal{K}$*, and ciphertext space* $\mathcal{CT}$*, is a tuple of algorithms* $\mathsf{KEM} = (\mathsf{KEM.KGen}, \mathsf{KEM.Encaps}, \mathsf{KEM.Decaps})$ *as defined below.*

```
$\mathrm{Expr}^{\text{mu-euf-cma}}_{\mathsf{DS}}(\mathcal{A}, N)$:
$\mathcal{L}_{\mathsf{Sign}} \leftarrow \emptyset$
$\textbf{for } i = 1, \ldots, N$
    $(vk_i, sk_i) \xleftarrow{\$} \mathsf{DS.KGen}(\mathsf{pp})$
$(i^\star, \sigma^\star, m^\star) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}}(vk_1, \ldots, vk_N)$
$\textbf{return } [\![\mathsf{DS.Vrfy}(vk^\star_i, \sigma^\star, m^\star) \wedge (i^\star, m^\star) \notin \mathcal{L}_{\mathsf{Sign}}]\!]$

$\mathcal{O}_{\mathsf{Sign}}(i, m)$:
$\sigma \xleftarrow{\$} \mathsf{DS.Sign}(sk_i, m)$
$\mathcal{L}_{\mathsf{Sign}} \leftarrow \mathcal{L}_{\mathsf{Sign}} \cup \{(i, m)\}$
$\textbf{return } \sigma$
```

```
$\mathrm{Expr}^{\text{mu-suf-cma}}_{\mathsf{DS}}(\mathcal{A}, N)$:
$\mathcal{L}_{\mathsf{Sign}} \leftarrow \emptyset$
$\textbf{for } i = 1, \ldots, N$
    $(vk_i, sk_i) \xleftarrow{\$} \mathsf{DS.KGen}(\mathsf{pp})$
$(i^\star, \sigma^\star, m^\star) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}}(vk_1, \ldots, vk_N)$
$\textbf{return } [\![\mathsf{DS.Vrfy}(vk^\star_i, \sigma^\star, m^\star) \wedge (i^\star, m^\star, \sigma^\star) \notin \mathcal{L}_{\mathsf{Sign}}]\!]$

$\mathcal{O}_{\mathsf{Sign}}(i, m)$:
$\sigma \xleftarrow{\$} \mathsf{DS.Sign}(sk_i, m)$
$\mathcal{L}_{\mathsf{Sign}} \leftarrow \mathcal{L}_{\mathsf{Sign}} \cup \{(i, m, \sigma)\}$
$\textbf{return } \sigma$
```

Figure 2.15: Multi-user MU-EUF-CMA experiment (top) and MU-SUF-CMA experiment (bottom) of a DS = (DS.KGen, DS.Sign, DS.Vrfy) scheme with differences to the single-user security notions highlighted in gray for the EUF-CMA case.

- ***Key Generation*** $(ek, dk) \xleftarrow{\$} \mathsf{KEM.KGen}(\mathsf{pp})$*: takes as input the public parameter* pp *and outputs a public-secret key pair* $(ek, dk) \in \mathcal{EK} \times \mathcal{DK}$ *.*
- ***Encapsulation*** $(c, k) \xleftarrow{\$} \mathsf{KEM.Encaps}(ek)$*: takes as input a public key* $ek \in \mathcal{EK}$ *and outputs a ciphertext* $c \in \mathcal{CT}$ *and a symmetric key* $k \in \mathcal{K}$*. We write* $(c, k) \xleftarrow{\$} \mathsf{KEM.Encaps}(ek; r^{\mathsf{Encaps}})$ *if the random coins* $r^{\mathsf{Encaps}} \in \mathcal{R}$ *is specified.*
- ***Decapsulation*** $k \leftarrow \mathsf{KEM.Decaps}(dk, c)$*: takes as input a secret key* $dk \in \mathcal{DK}$ *and a ciphertext* $c \in \mathcal{CT}$ *and outputs either a symmetric key* $k \in \mathcal{K}$ *or an error symbol* $\perp$*.*

We say a KEM = (KEM.KGen, KEM.Encaps, KEM.Decaps) is δ-correct if for every $(ek, dk) \xleftarrow{\$} \mathsf{KEM.KGen}()$, we have

$$\Pr[k \neq \mathsf{KEM.Decaps}(dk, c) : (c, k) \xleftarrow{\$} \mathsf{KEM.Encaps}(ek)] \leq \delta.$$

In particular, we call a KEM *(perfectly) correct* if $\delta = 0$.

We say a KEM is δ-strongly correct if for every $(ek, dk) \xleftarrow{\$} \mathsf{KEM.KGen}()$ and every $r^{\mathsf{Encaps}} \in \mathcal{R}$, we have

$$\Pr[k \neq \mathsf{KEM.Decaps}(dk, c) : (c, k) \xleftarrow{\$} \mathsf{KEM.Encaps}(ek; r^{\mathsf{Encaps}})] \leq \delta$$

Compared to the conventional correctness, the strong correctness requires that the encapsulate keys can be correctly recovered for every randomness coins involved during the encapsulation. In particular, we call a KEM *(perfectly) strongly correct* if $\delta = 0$.

We define the min-entropy α_{ek} of public keys ek and α_c of the ciphertext c by

$$\alpha_{ek} := -\log \max_{ek' \in \mathcal{EK}} \Pr[ek' = ek : (ek, dk) \xleftarrow{\$} \mathsf{KEM.KGen}()]$$
$$\alpha_c := -\log \max_{c' \in \mathcal{CT}} \Pr_{(ek,dk) \xleftarrow{\$} \mathsf{KEM.KGen}()} [c = c' : (c, k) \xleftarrow{\$} \mathsf{KEM.Encaps}(ek)]$$

```
$\underline{\mathrm{Expr}^{\mathsf{IND\text{-}CPA}}_{\mathsf{KEM}}(\mathcal{A})}$:
$\mathsf{b} \xleftarrow{\$} \{0,1\}$
$(ek, dk) \xleftarrow{\$} \mathsf{KEM.KGen}()$
$(c^\star, k_0^\star) \xleftarrow{\$} \mathsf{KEM.Encaps}(ek)$
$k_1^\star \xleftarrow{\$} \mathcal{K}$
$\mathsf{b}' \xleftarrow{\$} \mathcal{A}(ek, c^\star, k_{\mathsf{b}}^\star)$
**return** $[\![\mathsf{b} = \mathsf{b}']\!]$
```

```
$\underline{\mathrm{Expr}^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}}(\mathcal{A})}$:
$\mathsf{b} \xleftarrow{\$} \{0,1\}$
$(ek, dk) \xleftarrow{\$} \mathsf{KEM.KGen}()$
$(c^\star, k_0^\star) \xleftarrow{\$} \mathsf{KEM.Encaps}(ek)$
$k_1^\star \xleftarrow{\$} \mathcal{K}$
$\mathsf{b}' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Decaps}}}(ek, c^\star, k_{\mathsf{b}}^\star)$
**return** $[\![\mathsf{b} = \mathsf{b}']\!]$
```

```
$\underline{\mathcal{O}_{\mathsf{Decaps}}(c)}$:
if $c = c^\star$
    **return** $\perp$
$k' \leftarrow \mathsf{KEM.Decaps}(dk, c)$
**return** $k'$
```

Figure 2.16: IND-CPA and IND-CCA experiments for a KEM = (KEM.KGen, KEM.Encaps, KEM.Decaps) scheme.

In terms of the security notions, we recall the standard *indistinguishability under chosen plaintext/ciphertext attacks* (IND-CPA/IND-CCA). The IND-CPA security prevents an attacker from distinguishing the encapsulated symmetric key of a challenge ciphertext from a random one. The IND-CCA security additionally allows the attacker to access a decapsulation oracle.

Definition 29. *Let* KEM = (KEM.KGen, KEM.Encaps, KEM.Decaps) *be a key encapsulation mechanism scheme with symmetric key space* $\mathcal{K}$. *We say* KEM *is* ϵ-IND-CPA *secure, if the below defined advantage of every (potential quantum) attacker* $\mathcal{A}$ *against* $\mathrm{Expr}^{\mathsf{IND\text{-}CPA}}_{\mathsf{KEM}}$ *experiment in Figure 2.16 is bounded by,*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{KEM}}(\mathcal{A}) := \left| \Pr[\mathrm{Expr}^{\mathsf{IND\text{-}CPA}}_{\mathsf{KEM}}(\mathcal{A}) = 1] - \frac{1}{2} \right| \leq \epsilon$$

Definition 30. *Let* KEM = (KEM.KGen, KEM.Encaps, KEM.Decaps) *be a key encapsulation mechanism scheme with symmetric space* $\mathcal{K}$. *We say* KEM *is* ϵ-IND-CCA *secure, if the below defined advantage of every (potential quantum) attacker* $\mathcal{A}$ *against* $\mathrm{Expr}^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}}$ *experiment in Figure 2.16 is bounded by,*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}}(\mathcal{A}) := \left| \Pr[\mathrm{Expr}^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}}(\mathcal{A}) = 1] - \frac{1}{2} \right| \leq \epsilon$$

2.3 The Random Oracle Methodology

This random oracle model was first introduced by Bellare and Rogaway [34] and enabled security proofs for many efficient schemes that previously had eluded the provable security paradigm. It does so by representing a hash function in a cryptographic scheme as an idealized random function (the *random oracle*). With this idealization in place, an attacker can *only* evaluate the hash function H on input x, if it queries this random oracle on x. It is no longer able to simply evaluate the hash function locally. In particular, this allows us to "peek" at the attacker's inputs to the hash function, a property of the model that is often referred to as *extractability*. When queried on input x, the random oracle then returns uniformly random answers from the range of H for each input. For each new query a fresh uniformly random output is sampled, but just like for real hash functions, repeated queries are answered *consistently*, i.e., the same inputs yield the same outputs. Another essential property of the random oracle is *programmability*: if the attacker queries some input x for the first time, we can set the value $\mathsf{H}(x)$ to a specific, freely-chosen output value y as long as it is correctly distributed and does not collide with previously set outputs.

Note that when our proofs are in the random oracle model for hash function H, the security notions introduced previously get the extra query parameter Q_{H} of the maximal number of queries the attacker makes to the random oracle.

Chapter 3

Provable Security of Ed25519

This chapter is based on the paper:

Jacqueline Brendel, Cas Cremers, Dennis Jackson, and Mang Zhao. ”The Provable Security of Ed25519: Theory and Practice,” 2021 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2021, pp. 1659-1676. DOI: 10.1109/SP40001.2021.00042.

This paper was joint work with my co-authors Jacqueline Brendel, Cas Cremers, and Dennis Jackson. All authors actively contributed to the completion of this work.

3.1 Introduction

The Ed25519 signature scheme was introduced in 2011 by Bernstein, Duif, Lange, Schwabe, and Yang in the paper "High-speed high-security signatures" [43]. The efficiency of the scheme has led to a global uptake in modern applications, and it is now used in SSH, Tor, ZCash, and messaging protocols based on the Signal protocol such as WhatsApp.

For modern digital signature schemes the quintessential property is *existential unforgeability under chosen message attacks* (EUF-CMA) [94]. This property basically requires that an attacker cannot construct a signature for a message that the key owner did not sign previously.

A stronger property that can be met by signature schemes, is that of *strong unforgeability under chosen message attacks* (SUF-CMA). This property additionally requires that the attacker cannot construct an alternative signature for a given signed message. A high-profile example exploiting the absence of this property is the Mt. Gox attack on Bitcoin [129]. In this attack, signatures on transactions were mauled by an adversarial recipient before being stored on the blockchain. The recipient would then claim that the transaction failed. The sender would check the blockchain, and would indeed not find their exact signature (due to mauling), conclude that it apparently failed, and start a new transfer. Yet both signatures would be valid and the recipient would thus receive the double amount. This attack would not have been possible with a strongly unforgeable (SUF-CMA) signature scheme as this notion prohibits malleability.

Additionally, in many practical systems, it is highly desirable that signature schemes resist key substitution attacks [117, 158]. In such attacks the attacker computes specific public keys, e.g., based on observed signatures of honest signers, such that these honest signatures can also be verified under the attacker's new public keys. This has been shown to lead to attacks on protocols such as Let's Encrypt Certificate Issuance and SOAP's WS-Security [117].

Surprisingly, *full proofs of any of these security properties have never been given for Ed25519*. The original publications [43, 44] focused on efficiency of computation, and do not contain a precise statement on the security property that is offered by the scheme, which in the following we will refer to as Ed25519-Original. Because the scheme is said to be constructed via the Fiat-Shamir transform, it should follow that Ed25519-Original at least provides EUF-CMA security, but full details were never provided. The papers do refer to malleability and argue that is not relevant for the standard definitions of signature scheme security, but their definition of malleability does not agree with the common usage of the term. It also transpires that, whilst the source code presented alongside the paper accepts mangled signatures (hence is not SUF-CMA), the additional check included in the paper's description *but not the source code*, is actually sufficient to prove SUF-CMA, as

```
DS.KGen(pp):
    $k \xleftarrow{\$} \{0,1\}^b$
    $h \leftarrow \mathsf{H}(k)$
    $s \leftarrow 2^n + \sum_{i=c}^{n-1} 2^i h[i]$
    $A \leftarrow sB$
    return $(\underline{A}, k)$

DS.Sign$(k, m)$:
    $h \leftarrow \mathsf{H}(k)$
    $s \leftarrow 2^n + \sum_{i=c}^{n-1} 2^i h[i]$
    $r \leftarrow \mathsf{H}(h[b], \ldots, h[2b-1], m)$
    $R \leftarrow rB$
    $S \leftarrow (r + \mathsf{H}(\underline{R}, \underline{A}, m)s) \mod L$
    return $\sigma = (\underline{R}, \underline{S})$

DS.Vrfy$(\underline{A}, \sigma = (\underline{R}, \underline{S}), m)$:
    Check $R, A \in E$
    Variant Specific Checks
    return Checks succeed $\wedge$ $[\![2^c SB = 2^c R + 2^c \mathsf{H}(\underline{R}, \underline{A}, m)A]\!]$
```

Figure 3.1: Generic description of the Ed25519 signature scheme algorithms DS = (DS.KGen, DS.Sign, DS.Vrfy). Note that the highlighted Line 13 varies depending on the version of Ed25519 and the appropriate check is listed in Table 3.1.

Ref	Scheme	Variant Specific Checks	EUF-CMA (Thm. 4)	SUF-CMA (Thm. 5)	S-UEO (Thm. 6)	MBS (Thm. 7)	M-S-UEO (Thm. 8)
[42]	Ed25519-Original	$S \in \{0, ..., 2^b - 1\}$	✓	×	✓	×	×
[121]	Ed25519-IETF	$S \in \{0, ..., L - 1\}$	✓	✓	✓	×	×
[119]	Ed25519-LibS	$S \in \{0, ..., L-1\} \wedge \lvert R \rvert \geqslant L \wedge \lvert A \rvert \geqslant L$	✓	✓	✓	✓	✓

Table 3.1: Ed25519 Schemes. To form each variant, replace the highlighted section in Figure 3.1 with the text presented here. Note that $2^b - 1 > L$ (see Table 3.2) so the latter checks are stricter. EUF-CMA: Existential Unforgeability; SUF-CMA: Strong Unforgeability; S-UEO and M-S-UEO denote resilience against key substitution attacks; MBS: Message Bound Security, ensuring a signature verifies a unique message, even for malicious keys.

we will show. This further adds to the confusion around the security properties enjoyed by Ed25519.

While the original papers came with a full implementation of Ed25519-Original, later implementations made various modifications. Notably, the Ed25519-IETF version that was standardized by the Internet Engineering Task Force (IETF) in [121] includes a check that is claimed to prevent malleability, thereby implicitly suggesting that Ed25519-IETF is strongly unforgeable (SUF-CMA). Later versions, such as the ones used by LibSodium [119] and ZCash [108] included additional group element checks. We return to the details of these differences in Section Section 3.4.1. This leads to the obvious question: which exact properties are actually provided by the various Ed25519 schemes? This question is especially timely as Ed25519 is currently proposed for inclusion in the USA's National Institute of Standards and Technology (NIST) standard for Digital Signature Schemes [145, 146, 147] and was recently included in the TLS 1.3 standard [161].

Over the last years, several published works reported security proofs of systems that use Ed25519, but require specific cryptographic security notions from their signature

Parameter	Description	Instantiation for Ed25519
q	The finite field size q	$2^{255} - 19$
n	Secret scalars are $n + 1$-bits	254
b	The public key bit-length $2^{b-1} > q$	256
a, d	Curve parameters in $\mathbb{F}_q$	$-1, -121665/121666$
B	A generator of the prime order subgroup of E	$(x, 4/5), x > 0$
c	The $\log_2$ of curve cofactor	$c = 3$
L	The prime generator order $LB = 0$ and $2^c L = \|E\|$	$2^{252} + 27742 \ldots 8493$
H	Secure hash function producing $2b$-bit output	SHA-512
EC	Curve equation in Twisted Edwards form	$x^2 + y^2 = 1 + dx^2y^2$

Table 3.2: Parameters for Ed25519 signatures as described in Figure 3.1.

schemes, such as computational proofs of TLS 1.3's properties [47, 48, 84, 131]. These proofs assume that Ed25519-IETF satisfies EUF-CMA, leaving a proof gap. A claimed computational proof of SSH [37] requires that all supported signature schemes provide SUF-CMA. However, SSH implementations also allow the use of the malleable Ed25519-Original, which leads to a counterexample to the security statement. The Signal protocol library implements yet another variant of the Ed25519 signature scheme. Adding to the confusion, a recent work [100] claims that the results on Schnorr signatures in prime order groups implies Ed25519-Original enjoys SUF-CMA and resistance to key substitution attacks. We will see in Section 3.5.4 that this is not the case.

In this work we remedy these proof gaps, and establish further properties of Ed25519 signature schemes. As we will see, the devil is in the detail: the specific details of Ed25519 (e.g., small subgroup elements, scalar clamping) require subtle tailoring of the proof details, and lead to mismatches such as the lack of SUF-CMA security for Ed25519-Original despite its similarities to Fiat-Shamir applied to the Schnorr identification scheme, which 'should' imply that SUF-CMA security holds. These subtleties also manifest themselves in the requirements of various checks. Thus, our work not only fills these highly-needed proof gaps, but also provides additional insight into the subtle differences between Ed25519 schemes which are summarized in Table 3.1.

Our main contributions are the following.

- We provide the first detailed proof that Ed25519-Original [43] is indeed EUF-CMA secure.
- We provide the first proof that Ed25519-IETF [121] is actually SUF-CMA secure.
- We prove that all Ed25519 schemes are resilient against key substitution attacks, and that if small subgroup keys are rejected as in LibSodium, a signature uniquely identifies a message, even for malicious keys.
- In a wider sense, our results retroactively support the standardisation of Ed25519-IETF, and support the ongoing standardisation by NIST.

Overview: In Section 3.2 we present related work. Section 3.4 presents Ed25519 signature schemes and their subtle differences from Schnorr identification schemes. We provide the security analysis in Section 3.5 and we conclude in Section 3.6. In Section 3.7, we give the full proofs for all lemmas and theorems in this chapter.

3.2 Related Work

3.2.1 History of EdDSA and Ed25519

Ed25519-Original is just one instantiation of the more general EdDSA signature scheme, which was introduced in the same paper [43, 44]. EdDSA is itself a variant of the well-known Schnorr signature scheme [166, 167]. Ed25519 is EdDSA instantiated over curve Edwards25519 [43] and remains by far the most popular instantiation of EdDSA, despite its later extension to support alternative curves [45, 121].

EdDSA instantiations such as Ed25519-Original can sign and verify signatures substantially faster than almost all other signatures schemes at similar security levels. For schemes that have comparable speeds, Ed25519-Original further provides considerably smaller signatures, producing 64-byte signatures and 32-byte public keys. Additionally, EdDSA is widely considered to provide better resistance to side-channel attacks than alternative schemes. However, the original papers [43, 44] contain no formal statements (and consequently, no actual proofs) of its security properties.

By virtue of its outstanding performance with respect to efficiency and bandwidth, EdDSA was standardised by the IETF between 2015 and 2017 [121]. In 2019, EdDSA was proposed to also be adopted as part of NIST's Digital Signature Standard (DSS) [145, 146]. In early 2020, the public call for comments was closed [147], but as of writing, no new version has appeared.

3.2.2 Related Proofs

The Fiat-Shamir paradigm was proposed by Fiat and Shamir [90] as a generic approach to derive a secure signature scheme from a canonical identification schemes (CID). A vast body of work followed this seminal result and the aforementioned Schnorr signatures [166, 167], on which EdDSA was built, is probably one of the most famous examples of the transform's power to build efficient and provably-secure signatures. Here we merely present some of the many milestones related to Fiat-Shamir that are most relevant for our work. While the original presentation [90] lacked security proofs, Pointcheval and Stern [156, 157] closed this gap by providing proofs in the (then) relatively new random oracle model [34]. Abdalla et al. [1] indicated the minimal conditions for the underlying identification scheme to prove Fiat-Shamir transformed signatures to be EUF-CMA secure. In 2016, Kiltz et al.

[125, 126] provided a concrete and modular security analysis of Fiat-Shamir signatures in both the single-user and multi-user setting, closing the tightness gap of the reduction.

The treatment of the multi-user setting is especially interesting as in practical applications there exist many different public keys for an attacker to attack. In 2002, Galbraith, Malone-Lee, and Smart [92] considered security of signatures in this multi-user setting. They showed that if an attacker were to attack N keys at once, its advantage can increase only at most by a factor of N (this is often referred to as the *generic bound*). Their second result claimed that for *Schnorr-like* signatures one can do even better and achieve a tight reduction between single-user and multi-user security. Much later, Bernstein [40] exposed a flaw in this tight proof that to this date could not be resolved. However, Bernstein [40] was able to show that one can achieve a tight reduction between single-user security of Schnorr signatures and multi-user security of *key-prefixed* Schnorr signatures. Key prefixing had been introduced earlier by Menezes and Smart [170] in the context of key-substitution attacks, where they also (controversially) claimed that it is not necessary to actively mitigate key-substitution attacks for Schnorr signatures. In contrast, the designs of Ed25519 signatures [43, 121] nevertheless employ key-prefixing. Kiltz et al. [125] show that if the underlying canonical identification scheme achieves random self-reducibility in the random oracle model, then a tight reduction between multi-user and single-user security can be achieved without key prefixing. In Section 3.5.3, we briefly discuss multi-user security in light of these results.

3.2.3 Computational Proofs of Systems that Use Ed25519

Because of its performance and conjectured security, EdDSA's Ed25519 instantiation over Edwards25519 has become one of the most popular digital signature schemes, appearing in innumerable applications and protocols including TLS 1.3 [161], SSH [143], Tor, ZCash, and the Signal protocol [151].

Regarding such systems, there exists numerous security proofs which hold only when the deployed digital signatures satisfy certain conditions. For example, Bhargavan et al. [47] developed the first machine-checked cryptographic proof for TLS 1.3 draft-18 using the verification tool CryptoVerif, thereby assuming that Ed25519-IETF meets EUF-CMA. Similarly, [84] proved the security of session resumption in the TLS 1.3 draft-05 full handshakes and [131] proved the security of (a slightly modified version of) the ephemeral Diffie-Hellman handshake of TLS 1.3 with unilateral authentication. Kobeissi, Bhargavan, and Blanchet [130] analyzed a model of the Signal protocol in CryptoVerif assuming EUF-CMA security of Signal's X-Ed25519 scheme. However, none of these schemes have actually been proven to achieve EUF-CMA security.

```
$\mathsf{FS.KGen}()$:
    $(pk, sk) \xleftarrow{\$} \mathsf{CID.KGen}()$
    return $(pk, sk)$

$\mathsf{FS.Sign}(sk, m)$:
    $(\mathsf{com}, \mathsf{st}) \xleftarrow{\$} \mathsf{CID.P}_1(sk)$
    $\mathsf{ch} \xleftarrow{\$} \mathsf{H}(\mathsf{com}, m)$, $\mathsf{rsp} \xleftarrow{\$} \mathsf{CID.P}_2(\mathsf{ch}, \mathsf{st})$, $\sigma \leftarrow (\mathsf{com}, \mathsf{ch}, \mathsf{rsp})$
    return $\sigma$

$\mathsf{FS.Vrfy}(pk, m, \sigma)$:
    $(\mathsf{com}, \mathsf{ch}, \mathsf{rsp}) \leftarrow \sigma$
    return $[\![\mathsf{ch} = \mathsf{H}(\mathsf{com}, m) \wedge \mathsf{CID.V}_2(pk, \mathsf{com}, \mathsf{ch}, \mathsf{rsp})]\!]$
```

Figure 3.2: Signature scheme $\mathsf{FS}[\mathsf{CID}, \mathsf{H}] = (\mathsf{FS.KGen}, \mathsf{FS.Sign}, \mathsf{FS.Vrfy})$ resulting from the (transcript variant of) Fiat-Shamir applied to the canonical identification protocol $\mathsf{CID} = (\mathsf{CID.KGen}, \mathsf{CID.P}, \mathsf{CID.V})$.

In 2014, SSH was proven to be secure even when the same signing key is used across multiple ciphersuites, assuming that the underlying signature is strongly unforgeable [37][1]. However, SSH implementations may use the originally-proposed version Ed25519-Original (e.g., [149]), which does *not* satisfy SUF-CMA. This yields a counterexample in their security model: mauling a signature in an otherwise honest session allows a session-key reveal on the peer, as the sessions no longer match. Thus, their proof does not apply as-is to SSH implementations that use Ed25519-Original. [100] claims that the results on Schnorr signatures in prime order groups imply that Ed25519 enjoys SUF-CMA and resistance to key substitution attacks, which, as we will see in Section 3.5.4 is not the case.

3.2.4 The Fiat-Shamir Transform

Finally, we review the Fiat-Shamir transform [90] which allows to transform passively-secure (interactive) identification protocols into (non-interactive) signature schemes which are secure against active attackers.

We follow the approach by Abdalla et al. [1] when applying the Fiat-Shamir transform, i.e., we start from a canonical identification protocol that is secure against impersonation under passive attack and model the hash function as a random oracle (cf. Section 2.3) to show the existential unforgeability of the resulting signature scheme.

Let $\mathsf{CID} = (\mathsf{CID.KGen}, \mathsf{CID.P}, \mathsf{CID.V})$ be an IMP-PA-secure canonical identification protocol and let $\mathsf{H} : \{0,1\}^\star \to \{0,1\}^n$ be a cryptographic hash function with output length n modeled as a random oracle. Then the signature scheme $\mathsf{FS}[\mathsf{CID}, \mathsf{H}] = (\mathsf{FS.KGen}, \mathsf{FS.Sign}, \mathsf{FS.Vrfy})$ constructed as described in Figure 3.2 is existentially unforgeable under chosen message attacks.

Note that there are different variants of the Fiat-Shamir transform in terms of how the signatures are constructed. The transform shown in Figure 3.2 is of the *transcript variant* as used, e.g., by Pointcheval and Stern [156], where the signature consists of the entire conversation of the identification scheme.

[1]The full version of the paper [38] explicitly uses the definition for strong unforgeability, even though both versions use a "euf-cma" shorthand.

```
$\mathsf{CID.KGen}(\mathsf{pp})$:
    $a \xleftarrow{\$} \mathbb{F}_q$
    $A \leftarrow g^a$
    $\textbf{return } (A, a)$

$\mathsf{CID.P}_1(a)$:
    $r \xleftarrow{\$} \mathbb{F}_q$
    $R \leftarrow g^r$
    $\textbf{return } (R, a)$

$\mathsf{CID.V}_1$:
    $\mathsf{ch} \xleftarrow{\$} \mathbb{F}_q$
    $\textbf{return } \mathsf{ch}$

$\mathsf{CID.P}_2(\mathsf{ch}, a)$:
    $S \leftarrow (R + \mathsf{ch} \cdot a) \mod q$
    $\textbf{return } S$

$\mathsf{CID.V}_2(A, R, \mathsf{ch}, S)$:
    $R' \leftarrow g^S \cdot A^{-\mathsf{ch}}$
    $\textbf{return } [\![R' = R]\!]$
```

```
$\mathsf{DS.KGen}(\mathsf{pp})$:
    $a \xleftarrow{\$} \mathbb{F}_q$
    $A \leftarrow g^a$
    $\textbf{return } (A, a)$

$\mathsf{DS.Vrfy}(A, m, \sigma = (\mathsf{ch}, S))$:
    $R' \leftarrow g^S \cdot A^{-\mathsf{ch}}$
    $\mathsf{ch}' \leftarrow \mathsf{H}(R', m)$
    $\textbf{return } [\![\mathsf{ch}' = \mathsf{ch}]\!]$

$\mathsf{DS.Sign}(a, m)$:
    $r \xleftarrow{\$} \mathbb{F}_q$
    $R \leftarrow g^r$
    $\mathsf{ch} \leftarrow \mathsf{H}(R, m)$
    $S \leftarrow (R + \mathsf{ch} \cdot a) \mod q$
    $\textbf{return } \sigma \leftarrow (\mathsf{ch}, S)$
```

Figure 3.3: Schnorr signature scheme $\mathsf{DS} = \mathsf{FS}[\mathsf{CID}, \mathsf{H}] = (\mathsf{DS.KGen}, \mathsf{DS.Sign}, \mathsf{DS.Vrfy})$ (right) resulting from the Fiat-Shamir transform applied to the Schnorr identification protocol $\mathsf{CID} = (\mathsf{CID.KGen}, \mathsf{CID.P}, \mathsf{CID.V})$ (left).

Example 1 (Schnorr Signatures)**.** *Schnorr signatures are a prime example of the way secure signatures can be constructed via the Fiat-Shamir transform from secure identification protocols. [166]. They achieve short, efficient signatures that are provably secure in the random oracle model, assuming the hardness of the discrete logarithm problem in the underlying group. The underlying Schnorr identification scheme, as well as the resulting signature scheme when the Fiat-Shamir transform is applied, are depicted in Figure 3.3. Here,* $\mathsf{pp} = (\mathbb{G}, q, g)$ *denote the public parameters of the scheme where* $\mathbb{G}$ *is a cyclic group of prime order* q *with generator* g*.*

Schnorr signatures as described in Example 1 are of the *challenge variant* where the signature consists only of the challenge and the response, i.e., $\sigma \leftarrow (\mathsf{ch}, \mathsf{rsp})$. This requires that there exists an algorithm that can reconstruct the commitment com from the public key, the challenge, and the response. Further signatures that are of the challenge variant are the original work by Fiat and Shamir, GQ signatures [99] and Okamoto signatures [148]. An in-depth treatment of these variants, including a third variant, the *commitment variant*, can be found in Backendal et al. [16]. As we will later see, Ed25519 signatures are a deterministic variant of Schnorr signatures but in the commitment-variant of the Fiat-Shamir transform. In order to make Fiat-Shamir signatures as described in Figure 3.2 deterministic, $\mathsf{CID.P}_1$ is *derandomized* by using $\mathsf{H}(sk, m)$ as randomness during commitment generation.

3.3 Additional Preliminaries

Notations: For an integer q, we denote by $\mathbb{F}_q$ the finite field with order q. For a bit string h and an integer i, we let $h[i]$ denote the i-th bit of h. Overloading notation, we write $\underline{a}$ for the bitstring encoding of a, where a can be an integer or a curve point. We later describe the details of these encodings at the start of Section 3.4.

3.3.1 Elliptic Curves

We briefly recap the main theory of elliptic curves. For a more in-depth treatment of specific concepts and constructions, we refer the interested reader to, e.g., [71, 102]. We begin by defining elliptic curves over a finite field $\mathbb{F}_q$, which are the most common types of elliptic curves in cryptography:

Definition 31 (Elliptic curve)**.** *Let $q \geq 5$ be a prime. An* elliptic curve EC *defined over the finite field $\mathbb{F}_q$ is an equation of the form*

$$y^2 = x^3 + ax + b \tag{3.1}$$

with $a, b \in \mathbb{F}_q$ such that $4a^3 + 27b^2 \neq 0$.

In elliptic-curve cryptography, the group in question is the set of points on the elliptic curve EC.

Definition 32 (Points on EC)**.** *Let $\mathsf{EC}(\mathbb{F}_q)$ be the set of pairs $(x, y) \in \mathbb{F}_q \times \mathbb{F}_q$ satisfying the elliptic curve equation. Let $\mathcal{O}$ denote a special point, the so-called* point at infinity*. Then the set*

$$\mathsf{EC}(\mathbb{F}_q) := \{(x, y) | x, y \in \mathbb{F}_q \wedge y^2 = x^3 + ax + b\} \cup \{\mathcal{O}\}$$

denotes the points on the elliptic curve EC*.*

With an adequately defined addition operation "+" $\mathsf{EC}(\mathbb{F}_q)$ forms a group with neutral element $(0, 1)$. The multiplication of a curve point P with an integer n is defined as adding P n times to itself, i.e.,

$$nP := \underbrace{P + P + \cdots + P}_{n \text{ times}},$$

where $0P := \mathcal{O}$.
For brevity, we often write EC instead of $\mathsf{EC}(\mathbb{F}_q)$ if the underlying field is clear from context.

Further Definitions The number of points on an elliptic curve EC over $\mathbb{F}_q$ is called the *order* of the curve and is denoted by $|\mathsf{EC}(\mathbb{F}_q)|$. We call an element B that generates a cyclic subgroup the *base point* and write $P \in \langle B \rangle$ to indicate that P is an element of the subgroup generated by B. For an element B, we overload notation and write $|B|$ to denote its order, i.e., the smallest integer n such that $nB = \mathcal{O}$. If B generates a subgroup of $\mathsf{EC}(\mathbb{F}_q)$, we define the *cofactor* to be the integer $\frac{|\mathsf{EC}(\mathbb{F}_q)|}{|B|}$.

3.3.1.1 Twisted Edwards Curves and Discrete-Log Problem

There exist several different forms of elliptic curve equations, such as *Weierstraß*, *Montgomery* or *Edwards* form. Most relevant for this paper are *twisted Edwards curves* [41] which are defined over a finite field $\mathbb{F}_q$ with $q > 3$ prime with additional parameter a (the *twist*) via the curve equation

$$\mathsf{EC}_{TEd} : ax^2 + y^2 = 1 + dx^2y^2,$$

where $a, d \in \mathbb{F}_q$ with $a, d \neq 0$ and $a \neq d$.

Addition "+" on $E_{TEd}(\mathbb{F}_q)$ is defined as follows. Let $P = (x_1, y_1), Q = (x_2, y_2) \in \mathsf{EC}_{TEd}(\mathbb{F}_q)$, then $P + Q = (x_3, y_3)$ is defined as

$$x_3 = \frac{x_1y_2 + x_2y_1}{1 + dx_1x_2y_1y_2}, \qquad y_3 = \frac{y_1y_2 - ax_1x_2}{1 - dx_1x_2y_1y_2}.$$

Note that if a is a square in $\mathbb{F}_q$ and d is non-square in $\mathbb{F}_q$, then the addition operation is complete and $(E_{TEd}(\mathbb{F}_q), +)$ is a group with neutral element $(0, 1)$. The inverse $-P$ of a point $P = (x, y) \in E_{TEd}(\mathbb{F}_q)$ is $(-x, y)$.

The twisted Edwards curve EC underlying the Ed25519 constructions, which we discuss in more detail in the next section, is birationally equivalent to `curve25519` introduced by Bernstein [39], which is of the Montgomery form and due to its efficient arithmetic implementation yields very performant constructions. `curve25519` is defined over the finite field $\mathbb{F}_q$ with $q = 2^{255} - 19$ prime via the curve equation

$$\texttt{curve25519} : y^2 = x^3 + 486662x^2 + x.$$

Definition 33 (ECDLP). *Let* EC *be an elliptic curve defined over a finite field* $\mathbb{F}_q$ *and let* $B \in \mathsf{EC}(\mathbb{F}_q)$ *be a point of order* n. *Let* $P \in \langle B \rangle$. *Then the* elliptic curve discrete-log problem *is to find an integer* $0 \leq k < n$ *such that* $P = kB$. *We say that* EC *is* (t, ϵ)-hard on EC *if for any algorithm* $\mathcal{A}$ *running in time at most* t *the probability of solving* ECDLP *is at most* ϵ.

3.3.2 Secure Canonical Identification Protocols

Canonical Identification (CID) protocols allow a so-called *prover* CID.P that holds a secret key sk to authenticate to a *verifier* CID.V who holds the corresponding public key vk. CID protocols consist of three moves: The prover CID.P sends a commitment com to the verifier CID.V. The verifier CID.V then samples a random challenge ch and sends it to CID.P. Finally, CID.P sends a response rsp to CID.V, whose decision is then a deterministic function of their *conversation* (com, ch, rsp) and CID.P's public key. More formally:

Definition 34 (Canonical identification protocol)**.** *A canonical identification (CID) protocol* $\mathsf{CID} = (\mathsf{CID.KGen}, \mathsf{CID.P} = (\mathsf{CID.P}_1, \mathsf{CID.P}_2), \mathsf{CID.V} = (\mathsf{CID.V}_1, \mathsf{CID.V}_2))$ *is a triple of algorithms:*

Key Generation $\mathsf{CID.KGen}$ *takes as input the public parameters* pp *and outputs a public key* pk *and a secret key* sk*, i.e.,* $(pk, sk) \xleftarrow{\$} \mathsf{CID.KGen}(\mathsf{pp})$*.*

The ***prover*** $\mathsf{CID.P} = (\mathsf{CID.P}_1, \mathsf{CID.P}_2)$ *is a two-stage algorithm that takes as input a secret key* sk*.* $\mathsf{CID.P}_1$ *takes as input* sk *and outputs a commitment* com *as well as some state* st*.* $\mathsf{CID.P}_2$ *takes as input the challenge* ch *(sent by the verifier* $\mathsf{CID.V}_1$*) as well as state* st *and outputs a response* rsp*.*

The ***verifier*** $\mathsf{CID.V} = (\mathsf{CID.V}_1, \mathsf{CID.V}_2)$ *is a two-stage algorithm that is initialized with a public key* pk*.* $\mathsf{CID.V}_1$ *selects a random challenge* ch *and sends it to the prover.* $\mathsf{CID.V}_2$ *takes as input the public key, the* com*,* ch *and* rsp *and outputs* 1 *if it* accepts *the* conversation $(\mathsf{com}, \mathsf{ch}, \mathsf{rsp})$ *for* pk *or* 0 *if it* rejects*.*

For all $(pk, sk) \xleftarrow{\$} \mathsf{CID.KGen}(\mathsf{pp})$*, we require that if* $\mathsf{CID.P}(sk)$ *and* $\mathsf{CID.V}(pk)$ *interact honestly within an instance of the protocol, then the verifier accepts. I.e., for* $(\mathsf{com}, \mathsf{st}) \xleftarrow{\$} \mathsf{CID.P}_1(sk)$*,* $\mathsf{ch} \xleftarrow{\$} \mathsf{CID.V}_1$*, and* $\mathsf{rsp} \xleftarrow{\$} \mathsf{CID.P}_2(\mathsf{com}, \mathsf{ch}, \mathsf{st})$*, we have verifier* $\Pr[1 \leftarrow \mathsf{CID.V}_2(pk, \mathsf{com}, \mathsf{ch}, \mathsf{rsp})] = 1$*.*

We sometimes denote the interactive run of the identification protocol CID between the prover and the verifier by $\mathsf{CID.P} \leftrightarrows \mathsf{CID.V}$ or $\mathsf{CID.P}(sk) \leftrightarrows \mathsf{CID.V}(pk)$. We write $\mathsf{Trans}\left[\mathsf{CID.P}(sk) \leftrightarrows \mathsf{CID.V}(pk)\right]$ to denote a conversation $(\mathsf{com}, \mathsf{ch}, \mathsf{rsp})$ resulting from the interaction between $\mathsf{CID.P}$ and $\mathsf{CID.V}$ and identify $\mathsf{CID.V}_2(pk, \mathsf{com}, \mathsf{ch}, \mathsf{rsp})$ with the final decision $\in \{0, 1\}$ of the verifier. If $1 \leftarrow \mathsf{CID.V}_2(pk, \mathsf{com}, \mathsf{ch}, \mathsf{rsp})$, we say that $(\mathsf{com}, \mathsf{ch}, \mathsf{rsp})$ is an *accepting conversation for pk*, or simply a *valid conversation.*

Intuitively, the basic security of identification protocols is defined in terms of the inability of an attacker $\mathcal{A}$ to impersonate the prover towards an honest verifier without knowledge of the prover's secret key. This can be in the setting where $\mathcal{A}$ only has access to the public key of the prover (called $\mathsf{IMP\text{-}KOA}$ for security against impersonation under key-only attacks), or in the stronger setting, where $\mathcal{A}$ can observe honest conversations between the prover and the verifier ($\mathsf{IMP\text{-}PA}$ for security against impersonation under passive attacks):

Definition 35 ($\mathsf{IMP\text{-}KOA}$ and $\mathsf{IMP\text{-}PA}$ security)**.** *Let* $\mathsf{CID} = (\mathsf{CID.KGen}, \mathsf{CID.P}, \mathsf{CID.V})$ *be a canonical identification protocol. Consider the security experiment* $\mathrm{Expr}_{\mathsf{CID}}^{\mathsf{IMP\text{-}KOA}}$ *as defined on the left in Figure 3.4. We say that* CID *is* (t, ϵ)-secure against impersonation under key-only attacks*, or simply* (t, ϵ)-$\mathsf{IMP\text{-}KOA}$*-secure, if for any* PPT *attacker* $\mathcal{A}$ *running in time at most* t *the probability*

$$\Pr\left[\mathrm{Expr}_{\mathsf{CID}}^{\mathsf{IMP\text{-}KOA}}(\mathcal{A}) = 1\right] \leq \epsilon.$$

```
$\underline{\mathrm{Expr}_{\mathsf{CID}}^{\mathsf{IMP\text{-}KOA}}(\mathcal{A})}$:
$(pk, sk) \xleftarrow{\$} \mathsf{CID.KGen}()$
$\mathsf{ch} \xleftarrow{\$} \mathsf{CID.V}_1$
$(\mathsf{com}, \mathsf{st}) \xleftarrow{\$} \mathcal{A}(pk)$
$\mathsf{rsp} \xleftarrow{\$} \mathcal{A}(\mathsf{ch}, \mathsf{st})$
$\textbf{return } [\![\mathsf{CID.V}_2(pk, \mathsf{com}, \mathsf{ch}, \mathsf{rsp})]\!]$
```

```
$\underline{\mathrm{Expr}_{\mathsf{CID}}^{\mathsf{IMP\text{-}PA}}(\mathcal{A})}$:
$(pk, sk) \xleftarrow{\$} \mathsf{CID.KGen}()$
$\mathsf{ch} \xleftarrow{\$} \mathsf{CID.V}_1$
$(\mathsf{com}, \mathsf{st}) \xleftarrow{\$} \mathcal{A}(pk)$
$\mathsf{rsp} \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Trans}}}(\mathsf{ch}, \mathsf{st})$
$\textbf{return } [\![\mathsf{CID.V}_2(pk, \mathsf{com}, \mathsf{ch}, \mathsf{rsp})]\!]$

$\underline{\mathcal{O}_{\mathsf{Trans}}}$:
$\textbf{return } \mathsf{Trans}\,[\mathsf{CID.P}(sk) \leftrightarrows \mathsf{CID.V}(pk)]$
```

Figure 3.4: $\mathsf{IMP\text{-}KOA}$ and $\mathsf{IMP\text{-}PA}$ experiments for a $\mathsf{CID} = (\mathsf{CID.KGen}, \mathsf{CID.P}, \mathsf{CID.V})$ scheme against impersonating attackers $\mathcal{A}$ with differences highlighted in gray.

Similarly, consider the experiment $\mathrm{Expr}_{\mathcal{A}}^{\mathsf{IMP\text{-}PA}}$ *as defined on the right in Figure 3.4. We say that* CID *is* (t, ϵ, Q_T)-secure against impersonation under passive attacks*, or simply* (t, ϵ, Q_T)-$\mathsf{IMP\text{-}PA}$*-secure, if for any PPT attacker* $\mathcal{A}$ *running in time at most* t *and with at most* Q_T *queries to the oracle* $\mathcal{O}_{\mathsf{Trans}}$*, the probability*

$$\Pr\left[\mathrm{Expr}_{\mathsf{CID}}^{\mathsf{IMP\text{-}PA}}(\mathcal{A}) = 1\right] \leq \epsilon.$$

To argue about the security of canonical identification protocols CID, it is useful to talk about the min-entropy of an identification scheme as well as the notion of *honest-verifier zero-knowledge*, or HVZK, for short. The former notion captures the unpredictability of commitments in the protocol, whereas HVZK formalizes the property that an attacker $\mathcal{A}$ gains no additional knowledge from honest interactions $\mathsf{CID.P} \leftrightarrows \mathsf{CID.V}$, since $\mathcal{A}$ could generate such conversations on its own.

This is done by showing that there exists an algorithm Sim, that only takes as input the public key and can output conversations that are indistinguishable from *real* interactions $\mathsf{CID.P}(sk) \leftrightarrows \mathsf{CID.V}(pk)$.

Definition 36 (Min-entropy of identification scheme)**.** *We say that a canonical identification protocol* $\mathsf{CID} = (\mathsf{CID.KGen}, \mathsf{CID.P}, \mathsf{CID.V})$ *has* α bits min-entropy *if the probability over the choice* $(pk, sk) \xleftarrow{\$} \mathsf{CID.KGen(pp)}$*, that the commitment generated by* $\mathsf{CID.P}_1(sk)$ *is from a distribution with at least* α *bits of min-entropy, is at least* $1 - 2^{\alpha}$*. Recall that a discrete random variable* X *has* α *bits of min-entropy, denoted by* $H_\infty(X) := \alpha$*, if* $\max_x (\Pr[X = x]) = 2^{-\alpha}$.

Definition 37 (Honest-verifier zero-knowledge)**.** *Let* $\mathsf{CID} = (\mathsf{CID.KGen}, \mathsf{CID.P}, \mathsf{CID.V})$ *be a canonical identification protocol. We say that* CID *is* $\epsilon_{\mathsf{CID}}^{\mathsf{hvzk}}$-honest-verifier zero-knowledge*, or* $\epsilon_{\mathsf{CID}}^{\mathsf{hvzk}}$-$\mathsf{HVZK}$ *for short, if there exists a* PPT *algorithm* Sim*, called the* simulator*, such that for all* $(pk, sk) \xleftarrow{\$} \mathsf{CID.KGen(pp)}$*, the outputs of* $\mathsf{Sim}(pk)$ *can only be distinguished from real conversations* $\mathsf{CID.P}(sk) \leftrightarrows \mathsf{CID.V}(pk)$ *with probability at most* $\epsilon_{\mathsf{CID}}^{\mathsf{hvzk}}$.

We recall two important properties of CID protocols. The first one is *commitment recoverable*, which means that commitments can be efficiently and publicly computed from the public key, the challenge, and the response. The second one is *(computationally) unique responses*, meaning that for a fixed instance of the protocol and commitments and challenges, there exists at most one response such that the verifier will accept the conversation (or a second response is computationally infeasible to find). More formally:

Definition 38 (Commitment-recoverability CR)**.** *Let* $\mathsf{CID} = (\mathsf{CID.KGen}, \mathsf{CID.P}, \mathsf{CID.V})$ *be a canonical identification protocol. We say that* CID *is* commitment-recoverable*, or* CR*, for short, if for any* $(pk, sk) \xleftarrow{\$} \mathsf{CID.KGen}(\mathsf{pp})$*,* $\mathsf{ch} \xleftarrow{\$} \mathsf{CID.V}_1$*,* $\mathsf{rsp} \xleftarrow{\$} \mathsf{CID.P}_2(\mathsf{ch}, \mathsf{st})$*, there exists a unique* com *such that* $\mathsf{CID.V}_2(pk, \mathsf{com}, \mathsf{ch}, \mathsf{rsp}) = 1$ *and this* com *can be efficiently computed given only* $(pk, \mathsf{ch}, \mathsf{rsp})$*.*

Definition 39 ((Computationally) unique responses CUR)**.** *Let* $\mathsf{CID} = (\mathsf{CID.KGen}, \mathsf{CID.P}, \mathsf{CID.V})$ *be a canonical identification protocol. We say that* CID *has* $\epsilon^{\mathsf{cur}}_{\mathsf{CID}}$-computationally unique responses *or* CUR*, if for any* $(pk, sk) \xleftarrow{\$} \mathsf{CID.KGen}(\mathsf{pp})$*,* $\mathsf{com} \xleftarrow{\$} \mathsf{CID.P}_1(sk)$ *and* $\mathsf{ch} \xleftarrow{\$} \mathsf{CID.V}_1$ *the probability of an attacker being able to output two responses* rsp *and* rsp' *such that* $\mathsf{CID.V}_2(pk, \mathsf{com}, \mathsf{ch}, \mathsf{rsp}) = 1$ *and* $\mathsf{CID.V}_2(pk, \mathsf{com}, \mathsf{ch}, \mathsf{rsp}') = 1$ *is at most* $\epsilon^{\mathsf{cur}}_{\mathsf{CID}}$*. If* $\epsilon^{\mathsf{cur}}_{\mathsf{CID}} = 0$ *we say that* CID *has* unique responses*.*

3.4 Ed25519 Signatures

In this section we describe how the Ed25519 [43, 121] signature scheme operates in detail, unravel its relationship with Schnorr signatures and why proofs for Schnorr are not directly applicable to Ed25519. We also describe several of the proposed variants of Ed25519, which target stronger security properties than provided by the original formulation.

We define the generic signature scheme Ed25519 in Figure 3.1. Part of the generic scheme description, highlighted on line 13 in Figure 3.1, is replaced in the variant schemes. We summarise these variations in Table 3.1 and discuss them further below. The various parameters common to all variants are listed in Table 3.2. These parameters include those necessary to define the elliptic curve EC over which the signature scheme operates and the hash function used.

Encodings Integers mod L are encoded as as b-bit strings in little endian format. Elliptic curve points (x, y) are encoded as a $(b-1)$-bit little-endian encoding of y, followed by a sign bit which is 1 if and only if x is negative. We note an oft-omitted property of the encoding scheme: the field element that represents y is encoded as a $(b - 1) = 255$-bit string, but the size of the field is $q = 2^{255} - 19$, yielding a larger space of encodings than actual elements. Similarly, the S part of the signature is expected to be a b-bit integer, but is necessarily reduced $\bmod L$ prior to use in the signature verification. These details turn out to have substantial consequences when showing security of the scheme and we will highlight this in the respective proofs. In places, the original presentation [43] and its accompanying source code disagree on the necessary tests, e.g., the range of L. As the source code is the basis of the most popular Ed25519 implementations, we treat it as authoritative.

3.4.1 Variants of Ed25519

Recall that we refer to the original, and currently most widely deployed formulation of Ed25519 as Ed25519-Original. Several alternative specifications have also been published which largely maintain wire compatibility with Ed25519-Original, but substantially alter the security properties of the scheme. In particular, variants have been published by the IETF [121], NIST [70], and the ZCash Foundation [108]. Furthermore, LibSodium [119], one of the most popular cryptographic libraries, uses a set of additional checks that render it a de-facto standard.

Another variant is the use of Ristretto encodings [162] for Ed25519. Whilst this appears promising, the draft RFC [96] is still under active development and we do not analyze it here.

We note that the Signal Foundation [151] have also proposed a variant with enhanced resistance to side-channel attacks and fault resistance during signature generation. However, this variant operates similarly to Ed25519-Original with regards to signature verification and consequently we do not treat it separately.

3.4.1.1 Pre-Hashing Variants

The IETF-standardised RFC 8032 [121] and the NIST draft standard [70] support a *pre-hashing* mode for their variants. This mode allows implementations to sign large messages whilst only needing to perform a single pass over the message. The signed message value m is replaced with the pre-hash $\mathsf{PH}(m)$, where PH is a hash function. The IETF specification explicitly recommends against the use of this mode, stressing it is included only to support legacy signing interfaces. Consequently, we do not discuss it further.

3.4.1.2 Bounds Checking Variants

Some variants of Ed25519 require an additional check on the received alleged signatures during verification. In particular, this is enforced by the IETF standard [121] and proposed in the NIST draft standard [70]. In these variants, implementers are required to reject signatures whose S parameter is equal to or larger than L, where L is the order of the prime-order subgroup. Contrastingly, Ed25519-Original implementations merely check S is a 256-bit integer. We refer to such implementations as Ed25519-IETF. This is claimed to have a substantial impact and achieve strong unforgeability. We investigate this further in Section 3.4 but already note that, contrastingly to Ed25519-IETF variants, Ed25519-Original implementations simply reduce the received signature element $S \bmod L$ during signature verification, thereby immediately ruling out SUF-CMA security.

3.4.1.3 Point and Bound Checking Variants

Some popular implementations of Ed25519, such as LibSodium [119], perform the bounds check on S values in addition to point validation. Specifically, Ed25519-LibS ensures that received elements are canonically encoded and have order $\geqslant L$. This check means that certain R values, which are low order points on the curve, are rejected during verification as well as low order public keys. Additionally, R values and public keys which have y-coordinates above q are also rejected to ensure unique encodings. We see the impact of these decisions in Section 3.5.4.

3.4.2 Differences from Schnorr Signatures

Ed25519 has several notable features which differentiate it from traditional Schnorr signatures. The original Schnorr signature scheme and its underlying identification protocol can be recalled in Example 1. In particular, private keys in Ed25519 are *clamped* to a specific format, signature nonces are chosen deterministically, and signed messages are prefixed with public keys. As indicated before, these differences impact the security of the overall signature scheme and our analysis. We next discuss these alterations in more detail.

3.4.2.1 Group Structure

Ed25519's curve is of order $8L$ for L a large prime defined in Table 3.2. Contrastingly, Schnorr signatures are typically constructed over prime order groups and implementations are assumed to reject the identity element.

Non-prime order groups contain a more complex group structure than prime order groups. In particular, non-prime order groups entail the presence of additional subgroups, whose elements lie outside the intended prime order subgroup. Performing group operations on these elements can lead to surprising results, including confinement under exponentiation, where they map to a small range of elements and leakage, where performing exponentiation with a private scalar leaks information about that scalar. The original paper [43] allowed Ed25519 implementations to optionally include multiplication by the cofactor in the verification equation. Including the cofactor makes the verification function strictly more permissive and we assume it is present so that our proofs carry over to the cofactorless case.

Proofs about systems defined over prime order groups do not necessarily hold if the system is implemented with non-prime order groups, even when the proof does not explicitly rely on the prime order structure. For example, proofs typically assume that any group element can be written as the exponentiation of a fixed generator, or that exponentiation of an element is uniformly distributed over the group. In the non-prime order case, neither assumption is true in general.

3.4.2.2 Group Element Checks

The question of how values should be decoded and parsed is often omitted from academic papers. Ed25519-Original and Ed25519-IETF are of particular interest in this regard as it is explicitly argued that elements do not need to be checked to ensure they belong to the prime order subgroup. This means any group element, including small order elements, the identity element and elements of order $8L$ will be accepted. We discussed variants that mandate this check in Section 3.4.1.3. The related Diffie-Hellman function X25519 also omits these checks which has previously led to otherwise avoidable attacks on protocols employing it [77].

A related issue is whether elements are checked to ensure they lie on the intended curve, rather than its twist. Whilst X25519 does not reject elements on the twist, Ed25519-Original explicitly mandates that points are checked to ensure they do belong to EC. This is checked during the decompression of received points prior to signature verification. We assume all implementations uphold this requirement as otherwise point addition during signature verification is not necessarily defined.

3.4.2.3 Private Key Clamping

In part due to the non-prime order nature of the curve and the lack of group element validation, Ed25519 mandates the use of *key clamping* which involves the bitwise manipulation of private keys prior to use in signing. The rationale behind this requirement has been the subject of much debate [57, 142]. The original Ed25519 paper defines private keys, without discussion, such that a high bit is always set and three low bits are cleared. All subsequent variations have kept the same requirement. There are two rationales for clamping:

- Setting the high bit ensures that some deficient point multiplication implementations, which have variable execution time with respect to the position of the highest set bit in the scalar, become constant time [110].
- Clearing the low bits ensures that the scalar is a multiple of the cofactor. This ensures that the result of applying the scalar to any group element results in an element in the prime order subgroup. This avoids key leakage attacks, although these attacks are not relevant for Ed25519 signature schemes as private keys are never applied to attacker-provided group elements. However, as implementers may wish to re-use keys in both X25519 and Ed25519, this choice provides defence in depth.

As we will see later, the use of clamping complicates our security proofs. This is because not every element in the prime order subgroup is also a valid public key as produced by the key generation algorithm. Consequently, when providing reductions which must manipulate public keys, e.g., blinding them, there is a small chance an invalid public key is produced and the reduction must abort.

3.4.2.4 Key Prefixing

Unlike traditional Schnorr signatures, Ed25519 uses key prefixing, where the signature scheme, prior to signing or verification, prepends the public key to the message. This choice has also been the subject of much debate [40, 111, 112, 125] as to whether it provides a substantial improvement in security. Much of the discussion has revolved around multi-user security and whether key prefixing improves security in the presence of an attacker who is satisfied to break some subset of witnessed multiple keys, rather than one key in particular. We discuss the multi-user security of Ed25519 in more detail in Section 3.5.3. It transpires that key prefixing also has benefits when considering lesser-known multi-user security properties such as message key substitution attacks, as we show in Section 3.5.4.

3.4.2.5 Deterministic Nonce Generation

Signature schemes require the use of a nonce with each signed message. This has historically been an area prone to subtle implementation mistakes leading to critical real world vulnerabilities [14]. Ed25519 uses *deterministic signing* which removes the need for fresh random numbers during the signing process. This does not lead to any particular consequences for our security analysis since we model the key derivation function as a random oracle. However, it is well known not to reduce security [33].

3.5 The Security of Ed25519

We now present our security results for Ed25519-Original and Ed25519-IETF signatures. As suggested by earlier works that informally discuss the security of these schemes, we use the Fiat-Shamir approach. However, as elaborated in Section 3.4.2, there exist marked differences between Schnorr signatures and Ed25519 signatures such that the established security results for Schnorr do not hold without careful adjustment to the Ed25519 setting. We close this gap by proving both the existential unforgeability of Ed25519-Original and show that due to the additional check on the value S, Ed25519-IETF and Ed25519-LibS achieve strong unforgeability. As is common for signature proofs, we assume idealized versions of hash functions (random oracles), but do not make any strong assumptions on the properties of the underlying elliptic curve group. Tighter security bounds may be possible in the so-called *generic group model* (cf., e.g., [24, 125]), however we explicitly

```
CID.KGen(pp):
1  $k \xleftarrow{\$} \{0,1\}^b$
2  $h \leftarrow \mathsf{H}(k)$
3  $s \leftarrow 2^{b-2} + \sum_{i=3}^{b-3} 2^i h[i]$
4  $A \leftarrow sB$
5  return $(\underline{A}, k)$

CID.P_1($k$):
6  $h \leftarrow \mathsf{H}(k)$
7  $s \leftarrow 2^{b-2} + \sum_{i=3}^{b-3} 2^i h[i]$
8  $\mathsf{st} \leftarrow s$
9  $r \xleftarrow{\$} \{0,1\}^{2b}$
10 $R \leftarrow rB$
11 return $(\underline{R}, \mathsf{st})$

CID.V_1:
12 $\mathsf{ch} \xleftarrow{\$} \{0,1\}^{2b}$
13 return $\mathsf{ch}$

CID.P_2(ch, st):
14 $S \leftarrow (r + \mathsf{ch} \cdot \mathsf{st}) \mod L$
15 return $\underline{S}$

CID.V_2($\underline{A}, \underline{R}, \underline{S}, \mathsf{ch}$):
16 Check $R, A \in E$
17 Variant Specific Checks
18 return $[\![8SB = 8R + 8\mathsf{ch}A]\!]$
```

Figure 3.5: Canonical identification protocol $\mathsf{CID} = (\mathsf{CID.KGen}, \mathsf{CID.P} = (\mathsf{CID.P}_1, \mathsf{CID.P}_2), \mathsf{CID.V} = (\mathsf{CID.V}_1, \mathsf{CID.V}_2))$ underlying Ed25519-Original in Figure 3.1. Note that the highlighted Line 17 varies depending on the version of Ed25519 and the appropriate check is listed in Table 3.1.

want to explore security in a setting where the attacker may take advantage of, e.g., encoding details.

We recall that, in the following, EC refers to the twisted Edwards curve underlying Ed25519 (cf. Table 3.2) and analogously $\mathsf{EC}(\mathbb{F}_q)$ denotes the set of elements on the curve which forms a group with point addition, as defined in Section 3.3.1.

3.5.1 Existential Unforgeability of Original Ed25519

We start by showing EUF-CMA security of Ed25519-Original specifically by means of the Fiat-Shamir transform. We first define an appropriate canonical identification protocol CID in Figure 3.5 to which the transform can be applied, then show that CID satisfies the necessary prerequisites in Theorem 2 and Theorem 3 and finally apply the transform in Theorem 4 to establish existential unforgeability of the resulting signature scheme. We note that with the additional check in Line 17 of CID in Figure 3.5, the proof of EUF-CMA security directly carries over to Ed25519-IETF and with the further check described in Table 3.1 also to Ed25519-LibS.

To get from CID in Figure 3.5 to the Ed25519-Original in Figure 3.1, we apply a variant of Fiat-Shamir, denoted by $\mathsf{FS}_{\mathsf{det}}^{\mathsf{kp}}$, that captures *deterministic signing* and *key-prefixing.*

Deterministic signing is achieved by deterministically deriving the randomness r of $\mathsf{CID.P}_1$ in the signing algorithm via $r \leftarrow \mathsf{H}(h[b], ..., h[2b-1], m)$, where m is the message to be signed, and, as before, $h[i]$ denotes the i-th bit of h. The de-randomization of signing by computing the randomness deterministically as some $\mathsf{H}(sk, m)$ is common and in our case does not impact the security of the resulting signatures, assuming H is at least a pseudorandom function (cf., e.g., [33]).

Recall that key-prefixing, on the other hand, describes the derivation of the challenge ch by the prover during signature generation as $\mathsf{H}(\underline{R}, \underline{A}, m)$ instead of $\mathsf{H}(\underline{R}, m)$, i.e., by also including the public key into the hash function. It will become clear from the proof that this additional input to the random oracle H does not impact security, since this solely relies on $\underline{R}$ being sufficiently unpredictable in honest signature generations.

3.5.1.1 Security of the underlying identification protocol

We show that the underlying CID is secure against impersonation attacks by passive attackers ($\mathsf{IMP\text{-}PA}$ security, cf. Definition 35). We do this in a two-step process by first showing in Theorem 2 that CID is secure against impersonating attackers that only have access to the public key ($\mathsf{IMP\text{-}KOA}$ security). To lift this result to the setting of passive impersonating attackers, we then only need to be able to simulate queries to the oracle $\mathcal{O}_{\mathsf{Trans}}$, which can be achieved by the HVZK property of CID. Thus, in Lemma 1 we show that CID is HVZK and, combined with the $\mathsf{IMP\text{-}KOA}$ security, we achieve $\mathsf{IMP\text{-}PA}$ security.

Theorem 2 (CID is $\mathsf{IMP\text{-}KOA}$ secure)**.** *Let* $\mathsf{CID} = (\mathsf{CID.KGen}, \mathsf{CID.P}, \mathsf{CID.V})$ *be the identification protocol as defined in Figure 3.5. If* ECDLP *is* $(t, \epsilon_{\mathsf{EC}}^{\mathsf{DLP}})$*-hard on* $\mathsf{EC}(\mathbb{F}_q)$*, then* CID *is* (t', ϵ')*-*$\mathsf{IMP\text{-}KOA}$ *secure, where*

$$t \approx 2t' \quad \textit{and} \quad \frac{1}{8}\left(\epsilon' - \frac{1}{L}\right)^2 \leq \epsilon_{\mathsf{EC}}^{\mathsf{DLP}}.$$

Before we prove the theorem below, we want to recall that secret keys in the key generation of CID are *clamped*, which in particular means that an element $A \in \mathsf{EC}(\mathbb{Z}_q)$ is not necessarily a valid public key output of $\mathsf{DS.KGen}$. The reduction in the proof accounts for this and therefore loses a factor of 8. Furthermore, note again that the hardness of ECDLP on $\mathsf{EC}(\mathbb{Z}_q)$ as used in $\mathsf{Ed25519}$ carries over from the respective hardness on `curve25519` due to their birational equivalence [44].

Proof Sketch. The full proof can be found in the Section 3.7.1. We notice the first marked difference to proofs of Schnorr signatures. While in the latter, we have a straightforward reduction to ECDLP using the *rewinding technique* [157], we now need to account for the secret key clamping in $\mathsf{Ed25519}$. The clamping causes that an element $A \in \mathsf{EC}(\mathbb{F}_q)$ is not necessarily a valid public key output of $\mathsf{DS.KGen}$ and thus cannot be relayed by the reduction to the $\mathsf{IMP\text{-}KOA}$ attacker $\mathcal{A}$, resulting in the loss of a factor of $\frac{2^{b-5}}{L}$. As in the Schnorr setting, the reduction exploits the property that from two valid conversations $(\underline{R}^\star, \mathsf{ch}_1, \underline{S}_1)$ and $(\underline{R}^\star, \mathsf{ch}_2, \underline{S}_2)$ for the public key $\underline{A}$ with $\mathsf{ch}_1 \neq \mathsf{ch}_2 \pmod L$, we can extract the value $s = \frac{S_1 - S_2}{\mathsf{ch}_1 - \mathsf{ch}_2} \mod L$ such that $A = sB$. The *Reset Lemma* [32] which is the analogue of the *Forking Lemma* [157] for identification protocols instead of signatures,

```
Sim($\underline{A}$):
$\mathsf{ch} \xleftarrow{\$} \{0,1\}^{2b}$
$\tilde{s} \xleftarrow{\$} \{0,1\}^{2b}$
$S \leftarrow \tilde{s} \mod L$
$R \leftarrow (SB - \mathsf{ch}A) \mod L$
return $(\underline{R}, \mathsf{ch}, \underline{S})$
```

Figure 3.6: Simulator Sim for CID = (CID.KGen, CID.P = (CID.P$_1$, CID.P$_2$), CID.V = (CID.V$_1$, CID.V$_2$)) underlying Ed25519-Original.

ensures that two such conversations can be found with probability at least $(\epsilon' - \frac{1}{L})^2$, where ϵ' is the success probability of $\mathcal{A}$. We note the hardness of ECDLP on EC($\mathbb{F}_q$) as used in Ed25519 carries over from the respective hardness on `curve25519` due to their birational equivalence [44, 121]. □

As mentioned before, we now lift this result to show that an attacker cannot impersonate the prover, even when given access to an oracle $\mathcal{O}_{\mathsf{Trans}}$, that outputs valid accepting conversations Trans [CID.P $\rightleftarrows$ CID.V]. This is due to the fact, that the underlying canonical identification protocol CID is $\epsilon^{\mathsf{hvzk}}_{\mathsf{CID}}$-HVZK, i.e., there exists a simulator Sim which takes as input only the public key vk outputs conversations (com, ch, rsp) which are $\epsilon^{\mathsf{hvzk}}_{\mathsf{CID}}$-indistinguishable from real conversations CID.P(sk) $\leftrightarrows$ CID.V(pk). The simulator Sim(vk) for CID is defined in Figure 3.6 and exploits the fact that commitments can be recovered from the public key, the challenge, and the response.

Lemma 1. *Let* CID = (CID.KGen, CID.P, CID.V) *be as defined in Figure 3.5. Then* CID *is* HVZK *with* $\epsilon^{\mathsf{hvzk}}_{\mathsf{CID}} = 0$.

Theorem 3 (CID is IMP-PA secure)**.** *Let* CID = (CID.KGen, CID.P, CID.V) *be the* $\epsilon^{\mathsf{hvzk}}_{\mathsf{CID}}$-HVZK *and* (t, ϵ)-IMP-KOA-*secure identification protocol as defined in Figure 3.5. Then* CID *is* (t', ϵ', Q_T)-IMP-PA-*secure with*

$$t \approx t' \quad \text{and} \quad \epsilon' \leq \epsilon.$$

3.5.1.2 Applying the Fiat-Shamir transform

Now that we have shown that the identification protocol satisfies the prerequisites for the Fiat-Shamir transform, we can apply the transform to show that Ed25519 is existentially unforgeable.

We state the theorem in terms of the most basic version Ed25519-Original. For Ed25519-IETF and thus Ed25519-LibS the proof below only needs to account for the difference in the verification algorithm.

Theorem 4 ($\mathsf{Ed25519\text{-}Original}$ is $\mathsf{EUF\text{-}CMA}$ secure)**.** *Let* $\mathsf{CID} = (\mathsf{CID.KGen}, \mathsf{CID.P}, \mathsf{CID.V})$ *be the* $(t, \epsilon, Q_T, (Q_{\mathsf{H}} + Q_\kappa + Q_\beta))$-$\mathsf{IMP\text{-}PA}$*-secure identification protocol as defined in Figure 3.5 with* $\mathsf{H} : \{0,1\}^\star \to \{0,1\}^{2b}$ *modeled as a programmable random oracle and* α *bits min-entropy. Then* $\mathsf{Ed25519\text{-}Original} = \mathsf{FS}^{\mathsf{kp}}_{\mathsf{det}}[\mathsf{CID}, \mathsf{H}]$ *as defined in Figure 3.1 is* $(t', \epsilon', Q_S, (Q'_{\mathsf{H}} + Q_\kappa + Q_\beta))$-$\mathsf{EUF\text{-}CMA}$ *secure, where*

$$t \approx t' \quad \text{and} \quad \epsilon' \leq Q'_{\mathsf{H}} \cdot (\epsilon + Q_S Q'_{\mathsf{H}} 2^{-\alpha} + Q_\kappa 2^{-b})$$

for $Q_S = Q_T$ *and* $Q'_{\mathsf{H}} = Q_{\mathsf{H}} + 1$*, where* Q_κ *and* Q_β *refer to random oracle queries of a certain format (details in proof.)*

3.5.2 Strong Unforgeability of Standardized Ed25519

Our previous results confirm that for a target public key, the attacker is not able to forge a signature on a message m for which it has not seen valid signatures beforehand. In a real-world scenario, the security provided by existential unforgeability may be insufficient, as we have mentioned before, e.g., regarding Bitcoin transaction security or SSH multi-ciphersuite security. Another commonly named example is that of blocking certain public-key certificates. This could be achieved by storing the hash of the certificate in a list and comparing incoming certificates with this list. Here, a certificate can simply be viewed as a signature over a message, i.e., the contents of the certificate. An attacker wanting to bypass this blocking mechanism may create a new valid signature on the certificate, thereby altering its hash value that made the certificate efficiently recognizable by the filter. This is not prevented by existential unforgeability.

The security notion that bars attackers from forging new signatures on known (message, signature)-pairs is that of *strong* unforgeability, or $\mathsf{SUF\text{-}CMA}$ security, which is closely related to the concept of *malleability.* Malleable signatures retain their validity even if they are slightly changed, for example, by some bits being flipped. Obviously such signature schemes cannot hope to achieve strong unforgeability.

As mentioned earlier, $\mathsf{Ed25519\text{-}Original}$ without the check of $S \in \{0, \ldots, L-1\}$ during signature verification is not strongly unforgeable as any $S' \leftarrow S + mL$ with integer m also satisfies the verification equation.

For $\mathsf{Ed25519\text{-}IETF}$ and $\mathsf{Ed25519\text{-}LibS}$, this is avoided by additionally requiring that the decoded S already be reduced modulo L, leading to the rejection of values $S' \leftarrow S + mL$ during signature verification. The property that results from this additional check on the CID level is that of (computationally) unique responses of the identification protocol. Recall that this property guarantees that for a given commitment com and ch in the interaction $\mathsf{CID.P} \leftrightarrows \mathsf{CID.V}$, there exists (at most) one response rsp such that $(\mathsf{com}, \mathsf{ch}, \mathsf{rsp})$ is an accepting conversation (or a second response is only possible to find with probability at most ϵ_{cur}).

For the identification protocol underlying Ed25519-IETF, we in fact even have $\epsilon_{\mathsf{cur}} = 0$, i.e., for all $(\underline{A}, k) \xleftarrow{\$} \mathsf{CID.KGen}$, $(R = rB, s) \leftarrow \mathsf{CID.P}_1(k; r)$ and $\mathsf{ch} \xleftarrow{\$} \{0, 1\}^{2b}$, there exists only at (most) one valid response S. This confirms the RFC's argumentation that "Ed25519 [. . .] signatures are not malleable due to the verification check that decoded S is smaller than L" [121]. On an Ed25519-IETF signature level this then means that given a (message,signature)-pair $(m, (\underline{R}, \underline{S}))$ there exists no second signature $(\underline{R}, \underline{S}')$ with $\underline{S}' \neq \underline{S}$ as we will show in the next theorem. Furthermore, we will argue that there also cannot exist a second valid signature $(\underline{R}', \underline{S}')$ with $\underline{R}' \neq \underline{R}$ with $\underline{S}'$ different or equal to $\underline{S}$.

Theorem 5 (Ed25519-IETF is SUF-CMA secure)**.** *Let* Ed25519-IETF *be the* $(t, \epsilon, Q_S, Q_{\mathsf{H}})$-EUF-CMA *secure signature scheme derived by applying the Fiat-Shamir transform to the identification protocol* CID *given in Figure 3.5 with the check in Line 17. Then* Ed25519-IETF *is* $(t', \epsilon', Q'_S, Q'_{\mathsf{H}})$-SUF-CMA *secure with* $t \approx t'$ *and* $\epsilon' \leq \epsilon$.

3.5.3 Multi-User Security

We recall that a flaw in the tight reduction from multi-user security of signatures to the single-user case in [92] was exposed by Bernstein [40], who then was able to give an alternative tight reduction from the multi-user security of *key-prefixed* Schnorr to the single-user security of standard Schnorr. This result was taken as a justification for the much-debated employed key prefixing in Ed25519 signatures. Shortly after the result by Bernstein, Kiltz et al. [125] were able to provide a tight reduction in the random oracle model for general Fiat-Shamir signatures, assuming the property of random self-reducibility of the underlying identification protocol, further fueling the debate (though at this time the IETF standardisation of Ed25519-IETF had already been completed). Interestingly, when trying to apply either of the above results to Ed25519 signatures specifically, several peculiarities arise. The result by Bernstein [40] is transferable to Ed25519 signatures, but loses tightness. As explained in [40, Sec. 5.3] this is due to the clamping of secret keys in Ed25519 which yields an additional failure case in the reduction. The more general result by Kiltz et al. [125] on the other hand is not applicable at all, although Ed25519 is a Fiat-Shamir transformed signature scheme. This is precisely due to the key prefixing as this prohibits the achievement of the necessary random self-reducibility property. Consequently, only the non-tight bounds in [40, Sec. 5.3] apply to Ed25519.

3.5.4 Key Substitution Attacks

Key Substitution Attacks (KSA) were first introduced by Blake-Wilson and Menezes [52] and later formalized by Smart and Menezes [170]. Informally, KSA cover the scenario where an attacker learns one or more (message,signature)-pairs for a given public key, and wishes to find a different public key and message such that one of the valid signatures verify under the attacker's new public key. Maliciously generated public keys fall outside the traditional notions for signature security such as existential unforgeability. However, these attacks have practical consequences in real-world contexts: examples include an attack on the popular Let's Encrypt Certificate Issuance protocol that allowed an attacker to impersonate any website, compromise of confidentiality in the WS-Security Standard, and attacks on the well known Station to Station protocol [117].

Comparatively few publications have investigated key substitution attacks and how they apply to different signature schemes. Consequently, many signature schemes are vulnerable. For example, [170] described a KSA on the Gennaro-Halevi-Rabin signature scheme [93] and the standard-model secure scheme by Boneh and Boyen [56] also proved to be vulnerable [54]. While these schemes are more of an academic interest, [52, 54] also showed that under certain conditions the widely-deployed RSA, DSA, and ECDSA signatures are insecure against KSA attackers. Menezes and Smart also highlighted the relevance of key-substitution [170] to the multi-user setting.

In [100], a practical scheme for email authentication is proposed that requires the underlying signature scheme to be resistant to key substitution attacks. In this paper, it is stated that Schnorr signatures in prime order groups achieve SUF-CMA and resistance to key substitution attacks. The paper goes on to claim these results transfer to a variant of Ed25519 without key prefixing. We have already seen that this is not true in general and we also point out that it is not correct for their modified form of Ed25519. In particular, absent key prefixing, an attacker can submit a mangled public key lying outside the prime order group which is a distinct bitstring from the 'honest' signature yet passes the verification checks. Later in this section we consider Ed25519 with key-prefixing and find the opposite result, that this attack is provably prevented.

In the following, we investigate the resistance of $\mathsf{Ed25519}$ against various *exclusive ownership* definitions from [158] which rule out multiple key substitution attacks. Theorem Theorem 6 shows that $\mathsf{Ed25519}$ achieves this stronger version, cf. Definition 40, where the attacker is allowed to adaptively query the signing oracle to learn (message,signature)-pairs of its choice and may choose which signature to attack. Furthermore, we show in Theorem 7 that $\mathsf{Ed25519}$ has so-called *message-bound signatures* (cf. Definition 41), i.e., that there exist no two distinct messages for which the same signature would verify with respect to a given (potentially maliciously generated) public key. Lastly, Theorem 8 shows that if small order elements are rejected, even *malicious* strong universal exclusive ownership guarantees are provided.

Firstly, we find that an attacker cannot substitute an alternative public key to verify against an honest party's signature in any of the Ed25519 variants we have discussed.

Definition 40 (Strong Universal Exclusive Ownership)**.** *Let* $\mathsf{DS} = (\mathsf{DS.KGen}, \mathsf{DS.Sign}, \mathsf{DS.Vrfy})$ *be a signature scheme. Consider the security experiment* $\mathrm{Expr}_{\mathsf{DS}}^{\mathsf{S\text{-}UEO}}$ *as defined in Figure 3.7. We say that a signature scheme* DS *is* (t, ϵ, Q_S)-$\mathsf{S\text{-}UEO}$*-secure or* achieves

```
Expr_DS^S-UEO(A):
    $(vk, sk) \xleftarrow{\$} \mathsf{DS.KGen}(\mathsf{pp})$
    $((m,\sigma), vk', m') \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}}(vk)$
    return $[\![(m,\sigma) \in \mathcal{L}_{\mathsf{Sign}} \wedge vk \neq vk' \wedge \mathsf{DS.Vrfy}(vk', \sigma, m')]\!]$

O_Sign(m):
    $\sigma \xleftarrow{\$} \mathsf{DS.Sign}(sk, m)$
    $\mathcal{L}_{\mathsf{Sign}} \leftarrow \mathcal{L}_{\mathsf{Sign}} \cup \{(m,\sigma)\}$
    return $\sigma$
```

Figure 3.7: S-UEO experiment for a $\mathsf{DS} = (\mathsf{DS.KGen}, \mathsf{DS.Sign}, \mathsf{DS.Vrfy})$ scheme.

```
Expr_DS^MBS(A):
    $(vk, \sigma, m, m') \xleftarrow{\$} \mathcal{A}()$
    return $[\![m \neq m' \wedge \mathsf{DS.Vrfy}(vk, \sigma, m) \wedge \mathsf{DS.Vrfy}(vk, \sigma, m')]\!]$
```

Figure 3.8: MBS experiment for a $\mathsf{DS} = (\mathsf{DS.KGen}, \mathsf{DS.Sign}, \mathsf{DS.Vrfy})$ scheme.

strong universal exclusive ownership *if for any* PPT *attacker* $\mathcal{A}$ *making at most* Q_S *queries to the signing oracle, the probability*

$$\Pr\left[\mathrm{Expr}_{\mathsf{DS}}^{\mathsf{S\text{-}UEO}}(\mathcal{A}) = 1\right] \leq \epsilon.$$

Theorem 6 (Ed25519-Original achieves S-UEO)**.** *Let* $\mathsf{Ed25519} = (\mathsf{DS.KGen}, \mathsf{DS.Sign}, \mathsf{DS.Vrfy})$ *be as defined in Figure 3.1, where* $\mathsf{H} : \{0,1\}^\star \to \{0,1\}^{2b}$ *is modelled as a random oracle. Then* Ed25519 *is* $(\epsilon, Q_S, Q_{\mathsf{H}})$-S-UEO *secure,*

$$\epsilon \leq 2 \cdot Q_{\mathsf{H}} \cdot \lceil \frac{2^{2b}}{L} \rceil \cdot 2^{-2b}$$

where Q_S *and* Q_{H} *are the maximum number of queries to* $\mathcal{O}_{\mathsf{Sign}}$ *and* H.

We also find that, even when the signer is dishonest, Ed25519 schemes which reject public keys and signatures with low order elements, ensure that for a particular public key, signatures can only verify under a single message. However, if low order elements are accepted, an attacker can submit a low order element as their public key and any value for their signature such that $SB = R$. The resulting signature verifies under any message. This was pointed out in [43] but deemed unproblematic.

Definition 41 (Message Bound Signatures)**.** *Let* $\mathsf{DS} = (\mathsf{DS.KGen}, \mathsf{DS.Sign}, \mathsf{DS.Vrfy})$ *be a signature scheme. Consider the security experiment* $\mathrm{Expr}_{\mathsf{DS}}^{\mathsf{MBS}}$ *as defined in Figure 3.8. We say that a signature scheme* DS *is* ϵ-MBS-*secure or* achieves message bound signatures *if for any* PPT *attacker* $\mathcal{A}$ *the probability:*

$$\Pr\left[\mathrm{Expr}_{\mathsf{DS}}^{\mathsf{MBS}}(\mathcal{A}) = 1\right] \leq \epsilon.$$

Theorem 7 (Ed25519-LibS achieves MBS)**.** *Let* $\mathsf{Ed25519} = (\mathsf{DS.KGen}, \mathsf{DS.Sign}, \mathsf{DS.Vrfy})$ *be as defined in Figure 3.1 and the hash function* $H : \{0,1\}^* \to \{0,1\}^{2b}$ *is a random oracle. If the small subgroup elements are rejected, then* DS *is* ϵ'-MBS-*secure with*

```
$\mathrm{Expr}^{\mathsf{M\text{-}S\text{-}UEO}}_{\mathsf{DS}}(\mathcal{A})$:
$(vk, vk', \sigma, m, m') \xleftarrow{\$} \mathcal{A}()$
return $[\![vk \neq vk' \wedge \mathsf{DS.Vrfy}(vk, \sigma, m) \wedge \mathsf{DS.Vrfy}(vk', \sigma, m')]\!]$
```

Figure 3.9: M-S-UEO experiment for a DS = (DS.KGen, DS.Sign, DS.Vrfy) scheme.

$$\epsilon' \leq \lceil \frac{2^{2b}}{L} \rceil \cdot 2^{-2b} \cdot (Q_H + 2)^2,$$

where Q_H is the maximal number of queries to the random oracle.

We now consider a stronger variant of S-UEO where the attacker collaborates or compromises with the signer, in order to generate a signature valid under two distinct public keys. Provided small subgroup elements are rejected, this property also holds. However, there is a straightforward attack if they are accepted where the attacker chooses its public key to be two distinct low order elements.

Definition 42 (Malicious Strong Universal Exclusive Ownership)**.** *Let* DS = (DS.KGen, DS.Sign, DS.Vrfy) *be a signature scheme. Consider the security experiment* $\mathrm{Expr}^{\mathsf{M\text{-}S\text{-}UEO}}_{\mathsf{DS}}$ *as defined in Figure 3.9. We say that a signature scheme* DS *is* ϵ-M-S-UEO*-secure or* malicious strong universal exclusive ownership *if for any* PPT *attacker* $\mathcal{A}$ *the probability:*

$$\Pr\left[\mathrm{Expr}^{\mathsf{M\text{-}S\text{-}UEO}}_{\mathsf{DS}}(\mathcal{A}) = 1\right] \leq \epsilon.$$

Theorem 8 (Ed25519-LibS achieves M-S-UEO)**.** *Let* DS = (DS.KGen, DS.Sign, DS.Vrfy) *be as defined in Figure 3.1, with the* Ed25519-LibS *variant and the hash function $H : \{0,1\}^* \rightarrow \{0,1\}^{2b}$ is a random oracle. Then* DS *is* ϵ'-M-S-UEO*-secure with* $\epsilon' \leq \frac{\lceil 2^{2b}/L \rceil}{2^{2b}} \cdot {Q_h}^2$.

3.6 Conclusions

We proved that Ed25519 achieves its goal of existential unforgeability (EUF-CMA), as is assumed by many published works. While Ed25519 seems similar to Fiat-Shamir applied to the Schnorr identification scheme, the devil is in the detail. We took into account the non-prime order group, the clamping of private scalars and many other details.

Moreover, we also proved that Ed25519-IETF achieves SUF-CMA. We proved that all Ed25519 schemes resilient against key substitution attacks, however, we also showed that rejecting small order elements does yield additional properties, enabling Ed25519-LibS to achieve an even stronger form of key substitution resilience as well as message bound security.

Our results, summarized in Table 3.1 on page 28, thereby provide not only theoretical foundations, but also meaningful insights for choosing among the variants.

3.7 Full Proofs

Below, we provide details proofs for all lemmas and theorems in this chapter.

3.7.1 Proof for Theorem 2

Proof. Assume there exists an attacker $\mathcal{A}$ against the (t', ϵ')-$\mathsf{IMP\text{-}KOA}$ security of CID. From this we then construct another attacker $\mathcal{B}$, the reduction, that can solve the $(t, \epsilon_{\mathsf{EC}}^{\mathsf{DLP}})$-$\mathsf{ECDLP}$ on $\mathsf{EC}(\mathbb{F}_q)$. Since we assume ECDLP to be hard to break with non-negligible probability $\epsilon_{\mathsf{EC}}^{\mathsf{DLP}}$ in any reasonable amount of time t, this then leads to a contradiction, yielding $\mathsf{IMP\text{-}KOA}$ security. $\mathcal{B}$ is constructed as follows: $\mathcal{B}$ gets as input the base point $B \in \mathsf{EC}(\mathbb{F}_q)$ of order L, as well as a random point $A \in \langle B \rangle$. Then, $\mathcal{B}$ runs $\mathcal{A}$ on input $\underline{A}$.

We first analyze the probability of the received value A falling into the subset of correctly distributed public keys to invoke $\mathcal{A}$. Note that the public key $\underline{A}$ in the identification protocol is computed as $A \leftarrow sB$ with an $s \in \{2^{b-2}, 2^{b-2} + 8, \ldots, 2^{b-1} - 8\}$. We claim that for any public keys $A_1 = s_1 B, A_2 = s_2 B$ with $s_1, s_2 \in \{2^{b-2}, 2^{b-2}+8, \ldots, 2^{b-1}-8\}$, $A_1 = A_2$ if and only if $s_1 = s_2$. Note that for above s_1, s_2, there must exist $i_1, i_2 \in \{0, ..., 2^{b-5} - 1\}$ such that $s_1 = 2^{b-2} + 8i_1$ and $s_2 = 2^{b-2} + 8i_2$. Since $L > 2^{b-5}$ is a prime, it holds that

$$\begin{aligned}
A_1 = A_2 &\Leftrightarrow s_1 B = s_2 B \\
&\Leftrightarrow s_1 = s_2 \mod L \\
&\Leftrightarrow 2^{b-2} + 8i_1 = 2^{b-2} + 8i_2 \mod L \\
&\Leftrightarrow i_1 = i_2 \mod L \\
&\Leftrightarrow i_1 = i_2 \\
&\Leftrightarrow 2^{b-2} + 8i_1 = 2^{b-2} + 8i_2 \\
&\Leftrightarrow s_1 = s_2
\end{aligned}$$

The above claim indicates that the cardinality of the set of valid public keys equals 2^{b-5}. Recall that the point $A \in \langle B \rangle$ is uniformly at random. The probability of $\underline{A}$ being a valid public key from $\mathcal{A}$'s view is therefore bounded by $\frac{2^{b-5}}{L}$. In particular, substituting the instantiation of $\mathsf{Ed25519}$ for the corresponding parameters, it holds that $\frac{2^{b-5}}{L} = \frac{2^{251}}{2^{252}+27742...8493} \approx \frac{1}{2}$, which is obviously non-negligible.

Challenge: At some point $\mathcal{A}$ outputs a commitment $\underline{R}^\star$ to its challenger. $\mathcal{B}$ then chooses a random challenge $\mathsf{ch}_1 \xleftarrow{\$} \{0,1\}^{2b}$ and sends ch_1 to $\mathcal{A}$. Finally, $\mathcal{A}$ terminates with output $\underline{S}_1$.

Resetting the attacker: Then, $\mathcal{B}$ resets $\mathcal{A}$'s internal state back to the point just after which it generated $\underline{R}^\star$ and returns a newly sampled challenge value $\mathsf{ch}_2 \xleftarrow{\$} \{0,1\}^{2b}$ to $\mathcal{A}$ with $\mathsf{ch}_1 \neq \mathsf{ch}_2 \mod L$.

Finally, again, $\mathcal{A}$ will output a response $\underline{S}_2$. $\mathcal{B}$ verifies whether $(\underline{R}^\star, \mathsf{ch}_1, \underline{S}_1)$ and $(\underline{R}^\star, \mathsf{ch}_2, \underline{S}_2)$ both are accepting conversations with $\mathsf{ch}_1 \neq \mathsf{ch}_2 \pmod L$, and aborts if this condition is not satisfied.

By the so-called *Reset Lemma* [32], we know that if $\mathcal{A}$ can find an $\underline{S}_1$ such that $(\underline{R}^\star, \mathsf{ch}_1, \underline{S}_1)$ is an accepting conversation with probability ϵ', then a reset of $\mathcal{A}$ with the same random tape will output an accepting conversation $(\underline{R}^\star, \mathsf{ch}_2, \underline{S}_2)$ for $\mathsf{ch}_1 \neq \mathsf{ch}_2 \pmod L$ with probability at least $(\epsilon' - \frac{1}{L})^2$.

$\mathcal{B}$ then outputs $s = \frac{S_1 - S_2}{\mathsf{ch}_1 - \mathsf{ch}_2} \mod L$.

Assume that $(\underline{R}^\star, \mathsf{ch}_1, \underline{S}_1)$ and $(\underline{R}^\star, \mathsf{ch}_2, \underline{S}_2)$ are accepting conversations with $\mathsf{ch}_1 \neq \mathsf{ch}_2 \pmod L$. In particular, it holds that $S_i \in \{0, ..., L-1\}$ and $8S_iB = 8R^\star + 8\mathsf{ch}_iA$ for $i \in \{1, 2\}$, which implies that

$$\begin{aligned} & 8(S_1 - S_2)B = 8(\mathsf{ch}_1 - \mathsf{ch}_2)A \\ \Leftrightarrow \quad & (S_1 - S_2) \cdot (\mathsf{ch}_1 - \mathsf{ch}_2)^{-1}B = A \end{aligned}$$

Therefore, $s = \frac{S_1 - S_2}{\mathsf{ch}_1 - \mathsf{ch}_2} \mod L$ is the desired solution to the ECDLP instance (B, A). Regarding the time complexity, it holds that $t \approx 2t'$, as $\mathcal{B}$ rewound $\mathcal{A}$'s internal state once. Finally, we can deduce that the probability of $\mathcal{B}$ successfully extracting the discrete logarithm is at least $\frac{2^{b-5}}{L}(\epsilon' - \frac{1}{L})^2$. ☐

3.7.2 Proof for Lemma 1

Proof. We must show that the conversations $(\underline{R}, \mathsf{ch}, \underline{S}) \xleftarrow{\$} \mathsf{Sim}(\underline{A})$ are distributed identically to $\mathsf{Trans}\,[\mathsf{CID.P}(k) \leftrightarrows \mathsf{CID.V}(\underline{A})]$ in honest executions of CID.

In the following let $(\mathsf{com}, \mathsf{ch}, \mathsf{rsp})$ be a valid honest execution between the prover and the verifier. It holds that com is the encoding of an element rB in the elliptic curve group with $r \xleftarrow{\$} \{0,1\}^{2b}$, $\mathsf{ch} \xleftarrow{\$} \{0,1\}^{2b}$ and rsp is the encoding of an element in $\{0, ..., L-1\}$ of the form $(r + \mathsf{ch} \cdot s) \mod L$, with s implicitly fixed by the decoding of $\underline{A} = \underline{sB}$.

Clearly, the challenges are distributed identically in both conversations. The (decoded) simulated responses $S \leftarrow \tilde{s}$ with $\tilde{s} \xleftarrow{\$} \{0,1\}^{2b}$ are also distributed identically to real responses $\mathsf{rsp} = (r + \mathsf{ch} \cdot s) \mod L$, with $r, \mathsf{ch} \xleftarrow{\$} \{0,1\}^{2b}$. The same holds for the (decoded) simulated commitments $R \leftarrow (SB - \mathsf{ch}A) \mod L = (S - \mathsf{ch} \cdot s)B \mod L$ and the real commitments com, since the latter are in the elliptic curve group of the form $\mathsf{com} \leftarrow rB$ with $r \xleftarrow{\$} \{0,1\}^{2b}$, which is equivalent to $r'B$ with $r' \leftarrow r \mod L$. ☐

3.7.3 Proof for Theorem 3

Proof. We have shown in Lemma 1 that CID is ϵ_{zk}-HVZK with $\epsilon_{\mathsf{zk}} = 0$. Since $\mathsf{Sim}(pk)$ uses public information only, any resulting conversations could also have been computed by $\mathcal{A}$ itself. $\mathcal{A}$ therefore learns nothing from the interaction of $\mathsf{CID.P} \leftrightarrows \mathsf{CID.V}$ via $\mathcal{O}_{\mathsf{Trans}}$

(replaced by Sim). Thus, for canonical identification protocols that are HVZK, $\mathsf{IMP\text{-}PA}$ security is equivalent to $\mathsf{IMP\text{-}KOA}$ security and we have ave $\epsilon' \leq \epsilon$ and $t \approx t'$ plus the running time of the Sim at most Q_T times. Since t is dominated by t', we simply write $t \approx t'$. □

3.7.4 Proof for Theorem 4

Proof. In the following we assume without loss of generality, that the attacker $\mathcal{A}$ never queries the same message twice to the oracle $\mathcal{O}_{\mathsf{Sign}}$, since $\mathsf{Ed25519\text{-}Original}$ is deterministic and thus $\mathcal{A}$ does not gain any advantage in doing so.

Assume there exists a $(t', \epsilon', Q_S, (Q'_{\mathsf{H}}+Q_\kappa+Q_\beta))$-attacker $\mathcal{A}$ against the unforgeability of the signature scheme $\mathsf{Ed25519\text{-}Original}$. We show that this immediately implies a successful $(t, \epsilon, Q_T, (Q_{\mathsf{H}} + Q_\kappa + Q_\beta))$ attacker $\mathcal{B}$ against the $\mathsf{IMP\text{-}PA}$ security of the underlying identification scheme CID. The reduction $\mathcal{B}$ receives as input a public key $\underline{A}$. $\mathcal{B}$ then runs $\mathcal{A}$ on input $\underline{A}$ as follows.

We note that the relevant random oracle queries of $\mathcal{A}$ can take three distinct and distinguishable forms: the most relevant to the reduction are those of the form $(\underline{\mathsf{com}}, \underline{A}, m)$, i.e., a b-bit string followed by the encoding $\underline{A}$, and some arbitrary-length bit-string m. The second distinct case are those queries of length exactly b bits. Any other query can be interpreted as a query of the form (β, m) with β a b-bit string and m some arbitrary-length bit-string.

Random oracle queries of the form κ: Let κ be a b-bit string. Let Q_κ be the maximum number of queries of the form κ that $\mathcal{A}$ makes to H. The reduction $\mathcal{B}$ simply forwards these queries to H and returns the answer to $\mathcal{A}$.

Random oracle queries of the form (β, m): Let β be a bit string of length b and let Q_β be the number of queries of the form (β, m) that $\mathcal{A}$ makes to H. Again, $\mathcal{B}$ simply relays these queries between H and $\mathcal{A}$.

Random oracle queries of the form $(\underline{\mathsf{com}}, \underline{A}, m)$: Let Q'_{H} be the (maximal) number of queries of this form that $\mathcal{A}$ makes to the random oracle H. $\mathcal{B}$ guesses the query $i \in \{1, 2, \ldots, Q'_{\mathsf{H}}\}$ for which $\mathcal{A}$ will eventually output the signature forgery, resulting in a loss of a factor Q'_{H}.

For every of the other $Q_{\mathsf{H}} = Q'_{\mathsf{H}} - 1$ queries $j \in \{1, 2, \ldots Q'_{\mathsf{H}}\}$ with $j \neq i$ to H, $\mathcal{B}$ simply relays the queries between $\mathcal{A}$ and H.

When the attacker $\mathcal{A}$ asks the i-th query, say on $(\underline{\mathsf{com}}', \underline{A}, m')$, $\mathcal{B}$ forwards $\underline{\mathsf{com}}'$ as its commitment to its own challenger. The challenger will then send a challenge $\mathsf{ch}^\star \xleftarrow{\$} \{0,1\}^{2b}$ to $\mathcal{B}$, which $\mathcal{B}$ returns as response to $\mathcal{A}$.

Signing queries: For every of the Q_S signing queries of $\mathcal{A}$, $\mathcal{B}$ runs its own $\mathcal{O}_{\mathsf{Trans}}$ oracle to obtain an accepting conversation $(\underline{R}, \mathsf{ch}, \underline{S})$. In order for $(\underline{R}, \underline{S})$ to be a valid signature on m, $\mathcal{B}$ must ensure that $\mathsf{ch} = \mathsf{H}(\underline{R}, \underline{A}, m)$, i.e., the reduction programs the random oracle on these values to return ch.

Note that for each commitment value $\underline{R}$ that was output by $\mathcal{O}_{\mathsf{Trans}}$, the probability that the value $\mathsf{H}(\underline{R}, \underline{A}, m)$ had been set by a previous query of $\mathcal{A}$ to the random oracle H, and thus that $\mathcal{A}$ could detect the inconsistency in the patched random oracle, is upper bounded by $Q'_{\mathsf{H}}2^{-\alpha}$, where α is the min-entropy of the identification scheme. The distribution of commitments has α bits of min-entropy for $\alpha = -\log_2\left(\lceil\frac{2^{2b}-1}{L}\rceil \cdot 2^{-2b}\right)$ due to the bias introduced by sampling first a uniformly random $2b$-bit string and then reducing it modulo L.

If this happens, the reduction $\mathcal{B}$ aborts, the probability of which is thus upper bounded by $Q_S Q'_{\mathsf{H}} \cdot 2^{-\alpha}$.

Furthermore, note that this also implies that the value $\underline{R}$ provided by the simulation via $\mathcal{O}_{\mathsf{Trans}}$ is with high probability different from the deterministic value $\underline{R}'$ with $R' = r'B$ that would be generated in the real signing process of the message m with secret key k belonging to the public key $\underline{A}$. However, $\mathcal{A}$ is not able to compute the deterministic commitment value $\underline{R}'$ by itself unless it can guess the correct value k to determine $h[b], ...h[2b-1]$ of $h \leftarrow \mathsf{H}(k)$ and thus r', the probability of which is bounded by $Q_\kappa 2^{-b}$.

Note that it does not help $\mathcal{A}$ in detecting the simulation to guess the values $h[b], ...h[2b-1]$, as $\mathcal{A}$ has no way of checking that these are the correct values leading to the "real" r' without also guessing k.

Existential Forgery: At some point $\mathcal{A}$ terminates with a forgery output $(m^\star, \sigma^\star = (\underline{R}^\star, \underline{S}^\star))$ with $\underline{R}^\star = \mathsf{com}'$ and $m^\star = m'$. If this is not the case, $\mathcal{B}$ aborts with probability $\frac{1}{Q'_{\mathsf{H}}}$ since it has wrongly guessed the index i for which the forgery will take place. Assuming this is a valid forgery in the $\mathsf{EUF\text{-}CMA}$ game, it holds that m' has not been queried to $\mathcal{O}_{\mathsf{Sign}}$. In particular this means that the output of $\mathsf{H}(\mathsf{com}', \underline{A}, m')$ has not been re-programmed in $\mathcal{A}$'s view by $\mathcal{B}$ to a value other than $\mathsf{ch}^\star$. Furthermore, $\mathsf{CID.V}_2(\underline{A}, \mathsf{com}', \mathsf{ch}^\star, \underline{S}^\star) = 1$ holds such that when $\mathcal{B}$ forwards $\underline{S}^\star$ to its own challenger as final output of its game, $\mathcal{B}$ will also be successful.

The running time t of $\mathcal{B}$ is that of $\mathcal{A}$ plus the time it takes to query the random oracle H $(Q'_{\mathsf{H}} + Q_\kappa + Q_\beta)$ times, the time it takes to query its challenger, and to query $\mathcal{O}_{\mathsf{Trans}}$ $Q_T = Q_S$ times. As before, we write $t \approx t'$ since the running time of the reduction is dominated by the running time t' of $\mathcal{A}$. If $\mathcal{A}$ outputs a forgery with probability ϵ', then $\mathcal{B}$ will be able to impersonate the prover with probability $\frac{\epsilon'}{Q'_{\mathsf{H}}} - Q_S Q'_{\mathsf{H}} \cdot 2^{-\alpha} - Q_\kappa 2^{-b}$. $\square$

3.7.5 Proof for Theorem 5

Proof. We recall that the games of EUF-CMA and SUF-CMA only differ in the winning condition for the attacker $\mathcal{A}$: EUF-CMA forbids that the attacker has queried the signing oracle $\mathcal{O}_{\mathsf{Sign}}$ on the message $m^\star$ for which it outputs the forgery, whereas SUF-CMA allows this and only requests that the signature forgery $\sigma^\star$ differs from the signature σ that was output by $\mathcal{O}_{\mathsf{Sign}}(m^\star)$.

We therefore focus on the case that the attacker $\mathcal{A}$ on input an Ed25519-IETF public key $\underline{A}$ outputs a valid strong forgery $(m^\star, \sigma^\star = (\underline{R}^\star, \underline{S}^\star))$ with probability ϵ' such that there exists an entry $(m^\star, \sigma' = (\underline{R}', \underline{S}')) \in \mathcal{L}_{\mathsf{Sign}}$ of recorded $\mathcal{O}_{\mathsf{Sign}}$ queries of $\mathcal{A}$, such that $\sigma^\star \neq \sigma'$. Since encodings are deterministic and unique [2], we omit them in the following discussion. Naturally, there are two ways in which $\sigma^\star = (\underline{R}^\star, \underline{S}^\star) \neq \sigma' = (\underline{R}', \underline{S}')$ for the same message $m^\star$:

Case 1 $R^\star \neq R'$: In this case, we can immediately build a reduction $\mathcal{B}$ against the EUF-CMA security of Ed25519-IETF. The reduction $\mathcal{B}$ gets as input a public key $\underline{A}$ and invokes the SUF-CMA attacker $\mathcal{A}(\underline{A})$. For any of the maximal Q'_S signing queries of $\mathcal{A}$ on message m, $\mathcal{B}$ simply uses the strategy of the simulator Sim (cf. Fig. Figure 3.6) to obtain valid conversations $(\underline{R}, \mathsf{ch}, \underline{S})$ and patches the random oracle H to return ch on input $(\underline{R}, \underline{A}, m)$. As before, this programming ensures that the response $(\underline{R}, \underline{S})$ to $\mathcal{A}$ is a valid signature from $\mathcal{A}$'s point of view. The at most Q'_{H} random oracle queries of $\mathcal{A}$ are simulated by $\mathcal{B}$ relaying queries to the "real" random oracle H on any inputs that had not been patched by a signature query, the latter are answered consistently with the patching. Note that to run the simulation of $\mathcal{A}$, $\mathcal{B}$ has made no signing query to its own $\mathcal{O}_{\mathsf{Sign}}$. Thus, once $\mathcal{A}$ outputs its strong forgery $(m^\star, \sigma^\star)$, $\mathcal{B}$ can immediately output the same pair as its existential forgery. Note that since $R^\star \neq R'$, H has not been patched by $\mathcal{B}$ on $(\underline{R}^\star, \underline{A}, m^\star)$.

Case 2 $S^\star \neq S'$: This leaves the possibility that $R^\star = R'$, but $S^\star \neq S'$, i.e., the strongly forged signature is of the form $(R', S^\star)$. We will argue that this also is not possible, as this contradicts the uniqueness of the underlying identification protocol: For CID it holds that there is only (at most) one valid response $\underline{S}$ for all $(\underline{A}, k) \xleftarrow{\$} \mathsf{CID.DS.KGen}$, $(R = rB, s) \leftarrow \mathsf{CID.P}_1(k; r)$ and $\mathsf{ch} \xleftarrow{\$} \{0,1\}^{2b}$. Assume otherwise, i.e., there exist $S \neq S'$ such that both $(\underline{R}, \mathsf{ch}, \underline{S})$ and $(\underline{R}, \mathsf{ch}, \underline{S}')$ are valid conversations wrt. the public key $\underline{A}$. To pass verification via $\mathsf{CID.V}_2$ it must hold that $S, S' \in \{0, ..., L-1\}$ and furthermore $8SB = 8R + 8\mathsf{ch}A$ and $8S'B = 8R + 8\mathsf{ch}A$, or, equivalently, $8SB = 8S'B$, contradicting the assumption that $S \neq S'$. Since verification of an Ed25519-IETF signature

[2] Note that there is a very small probability that there exist two different encodings $\underline{R_1}, \underline{R_2}$ such that they decode to the same element R. This is due to the fact that elements in $\mathsf{EC}(\mathbb{F}_q)$ are encoded as b bit strings with a $(b-1)$-bit encoding for the y coordinate, plus one bit for the sign of x. Thus the entire valid encoding space for the y coordinate encompasses integers from 0 to $2^{b-1} - 1 = 2^{255} - 1$, whereas $\mathbb{F}_q$ contains only the integers from 0 to $2^{255} - 20$. Nevertheless, Case 1 in the reduction also captures this.

$(\underline{R}, \underline{S})$ on message m for public key $\underline{A}$ is just executing the verifier $\mathsf{CID.V}_2$ on input $(\underline{A}, \underline{R}, \mathsf{H}(\underline{R}, \underline{A}, m^\star), \underline{S})$ it is clear by the same argument that there cannot be a strong forgery with $S^\star \neq S'$.

To conclude, since the probability of computing non-unique responses in CID is 0 and the signature scheme does not admit existential forgeries with non-negligible probability, the probability of an attacker $\mathcal{A}$ succeeding against the strong unforgeability is also negligible. □

3.7.6 Proof for Theorem 6

Proof. Assume there exists an attacker $\mathcal{A}$ that can break the (ϵ, Q_S)-$\mathsf{S\text{-}UEO}$ security of $\mathsf{Ed25519}$. This means that for an honestly generated key pair $(\underline{A}, k) \xleftarrow{\$} \mathsf{DS.KGen}$, given the public key $\underline{A}$, $\mathcal{A}$ can output $((m, \sigma = (\underline{R}, \underline{S})), \underline{A}', m')$ such that:

1. $\sigma \leftarrow \mathsf{Sign}(k, m)$ for one of the Q_S signing queries of $\mathcal{A}$. In particular, we have $8SB = 8R + 8\mathsf{H}(\underline{R}, \underline{A}, m)A$.
2. $\underline{A}' \neq \underline{A}$, which means that $A' \neq A$.
3. Verification $\mathsf{DS.Vrfy}(\underline{A}', \sigma, m')$ holds, i.e., $8SB = 8R + 8\mathsf{H}(\underline{R}, \underline{A}', m')A'$.

Let $\mathsf{ch} \leftarrow \mathsf{H}(\underline{R}, \underline{A}, m)$ and $\mathsf{ch}' \leftarrow \mathsf{H}(\underline{R}, \underline{A}', m')$. Observing property 2, properties 1) and 3) can only hold simultaneously if and only if one of the following (distinct) cases arises:

Case 1: It holds that $SB = R$ in property 1). Then $\mathcal{A}$ can simply output a low order point $\underline{A}'$, i.e., with $|A'| \leq L$ and $m' = m$, which causes property 3) to also collapse to $SB = R$, irrespective of the value ch'. This can only happen in property 1) if $\mathsf{ch} = 0 \pmod L$ or $\mathsf{ch} = s^{-1} \pmod L$ for $A = sB$. But since H is a random oracle, this happens only with probability at most $2 \cdot Q_H \cdot \left(\lceil \frac{2^{2b}}{L} \rceil \cdot 2^{-2b}\right)$. So in the following we have $SB \neq R$.

Case 2 $\mathcal{A}$ can guess $A' \neq A$ with $|A'| \geq L$ and m' such that $\mathsf{ch}'A' = \mathsf{ch}A$. But, again, since H is a random oracle, the probability of this succeeding, accounting for the attacker's ability to repeat the process, is bounded by $Q_H \cdot \lceil \frac{2^{2b}}{L} \rceil \cdot 2^{-2b}$.

□

3.7.7 Proof for Theorem 7

Proof. Let $\mathcal{A}$ denote an attacker against (ϵ')-MBS security of $\mathsf{Ed25519}$. We then give the concrete upper bound of ϵ' in the random oracle model. Assume that $\mathcal{A}$ terminates with $(vk = \underline{A}, \sigma = (\underline{R}, \underline{S}), m, m')$ and wins the MBS experiment in Fig. 3.8. Then, it holds that $m \neq m'$, $8SB = 8R + 8\mathsf{H}(\underline{R}, \underline{A}, m)A$, and $8SB = 8R + 8\mathsf{H}(\underline{R}, \underline{A}, m')A$, which further implies that $8\mathsf{H}(\underline{R}, \underline{A}, m)A = 8\mathsf{H}(\underline{R}, \underline{A}, m')A$. Note that A is not a small subgroup element, it must hold that

$$\mathsf{H}(\underline{R}, \underline{A}, m) = \mathsf{H}(\underline{R}, \underline{A}, m') \mod L, \quad m \neq m'. \tag{3.2}$$

Obviously, we have $\epsilon' \leq \Pr[\text{Eq.}(3.2) \text{ holds}]$. Note that the random oracle in the MBS experiment will evaluate at most $(Q_H + 2)$ different inputs, where at most Q_H ones are queried by $\mathcal{A}$ and two are queried by the challenger for final verifications. Moreover, Eq. (3.2) holds only if there exists two outputs of the random oracle on different inputs such that the outputs are congruent modulo L, which occurs with probability from above bounded by $\lceil \frac{2^{2b}}{L} \rceil \cdot 2^{-2b} \cdot (Q_H + 2)^2$. Hence, it holds that $\epsilon' \leq \lceil \frac{2^{2b}}{L} \rceil \cdot 2^{-2b} \cdot (Q_H + 2)^2$. □

3.7.8 Proof for Theorem 8

Proof. It follows from the verification equation that:

$$8\mathsf{H}(R, vk', m')vk' = 8\mathsf{H}(R, vk, m)vk$$

It then follows that from the rejection of small subgroup elements that:

$$\mathsf{H}(R, vk', m')a' = \mathsf{H}(R, vk, m)a \mod L \tag{3.3}$$

As a is in the range $1, \ldots, L$ and thus coprime to L it follows that

$$\mathsf{H}(R, vk', m')a'(a)^{-1} = \mathsf{H}(R, vk, m) \mod L \tag{3.4}$$

We fix a, m', a', then for a particular m, $\mathsf{H}(R, vk, m)$ is in the range $0, \ldots, 2^{2b}$ of which there are at most $\lceil 2^{2b}/L \rceil$ values such that the equation holds. Consequently a given guess has probability $\frac{\lceil 2^{2b}/L \rceil}{2^{2b}}$ of fulfilling the equation. However, the attacker can also vary m' and consequently perform a collision attack. Notice that the attacker can make up to Q_h queries and consequently the overall probability of success is bounded above by $\frac{\lceil 2^{2b}/L \rceil}{2^{2b}} \cdot {Q_h}^2$ □

Chapter 4

Exploration of AEAD Landscape

This chapter is based on the paper:

Cas Cremers, Alexander Dax, Charlie Jacomme, and Mang Zhao. Automated Analysis of Protocols that use Authenticated Encryption: Analysing the Impact of the Subtle Differences between AEADs on Protocol Security. In USENIX Security 2023, 2023 Aug 9.

This paper won the “Distinguished Paper Award” in Usenix 2023. I lead the research on the generalization of AEAD collision resistance and its relations to other security properties in the computational model in this paper and the substantial contributions in this chapter are my own.

My co-authors Cas Cremers, Alexander Dax, and Charlie Jacomme principally contributed to the initial conception of the work, the further exploration in the symbolic model, and the write up of the connection between computational and symbolic analyses.

4.1 Introduction

Authenticated Encryption (AE) and Authenticated Encryption with Associated Data (AEAD) are some of the most commonly used cryptographic building blocks. AEAD primitives are built from symmetric encryption primitives and augmented with authentication mechanisms. Their applications include the vast majority of encrypted internet data, such as in TLS, WPA2 from IEEE 802.11 (WiFi), WireGuard, and by messaging apps such as Signal or WhatsApp. For example, in TLS, TLSCiphertext is constructed from an AEAD applied to a header and payload: both are authenticated, but only the payload is encrypted, and the plaintext header includes the content type and the ciphertext length.

Historically, AEAD was formally introduced by Rogaway [163] to entwine privacy and authenticity for both messages and headers in a single and compact mode. The definition of AEAD is given in the nonce-based pattern, where the nonce is named after the ***n***umber that are supposed to be used only ***once***. The nonce-based AEADs are expected to relax the security requirements of the randomized or counter-based pattern – ensuring no reuse of the nonce during the encryption is sufficient for privacy and authenticity. Moreover, Bellare and Hoang [27] initialized the study on binding keys and other optional inputs to the ciphertexts.

While AEADs are ubiquitous in modern secure communications, there is no commonly agreed "strong" security notion that they should satisfy. In fact, the current landscape of security notions for AEADs is rather divergent and chaotic: there are many proposed frameworks and security notion variants [4, 10, 11, 23, 27, 31, 55, 67, 82, 97, 98, 135, 163, 168]. For some of these notions, their implication relations are known [23], but many of them are hard to compare for technical reasons.

To address this, we revisit several recent cryptographic AEAD definitions, extract the core requirement *collision resistance* for a generic computational AEAD model, and illustrate how satisfying collision resistance implies that many existing security notions are met. Our generic computational AEAD model enables us to develop a family of symbolic AEAD models that can be used with symbolic protocol analysis tools, e.g., the Tamarin prover, for further case studies or independent research interests.

Contribution Our main contributions are the following:

- We formally prove some well-known but merely conjectured relations for AEAD between the fundamental privacy and integrity.
- We formally prove the missing or conjectured relations between existing AEAD security notions w.r.t. collision resistance, completing the picture in the domain.

Overview We first prove some well-known but merely conjectured relations for AEAD between the fundamental privacy and integrity in Section 4.2. Then, we define and generalize a novel collision resistance notion and show its relationship with other existing security notions in Section 4.3. We conclude in Section 4.4 and give the full proofs of all theorems in this chapter in Section 4.5.

4.2 Relations between Privacy and Integrity

The canonical privacy and integrity notions are $\mathsf{IND\$\text{-}CPA}$ and $\mathsf{CTI\text{-}CPA}$, which are respectively extended to $\mathsf{IND\$\text{-}CCA}$ and $\mathsf{CTI\text{-}CCA}$ security, as we introduced in Section 2.2.4. The relation among them has been explored in [23]. Below, we recall the results.

Theorem 9 ([23])**.** *Let* $\mathsf{AEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$ *be an authenticated encryption with associated data. It holds that:*

1. *[23, Proposition 1]: If* AEAD *is* ϵ-$\mathsf{CTI\text{-}CPA}$ *secure, then* AEAD *is also* ϵ-$\mathsf{CTI\text{-}CCA}$ *secure, and vice versa.*
2. *[23, Theorem 6]: If* AEAD *is* $\epsilon_{\mathsf{AEAD}}^{\mathsf{ind\$\text{-}cpa}}$-$\mathsf{IND\$\text{-}CPA}$ *secure and* $\epsilon_{\mathsf{AEAD}}^{\mathsf{cti\text{-}cpa}}$-$\mathsf{CTI\text{-}CPA}$ *secure, then* AEAD *is also* $\epsilon_{\mathsf{AEAD}}^{\mathsf{ind\$\text{-}cca}}$-$\mathsf{IND\$\text{-}CCA}$ *secure such that*

$$\epsilon_{\mathsf{AEAD}}^{\mathsf{ind\$\text{-}cca}} \leq 2(\epsilon_{\mathsf{AEAD}}^{\mathsf{ind\$\text{-}cpa}} + \epsilon_{\mathsf{AEAD}}^{\mathsf{cti\text{-}cpa}})$$

Moreover, we also have some trivial results for the missing relations between the privacy and authenticity notions. We illustrate the full relations between privacy and integrity in Figure 4.1.

Theorem 10. *Let* $\mathsf{AEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$ *be an authenticated encryption with associated data. Then, it holds that:*

1. *If* AEAD *is* ϵ-$\mathsf{IND\$\text{-}CCA}$ *secure, then* AEAD *is also* ϵ-$\mathsf{IND\$\text{-}CPA}$ *secure.*
2. *[23]: If* AEAD *is* ϵ-$\mathsf{IND\$\text{-}CPA}$ *secure, then* AEAD *might not be* ϵ'-$\mathsf{IND\$\text{-}CCA}$ *secure for any negligible* ϵ'*.*
3. *If* AEAD *is* ϵ-$\mathsf{IND\$\text{-}CPA}$ *secure, then* AEAD *might not be* ϵ'-$\mathsf{CTI\text{-}CPA}$ *secure for any negligible* ϵ'*.*
4. *If* AEAD *is* ϵ-$\mathsf{CTI\text{-}CPA}$ *secure, then* AEAD *might not be* ϵ'-$\mathsf{IND\$\text{-}CPA}$ *secure for any negligible* ϵ'*.*
5. *If* AEAD *is* ϵ-$\mathsf{IND\$\text{-}CCA}$ *secure, then* AEAD *might not be* ϵ'-$\mathsf{CTI\text{-}CPA}$ *secure for any negligible* ϵ'*.*
6. *If* AEAD *is* ϵ-$\mathsf{CTI\text{-}CPA}$ *secure, then* AEAD *might not be* ϵ'-$\mathsf{IND\$\text{-}CCA}$ *secure for any negligible* ϵ'*.*

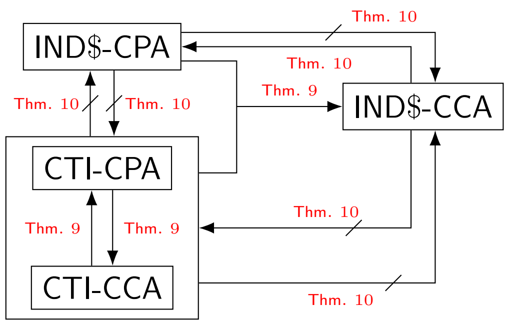


Figure 4.1: The relations between integrity and privacy for AEAD.

4.3 Generalizing AEAD Collision Resistance and Relations

4.3.1 Generalizing AEAD collision resistance

We consider the CMT-4 definition in [27] as a natural definition for *full collision resistance*. Roughly speaking, full collision resistance means that each AEAD ciphertext can only be computed by unique input. Moreover, we define variants of full collision resistance, denoted by X-CR, where $\mathsf{X} \subseteq (k, n, h, m)$[1] denotes the input portions that are unique for computing an AEAD ciphertext. In particular, the (k, n, h, m)-CR security is identical to full-CR security[2].

Additional Notation. For any $\mathsf{X} \subseteq (k, n, h, m)$, we define a class of projection functions $f_{\mathsf{X}} : \mathcal{K} \times \mathcal{N} \times \mathcal{H} \times \mathcal{M} \rightarrow \mathsf{dom}(\mathsf{X})$, where $\mathsf{dom}(\mathsf{X})$ denotes the domain of X. The function inputs a tuple (k, n, h, m) and outputs the values that X projects to. For instance, $f_k(k, n, h, m) = k$ and $f_{(k,n,h,m)}(k, n, h, m) = (k, n, h, m)$.

Definition 43. *We say an* $\mathsf{AEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$ *is* ϵ-X *collision resistant (or* ϵ-X-CR *) for* $\mathsf{X} \subseteq (k, n, h, m)$*, if the below defined advantage of any attacker* $\mathcal{A}$ *against* $\mathrm{Expr}^{\mathsf{X\text{-}CR}}_{\mathsf{AEAD}}$ *experiment in Figure 4.2 is bounded by:*

$$\mathsf{Adv}^{\mathsf{X\text{-}CR}}_{\mathsf{AEAD}}(\mathcal{A}) := \Pr[\mathrm{Expr}^{\mathsf{X\text{-}CR}}_{\mathsf{AEAD}}(\mathcal{A}) = 1] \leq \epsilon$$

In particular, we say an AEAD *is* ϵ*-full collision resistant (or* ϵ-full-CR*), if* AEAD *is* ϵ-(k, n, h, m)-CR *secure.*

Definition 44. *We say an* $\mathsf{AEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$ *has* ϵ-X *input bound ciphertext (or* ϵ-X-IBC *) for* $\mathsf{X} \subseteq (k, n, h, m)$*, if the below defined advantage of any attacker* $\mathcal{A}$ *against* $\mathrm{Expr}^{\mathsf{X\text{-}IBC}}_{\mathsf{AEAD}}$ *experiment in Figure 4.3 is bounded by:*

$$\mathsf{Adv}^{\mathsf{X\text{-}IBC}}_{\mathsf{AEAD}}(\mathcal{A}) := \Pr[\mathrm{Expr}^{\mathsf{X\text{-}IBC}}_{\mathsf{AEAD}}(\mathcal{A}) = 1] \leq \epsilon$$

```
$\mathrm{Expr}^{\mathsf{X\text{-}CR}}_{\mathsf{AEAD}}$:
$\big((k_1, n_1, h_1, m_1), (k_2, n_2, h_2, m_2)\big) \xleftarrow{\$} \mathcal{A}()$
if $\perp \in \{k_1, n_1, h_1, m_1, k_2, n_2, h_2, m_2\}$
    return $0$
if $f_{\mathsf{X}}(k_1, n_1, h_1, m_1) = f_{\mathsf{X}}(k_2, n_2, h_2, m_2)$
    return $0$
$c_1 \leftarrow \mathsf{AEAD.Enc}(k_1, n_1, h_1, m_1)$
$c_2 \leftarrow \mathsf{AEAD.Enc}(k_2, n_2, h_2, m_2)$
return $[\![c_1 = c_2]\!]$
```

Figure 4.2: X-CR security for an AEAD $=$ (AEAD.KGen, AEAD.Enc, AEAD.Dec) scheme. f_{X} is a projection function maps inputs to the subset indicated by X. In particular, the (k, n, h, m)-CR security is also named as full-CR security.

```
$\mathrm{Expr}^{\mathsf{X\text{-}IBC}}_{\mathsf{AEAD}}$:
$\big(c, (k_1, n_1, h_1, m_1), (k_2, n_2, h_2, m_2)\big) \xleftarrow{\$} \mathcal{A}()$
if $\perp \in \{k_1, n_1, h_1, m_1, k_2, n_2, h_2, m_2\}$
    return $0$
if $f_{\mathsf{X}}(k_1, n_1, h_1, m_1) = f_{\mathsf{X}}(k_2, n_2, h_2, m_2)$
    return $0$
$m_1' \leftarrow \mathsf{AEAD.Dec}(k_1, n_1, h_1, c)$
$m_2' \leftarrow \mathsf{AEAD.Dec}(k_2, n_2, h_2, c)$
return $[\![m_1 = m_1']\!]$ and $[\![m_2 = m_2']\!]$
```

Figure 4.3: X-IBC security for an AEAD $=$ (AEAD.KGen, AEAD.Enc, AEAD.Dec) scheme. f_{X} is a projection function maps inputs to the subset indicated by X.

The above X-CR and X-IBC security notions for $\mathsf{X} \in \{k, (k, n, h), (k, n, h, m)\}$ are respectively identical to the security notions CMT-l and CMTD-l for $l \in \{1, 3, 4\}$ in [27] (also see Definition 18 and Definition 19). More precisely, we have that

1. k-CR $=$ CMT-1
2. (k, n, h)-CR $=$ CMT-3
3. (k, n, h, m)-CR $=$ CMT-4
4. k-IBC $=$ CMTD-1
5. (k, n, h)-IBC $=$ CMTD-3
6. (k, n, h, m)-IBC $=$ CMTD-4

[1]Here, we slightly abuse notation and use $(\cdot)$ to denote a set. Thus, by $\mathsf{X} \subseteq (k, n, h, m)$ we mean that X is a subset of the set (k, n, h, m). For a single element set, we sometimes also omit the parenthesis and regard it as a single element. For instance, we write $k \in \mathsf{X} \Leftrightarrow (k) \subseteq \mathsf{X}$.

[2]In Theorem 13 we will show that (k, n, h, m)-CR implies all variants, which motivated our choice to abbreviate (k, n, h, m)-CR to full-CR.

Other interesting notions are the full robustness FROB and its extension eFROB, which were first defined for the randomized AEAD. In this paper, we define a generalized FROB for nonce-based AEAD.

Definition 45. *We say an* AEAD $=$ (AEAD.KGen, AEAD.Enc, AEAD.Dec) *has* ϵ-X *full robustness (or* ϵ-X-FROB *) for* X $\subseteq (k, n, h, m)$*, if the below defined advantage of any attacker* $\mathcal{A}$ *against* $\mathrm{Expr}^{\mathsf{X\text{-}FROB}}_{\mathsf{AEAD}}$ *experiment in Figure 4.4 is bounded by:*

$$\mathsf{Adv}^{\mathsf{X\text{-}FROB}}_{\mathsf{AEAD}}(\mathcal{A}) := \Pr[\mathrm{Expr}^{\mathsf{X\text{-}FROB}}_{\mathsf{AEAD}}(\mathcal{A}) = 1] \leq \epsilon$$

```
$\mathrm{Expr}^{\mathsf{X\text{-}FROB}}_{\mathsf{AEAD}}$:
$\big(c, (k_1, n_1, h_1), (k_2, n_2, h_2)\big) \xleftarrow{\$} \mathcal{A}()$
if $\perp \in \{k_1, n_1, h_1, k_2, n_2, h_2\}$
    return 0
$m_1 \leftarrow \mathsf{AEAD.Dec}(k_1, n_1, h_1, c)$
$m_2 \leftarrow \mathsf{AEAD.Dec}(k_2, n_2, h_2, c)$
if $f_{\mathsf{X}}(k_1, n_1, h_1, m_1) = f_{\mathsf{X}}(k_2, n_2, h_2, m_2)$
    return 0
return $[\![m_1 \neq \perp]\!]$ and $[\![m_2 \neq \perp]\!]$
```

Figure 4.4: X-FROB security for an AEAD $=$ (AEAD.KGen, AEAD.Enc, AEAD.Dec) scheme. f_{X} is a projection function maps inputs to the subset indicated by X.

The above (k, n)-FROB for nonce-based AEAD is defined in a similar way as the FROB definition for the randomized AEAD in [89]. The above (k, n, h, m)-FROB for nonce-based AEAD is defined in a similar way as the eFROB definition for the randomized AEAD in [97].

4.3.2 Overview of Relationship between Collision Resistance and Existing Frameworks

It turns out that this notion of collision resistance, while straightforward, is enough to cover in practice multiple notions of the literature from [10, 27, 89, 97, 135]. Informally, these notions are:

- *tidyness* - for a fixed key, is the encryption function the inverse of the decryption one? It implies that collisions over encryptions or decryptions are equivalent.
- *commitment* (CMT-l and CMTD-l for $l \in \{1, 3, 4\}$ [27]) - can we find collisions either over the encryption or the decryption, with different parts of the inputs being allowed to stay fixed based on l? In order to capture more variants in this property class that are not included in [27], in this paper we rename CMT-l to *collision resistance* (X-CR) and CMTD-l to *input bound ciphertexts* (X-IBC), where X $\subseteq (k, n, h, m)$ denotes the inputs that a AEAD scheme commits to.

- *full robustness* (FROB [89]) and *even fuller robustness* (eFROB [97]) - is any attacker able to compute a ciphertext that decrypts correctly under two distinct inputs? This notion was initially defined for randomized AEADs. In this paper, we extend the robustness notions FROB and eFROB for randomized AEADs to a unified notion X-FROB for nonce-based AEADs in Definition 45, where $\mathsf{X} \subseteq (k, n, h, m)$ denotes the degree of robustness. Moreover, we prove that X-FROB is equivalent to X-IBC in Theorem 14.
- *key committing* KC security [10] - is any attacker able to compute a ciphertext that decrypts correctly under different keys but same nonce? In this paper, we recall the KC definition in Definition 20 and show that X-FROB with $k \in \mathsf{X}$ implies KC, while the reverse does not hold, in Theorem 15.
- *multi-key collision resistance* (MKCR) [135] - is any attacker able to compute a ciphertext that decrypts correctly under multiple keys but same nonce and header? The MKCR is parameterized by the number of distinct keys $\kappa \geq 2$. In this paper, we focus on the simplified case where $\kappa = 2$. We recall the MKCR definition in Definition 21 and show that KC implies the simplified MKCR, while the reverse does not hold, in Theorem 16.
- *receiver binding* (r-BIND) [97] - is any attacker able to compute a ciphertext that can be verified under the different header and message? This notion was initially defined for a variant of compactly committing AEAD (ccAEAD), and showed how it can be instantiated for instance with an Encrypt then Mac construction[3]. Note that [97] also introduces how to transform any AEAD to ccAEAD by a "*traditionally committing encryption*" approach (ccAEAD[AEAD]). In this paper, we recall the r-BIND definition in Definition 24 and show its relations with other security notions (in this list) in Theorem 17 and Theorem 18.

We provide an overview for the full relations between the above notions in the theorem below, which is illustrated in Figure 4.5. We will unfold the formal theorems in Section 4.3.4 and give the detailed proofs in Section 4.5. While some of the relations were conjectured before ([27]), we are the first to provide the full proofs, as well as provide generalizations of some notions to enable a comparison.

Theorem 11 (Informal). *For any* AEAD *scheme, we have that*

1. X-FROB *implies* X-CR *for any* $\mathsf{X} \subseteq (k, n, h, m)$. *If* AEAD *is tidy, the reverse also holds. See Theorem 12.*
2. X-FROB/X-CR/X-IBC *resp. implies* X′-FROB/X′-CR/ X′-IBC *for any* $\mathsf{X}' \subseteq \mathsf{X} \subseteq (k, n, h, m)$. *See Theorem 13.*
3. X-FROB *and* X-IBC *are equivalent for any* $\mathsf{X} \subseteq (k, n, h, m)$. *See Theorem 14.*

[3] [97] proposes to use HMAC-SHA256 to instantiate a keyed random oracle, which is technically false without an additional assumption, as $HMAC(k, m) = HMAC(H(k), m)$ whenever k is bigger than 256 bits.

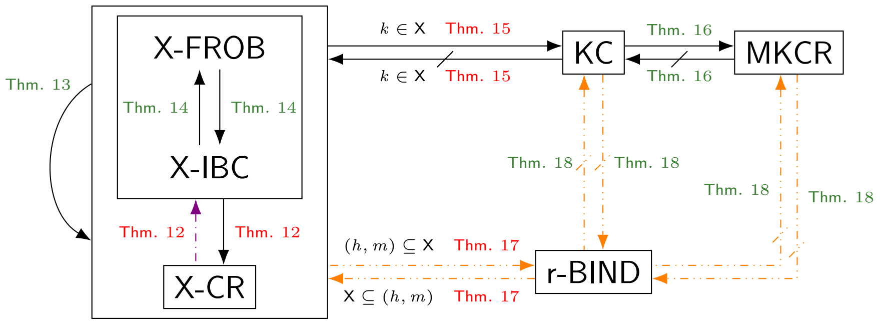


Figure 4.5: The relation between collision related properties for AEAD with key space $\mathcal{K}$. The black arrow $\rightarrow$ indicates the general implication. The purple dash-dotted arrow $\dashrightarrow$ indicates the implication for tidy AEAD. The orange dash-dot-dotted arrow $\dashrightarrow$ indicates the implication for ccAEAD[AEAD]. The X in the figure is a subset of (k, n, h, m), i.e., X $\subseteq (k, n, h, m)$. The theorems highlighted with red color are claimed or proven in other papers. The theorems highlighted with green color are part of our contributions.

4. *k-FROB implies KC but not in reverse. See Theorem 15.*

5. *KC implies MKCR but not in reverse. See Theorem 16.*

6. *(h, m)-FROB of AEAD implies r-BIND of ccAEAD[AEAD]. r-BIND of ccAEAD[AEAD] implies X-FROB of AEAD for any X $\subseteq (h, m)$. See Theorem 17.*

7. *Neither KC nor MKCR of AEAD implies r-BIND of ccAEAD[AEAD]. The reverse is same. See Theorem 18.*

Note that Theorem 12 was proven in [27]. Theorem 15 and Theorem 17 were respectively claimed in [27] and [97] without giving any proofs. Proofs for Theorem 13, Theorem 14, Theorem 16 and Theorem 18 are part of our contribution. Recall that we have (k, n, h, m)-CR = full-CR in this figure. This theorem indicates that the full collision resistance implies all other existing notions in this figure under the tidyness assumption, which is in fact met by all classical constructions.

4.3.3 Collision attacks on deployed AEADs

In general, any kind of collision between two ciphertexts can lead to a security issue, and we will advocate that general use AEADs should be fully resistant to collisions. However, many popular deployed AEADs do not meet the full collision resistance, as shown in Table 4.1. Below, we recall the known attacks against various kinds of collision resistances of different AEAD schemes in the literature.

1. r-BIND: [97] shows a generic attack against any EtM construction with unrelated keys by finding the second key that causes collision by . This attack also applies to real-world modes using Carter-Wegman MACs, e.g., GCM and ChaCha20-Poly1305. [82] shows a concrete attack against AES-GCM and OCB by finding the nonce that causes collision and sketches an faster attack by doing birthday attack on keys. Moreover, at the hand of

Concrete AEAD	Integrity and Privacy	Full Collision Resistance
XSalsa20-Poly1305	●	✗ [10]
AES-GCM	✓ [115, 139]	✗ [82]
ChaCha20-Poly1305	✓ [159]	✗ [10]
OCB3	✓ [49, 132]	✗ [10]
EtM (unrelated keys)	✓ [163]	✗ [97][3]
AES-CCM	✓ [91, 120]	●
AES-EAX	✓ [35, 141]	●
EtM (related keys)	✓ [163]	✓ [97]
CAU-C4	✓ [27]	✓ [27]
AES-GCM-SIV	✓ [98, 116]	✗ [10]
CAU-SIV-C4	✓ [27]	✓ [27]

✓ : proven in the cited work(s). ● : we conjecture that this holds, but do not know of a proof.
✗ : does not hold, with reference or explanation of counterexample.

Table 4.1: AEADs (in)-security guarantees: *Integrity and Privacy* refers to IND\$-CPA and CTI-CPA. *Full Collision Resistance* refers to Definition 38.

a corollary of Theorem 1 in [164], [82] claims that this attack also applies to any so-called *rate-1* AEAD, that is, "one blockcipher call per block of message" [82]. This potentially indicates the vulnerability of AES-GCM-SIV and ChaCha20-Poly1305 and any EtM constructions.

2. KC: [10] extends the known attack in [82] against AES-GCM to new proof-of-concept attacks against several commonly used AEAD, including AES-GCM, ChaCha20-Poly1305, AES-GCM-SIV, and OCB3. This attack shows how to create ciphertext collision on two distinct keys. Then, [10] also shows that their new attacks also make impacts in some real-world scenarios, such as the binary polyglots setting.
3. MKCR: [135] shows a novel partitioning oracle attack that feasibly breaks the MKCR security with parameter $\kappa \geq 2$ of widely used AEAD schemes, including AES-GCM, AES-GCM-SIV, ChaCha20-Poly1305, and XSalsa20-Poly1305.
4. X-CR and X-IBC: [27] finds that all above attacks also break the k-CR and -IBC security of respective AEAD schemes. Thus, AES-GCM, AES-GCM-SIV, XSalsa20-Poly1305, and ChaCha20-Poly1305 and OCB are all k-CR insecure, i.e., CMT-1-insecure in [27].

4.3.4 Theorems for Relationships between Collision Resistance and Existing Frameworks

We first recall the conclusion in [27] that X-IBC implies X-CR. Moreover, these two notions are equivalent if the AEAD is tidy.

Theorem 12 ([27, Appendix A]). *Let* $\mathsf{AEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$ *be an authenticated encryption with associated data scheme. Then, it holds that:*

1. *If* AEAD *is* ϵ-X-IBC *secure, then* AEAD *is also* ϵ-X-CR *secure.*[4]
2. *If* AEAD *is* ϵ-X-CR *secure and tidy, then* AEAD *is also* ϵ-X-IBC *secure.*[5]
3. *If* AEAD *is* ϵ-(k, n, h, m)-IBC *(resp.* CR*) secure, then* AEAD *is* ϵ-(k, n, h)-IBC *(resp.* CR*) secure, and vice versa.*
4. *If* AEAD *is* ϵ-(k, n, h)-IBC *(resp.* CR*) secure, then* AEAD *is* ϵ-k-IBC *(resp.* CR*) secure.*

For X-CR, X-IBC, and X-FROB, we can have following trivial conclusion that generalizes Theorem 12.

Theorem 13. *Let* $\mathsf{AEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$ *be an authenticated encryption with associated data. If* AEAD *is* ϵ-X-CR *(resp.* X-IBC *or* X-FROB*), then it is also* ϵ-X'-CR *(resp.* X'-IBC *or* X'-FROB*) for any* $\mathsf{X}' \subseteq \mathsf{X}$.

Notably, we find that X-FROB and X-IBC are identical.

Theorem 14. *Let* $\mathsf{AEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$ *be an authenticated encryption with associated data. If* AEAD *is* ϵ-X-FROB *secure for* $\mathsf{X} \subseteq (k, n, h, m)$*, then* AEAD *is also* ϵ-X-IBC *secure, and vice versa.*

Interestingly, [27] has the following observations without giving formal proof. We hereby provide the proof below.

Theorem 15. *Let* $\mathsf{AEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$ *be an authenticated encryption with associated data with key space* $\mathcal{K}$*. Then, it holds that:*

1. *If* AEAD *is* ϵ-k-FROB *secure, then* AEAD *is* ϵ-KC *secure.*
2. *If* AEAD *is* ϵ-KC *secure, then* AEAD *might not be* ϵ'-k-FROB *secure for any negligible* ϵ'.

Moreover, we also find that KC security implies MKCR security, while the reverse direction does not hold.

Theorem 16. *Let* $\mathsf{AEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$ *be an authenticated encryption with associated data with key space* $\mathcal{K}$*. Then, it holds that:*

1. *If* AEAD *is* ϵ-KC *secure, then* AEAD *is* ϵ-MKCR *secure.*
2. *If* AEAD *is* ϵ-MKCR *secure, then* AEAD *might not be* ϵ'-KC *secure for any negligible* ϵ'.

It is interesting to observe that any $\mathsf{ccAEAD}[\mathsf{AEAD}]$ is s-BIND secure. Moreover, it is stated in [97] that the r-BIND security for the randomized "traditionally committing encryption" AEAD can be implied by eFROB but cannot be implied by the standard FROB security without including headers. In terms of our syntax, we have similar conclusions.

[4]We stress that although [27, Appendix A] only proves the theorem for $\mathsf{X} \in \{k, (k, n, h), (k, n, h, m)\}$, the proof for all other $\mathsf{X} \subseteq (k, n, h, m)$ can be easily given in a similar way.

[5]Similar to above, the proof for all other $\mathsf{X} \subseteq (k, n, h, m)$ can be easily given in a similar way as in [27, Appendix A].

Theorem 17. *Let* $\mathsf{AEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$ *be an authenticated encryption with associated data. Let* $\mathsf{ccAEAD}[\mathsf{AEAD}]$ *denote the compactly committing* AEAD *derived from* AEAD *using "traditionally committing encryption" approach. Then, it holds that:*

1. $\mathsf{ccAEAD}[\mathsf{AEAD}]$ *is* 0-s-BIND *secure.*
2. *If* AEAD *is* ϵ-X-FROB *secure for* $(h, m) \subseteq \mathsf{X}$*, then* $\mathsf{ccAEAD}[\mathsf{AEAD}]$ *is also* ϵ-r-BIND *secure.*
3. *If* $\mathsf{ccAEAD}[\mathsf{AEAD}]$ *is* ϵ-r-BIND *secure, then* $\mathsf{ccAEAD}[\mathsf{AEAD}]$ *is also* ϵ-X-FROB *secure for* $\mathsf{X} \subseteq (h, m)$.

Moreover, we also observe that neither KC nor MKCR security of AEAD implies the r-BIND security of $\mathsf{ccAEAD}[\mathsf{AEAD}]$. Conversely, the r-BIND security of $\mathsf{ccAEAD}[\mathsf{AEAD}]$ does not imply KC or MKCR security.

Theorem 18. *Let* $\mathsf{AEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$ *be an authenticated encryption with associated data scheme. Let* $\mathsf{ccAEAD}[\mathsf{AEAD}]$ *denote the compactly committing* AEAD *derived from* AEAD *using "traditionally committing encryption" approach. Then, it holds that:*

1. *If* AEAD *is* ϵ-KC *secure, then* $\mathsf{ccAEAD}[\mathsf{AEAD}]$ *might not be* ϵ'-r-BIND *secure for any negligible* ϵ'.
2. *If* AEAD *is* ϵ-MKCR *secure, then* $\mathsf{ccAEAD}[\mathsf{AEAD}]$ *might not be* ϵ'-r-BIND *secure for any negligible* ϵ'.
3. *If* $\mathsf{ccAEAD}[\mathsf{AEAD}]$ *is* ϵ-r-BIND *secure, then* $\mathsf{ccAEAD}[\mathsf{AEAD}]$ *might not be* ϵ'-KC *secure for any negligible* ϵ'.
4. *If* $\mathsf{ccAEAD}[\mathsf{AEAD}]$ *is* ϵ-r-BIND *secure, then* $\mathsf{ccAEAD}[\mathsf{AEAD}]$ *might not be* (κ, ϵ')-MKCR *secure for any* $\kappa \geq 2$ *and any negligible* ϵ'.

4.4 Conclusions

We recalled the standard privacy and integrity of AEAD and provided missing proofs for some widely acknowledged relations between them. In addition, we defined a novel (full) collision resistance and generalize it with various variants. We proved the relations between (full) collision resistance and several related notions in the literature, e.g., robustness, key committing, and receiver binding, completing the picture in this domain. In particular, our novel full collision resistance implies all these related notions, which makes it to be the "strongest" security notion in this domain.

4.5 Full Proofs

4.5.1 Proof of Theorem 10

Proof. We prove each of these statements in turn.

1. Statement 1: This statement can be proven by a trivial reduction. If there exists an attacker $\mathcal{A}$ that breaks $\mathsf{IND\$\text{-}CPA}$ security of AEAD, then we can construct an attacker $\mathcal{B}$ that breaks $\mathsf{IND\$\text{-}CCA}$ security of AEAD by invoking AEAD. $\mathcal{B}$ simply invokes $\mathcal{A}$, forwards all $\mathcal{A}$'s queries to its challenger and returns the responses to $\mathcal{A}$, and finally outputs $\mathcal{A}$'s decision. We observe that $\mathcal{B}$ wins if and only if $\mathcal{A}$ wins, which concludes the proof.
2. Statement 2: See [23, Lemma 2].
3. Statement 3: We prove this statement by counter example. Let $\mathsf{AEAD}_1 = (\mathsf{AEAD}_1.\mathsf{KGen}, \mathsf{AEAD}_1.\mathsf{Enc}, \mathsf{AEAD}_1.\mathsf{Dec})$ and $\mathsf{AEAD}_2 = (\mathsf{AEAD}_2.\mathsf{KGen}, \mathsf{AEAD}_2.\mathsf{Enc}, \mathsf{AEAD}_2.\mathsf{Dec})$ denote two independent ϵ-$\mathsf{IND\$\text{-}CPA}$ secure authenticated encryption with associated data schemes with the same message space $\mathcal{M}$. We then construct $\mathsf{AEAD}' = (\mathsf{AEAD}'.\mathsf{KGen}, \mathsf{AEAD}'.\mathsf{Enc}, \mathsf{AEAD}'.\mathsf{Dec})$ from AEAD_1 and AEAD_2 as follows:

   - $\mathsf{AEAD}'.\mathsf{KGen}()$: runs $k_1 \xleftarrow{\$} \mathsf{AEAD}_1.\mathsf{KGen}()$ and $k_2 \xleftarrow{\$} \mathsf{AEAD}_2.\mathsf{KGen}()$ followed by outputting $k' := k_1 \parallel k_2$.
   - $\mathsf{AEAD}'.\mathsf{Enc}(k', n, h, m)$: first parses $k_1 \parallel k_2 \leftarrow k'$ and then runs $c_1 \leftarrow \mathsf{AEAD}_1.\mathsf{Enc}(k_1, n, h, m)$ and $c_2 \leftarrow \mathsf{AEAD}_2.\mathsf{Enc}(k_2, n, h, m)$, followed by outputting $c' := c_1 \parallel c_2$.
   - $\mathsf{AEAD}'.\mathsf{Dec}(k', n, h, c')$: parses $k_1 \parallel k_2 \leftarrow k'$ and $c_1 \parallel c_2 \leftarrow c'$, followed by outputting $\mathsf{AEAD}.\mathsf{Dec}(k_1, n, h, c_1)$.

   It is straightforward to prove that AEAD' is 2ϵ-$\mathsf{IND\$\text{-}CPA}$ secure by reduction. If there exists an attacker $\mathcal{A}$ that breaks the $\mathsf{IND\$\text{-}CPA}$ security of AEAD', then we can construct an attacker $\mathcal{B}$ that breaks the $\mathsf{IND\$\text{-}CPA}$ security of AEAD_1 or AEAD_2.

   However, AEAD' is not $\mathsf{CTI\text{-}CPA}$ secure. An attacker can queries $\mathcal{O}_{\mathsf{Enc}}(n, h, m)$ for any n, h, m for a ciphertext $c' = c_1 \parallel c_2$ such that $c_1 \neq c_2$, followed by outputting $c'' = c_1 \parallel c_1$. It is easy to see that $c'' \notin \mathcal{L}_{\mathcal{O}_{\mathsf{Enc}}}$ since $c' \neq c''$ and $\mathsf{AEAD}'.\mathsf{Dec}(k, n, h, c'') = \mathsf{AEAD}'.\mathsf{Dec}(k, n, h, c') = m \neq \bot$. Thus, this attacker always win the $\mathsf{CTI\text{-}CPA}$ experiment.
4. Statement 4: We prove this statement by counter example. Let $\mathsf{AEAD} = (\mathsf{AEAD}.\mathsf{KGen}, \mathsf{AEAD}.\mathsf{Enc}, \mathsf{AEAD}.\mathsf{Dec})$ denote an ϵ-$\mathsf{CTI\text{-}CPA}$ secure authenticated encryption with associated data scheme with the message space $\mathcal{M}$. We then construct $\mathsf{AEAD}' = (\mathsf{AEAD}'.\mathsf{KGen}, \mathsf{AEAD}'.\mathsf{Enc}, \mathsf{AEAD}'.\mathsf{Dec})$ from AEAD as follows:

   - $\mathsf{AEAD}'.\mathsf{KGen}()$: is identical to $k \xleftarrow{\$} \mathsf{AEAD}.\mathsf{KGen}()$.
   - $\mathsf{AEAD}'.\mathsf{Enc}(k, n, h, m)$: runs $c \leftarrow \mathsf{AEAD}.\mathsf{Enc}(k, n, h, m)$ followed by outputting $c' := c \parallel c$.

- $\mathsf{AEAD}'.\mathsf{Dec}(k, n, h, c')$: first parses $c_1 \parallel c_2 \leftarrow c'$ and outputs $\perp$ if $c_1 \neq c_2$. Otherwise, outputs $\mathsf{AEAD}.\mathsf{Dec}(k, n, h, c_1)$.

It is straightforward to prove that AEAD' is ϵ-CTI-CPA secure by reduction. If there exists an attacker $\mathcal{A}$ that breaks the CTI-CPA security of AEAD', then we can construct an attacker $\mathcal{B}$ that breaks the CTI-CPA security of AEAD.

However, AEAD' is not $\mathsf{IND\$}$-$\mathsf{CPA}$ secure since an attacker can easily distinguish any ciphertext $c' = c'_1 \parallel c'_2$ of AEAD' from a random string of the same length by checking whether $c'_1 = c'_2$.

5. Statement 5: We prove this statement by counter example. Let $\mathsf{AEAD} = (\mathsf{AEAD}.\mathsf{KGen}, \mathsf{AEAD}.\mathsf{Enc}, \mathsf{AEAD}.\mathsf{Dec})$ denote an ϵ-$\mathsf{IND\$}$-$\mathsf{CCA}$ secure authenticated encryption with associated data scheme with message space $\mathcal{M}$, nonce space $\mathcal{N}$, and header space $\mathcal{H}$, and ciphertext space $\mathcal{CT}$. In particular, let $\tilde{n} \in \mathcal{N}$ and $\tilde{h} \in \mathcal{H}$ denote two arbitrary strings in the respective domains. We then construct $\mathsf{AEAD}' = (\mathsf{AEAD}'.\mathsf{KGen}, \mathsf{AEAD}'.\mathsf{Enc}, \mathsf{AEAD}'.\mathsf{Dec})$ from AEAD as follows:

   - $\mathsf{AEAD}'.\mathsf{KGen}()$: runs $k \xleftarrow{\$} \mathsf{AEAD}_1.\mathsf{KGen}()$ and samples $r \xleftarrow{\$} \mathcal{M}$ followed by outputting $k' := k \parallel r$.
   - $\mathsf{AEAD}'.\mathsf{Enc}(k', n, h, m)$: first parses $k \parallel r \leftarrow k'$ and then outputs c as a string of $l(|m|)$ zero bits if $r = m$, $n = \tilde{n}$ and $h = \tilde{h}$. Otherwise, outputs $c \leftarrow \mathsf{AEAD}.\mathsf{Enc}(k, n, h, m)$.
   - $\mathsf{AEAD}'.\mathsf{Dec}(k', n, h, c)$: first parses $k \parallel r \leftarrow k'$, followed by outputting r if c is a string of $l(|m|)$ zero bits, $n = \tilde{n}$ and $h = \tilde{h}$. Otherwise, outputs $\mathsf{AEAD}.\mathsf{Dec}(k, n, h, c)$.

   It is straightforward to prove that AEAD' is $(\epsilon + \frac{q}{|\mathcal{M}|})$-$\mathsf{IND\$}$-$\mathsf{CCA}$ secure by reduction, where q denotes the number of queries that $\mathcal{A}$ can make in polynomial time. If there exists an attacker $\mathcal{A}$ that breaks the $\mathsf{IND\$}$-$\mathsf{CCA}$ security of AEAD', then we can construct an attacker $\mathcal{B}$ that breaks the $\mathsf{IND\$}$-$\mathsf{CCA}$ security of AEAD.

   However, AEAD' is not CTI-CPA secure since a string of $l(|m|)$ zero bits is always a ciphertext, which can be decrypted to a message $r \in \mathcal{M}$ for any $k \in \mathcal{K}$, $n = \tilde{n}$, and $h \in \tilde{h}$.

6. Statement 6: This statement is implied by Statements 1 and 4.

□

4.5.2 Proof of Theorem 13

Proof. This theorem can by proven by three trivial reductions. Here, we only give the trivial reductions for CR security, the reductions for IBC and FROB can be given in a similar way.

Let $\mathcal{A}$ denotes an attacker that breaks X'-CR security of AEAD. We define an attacker $\mathcal{B}$ that invokes $\mathcal{A}$ and outputs same as $\mathcal{A}$. Note that f_{X} is a projection function that maps inputs to the subset indicated by X. By $\mathsf{X}' \subseteq \mathsf{X}$, we have that $f_{\mathsf{X}'}(k, n, h, m)$ is a subset of $f_{\mathsf{X}}(k, n, h, m)$. This indicates that $f_{\mathsf{X}'}(k_1, n_1, h_1, m_1) \neq f_{\mathsf{X}'}(k_2, n_2, h_2, m_2) \implies f_{\mathsf{X}}(k_1, n_1, h_1, m_1) \neq f_{\mathsf{X}}(k_2, n_2, h_2, m_2)$. Thus, $\mathcal{B}$ wins X-CR security experiment of AEAD whenever $\mathcal{A}$ wins. □

4.5.3 Proof of Theorem 14

Proof. Suppose that $\mathcal{A}$ can break the ϵ-X-IBC security of AEAD. Then we can construct an attacker $\mathcal{B}$ that breaks the ϵ-X-FROB security of AEAD. When $\mathcal{A}$ outputs $\Big(c, (k_1, n_1, h_1, m_1), (k_2, n_2, h_2, m_2)\Big)$, $\mathcal{B}$ simply outputs $\Big(c, (k_1, n_1, h_1), (k_2, n_2, h_2)\Big)$. If $\mathcal{A}$ wins, then it must hold that

1. $\perp \notin \{k_1, n_1, h_1, m_1, k_2, n_2, h_2, m_2\}$. In particular, this implies that $\perp \notin \{k_1, n_1, h_1, k_2, n_2, h_2\}$, $m_1 \neq \perp$, and $m_2 \neq \perp$
2. $f_{\mathsf{X}}(k_1, n_1, h_1, m_1) = f_{\mathsf{X}}(k_2, n_2, h_2, m_2)$
3. $\mathsf{AEAD.Dec}(k_1, n_1, h_1, c) = m_1$. In particular, this implies that $\mathsf{AEAD.Dec}(k_1, n_1, h_1, c) \neq \perp$
4. $\mathsf{AEAD.Dec}(k_2, n_2, h_2, c) = m_2$. In particular, this implies that $\mathsf{AEAD.Dec}(k_2, n_2, h_2, c) \neq \perp$

This implies that $\mathcal{B}$ also wins.

In reverse, suppose that $\mathcal{A}$ can break the ϵ-X-FROB security of AEAD. Then we can construct an attacker $\mathcal{B}$ that breaks the ϵ-X-IBC security of AEAD. When $\mathcal{A}$ outputs $\Big(c, (k_1, n_1, h_1), (k_2, n_2, h_2)\Big)$, $\mathcal{B}$ simply computes $m_1 \leftarrow \mathsf{AEAD.Dec}(k_1, n_1, h_1, c)$ and $m_2 \leftarrow \mathsf{AEAD.Dec}(k_2, n_2, h_2, c)$ and outputs

$$\Big(c, (k_1, n_1, h_1, m_1), (k_2, n_2, h_2, m_2)\Big).$$

If $\mathcal{A}$ wins, then it must hold that

1. $\perp \notin \{k_1, n_1, h_1, k_2, n_2, h_2\}$
2. $f_{\mathsf{X}}(k_1, n_1, h_1, m_1) = f_{\mathsf{X}}(k_2, n_2, h_2, m_2)$
3. $m_1 \neq \perp$
4. $m_2 \neq \perp$

This implies that $\perp \notin \{k_1, n_1, h_1, m_1, k_2, n_2, h_2, m_2\}$. Then, $\mathcal{B}$ always wins, which concludes the proof. □

4.5.4 Proof of Theorem 15

Proof. We prove each of these statements in turn.

1. Statement 1: Suppose that $\mathcal{A}$ can break the (q, ϵ)-KC security of AEAD for any $q \geq 2$. Then we can construct an attacker $\mathcal{B}$ that breaks the ϵ-k-FROB security of AEAD. $\mathcal{B}$ initializes an empty list $\mathcal{L}$ and simulates experiment $\mathrm{Expr}^{\mathsf{KC}}_{\mathsf{AEAD},q}$ to $\mathcal{A}$. When $\mathcal{A}$ terminates, $\mathcal{B}$ checks there exist entries $(k_1, n_1, h_1, m_1, c_1), (k_2, n_2, h_2, m_2, c_2) \in \mathcal{L}$ such that

   (a) $k_1 \neq k_2$
   (b) $c_1 = c_2 \neq \bot$
   (c) $m_1 \neq \bot$
   (d) $m_2 \neq \bot$

   If such entries do not exist, $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ outputs $\Big(c_1, (k_1, n_1, h_1), (k_2, n_2, h_2)\Big)$. If $\mathcal{A}$ wins, then such entries must exist, which further implies that $\mathcal{B}$ wins. The proof is concluded.

2. Statement 2: We prove this statement by giving a counter-example. Let $\mathsf{SKE} = (\mathsf{AEAD}'.\mathsf{KGen}, \mathsf{AEAD}'.\mathsf{Enc}, \mathsf{AEAD}'.\mathsf{Dec})$ be an one-time pad with spaces $\mathcal{K}' = \mathcal{M}' = \{0,1\}^t$ for some $t > 0$. Let $\mathsf{cPRF} : \mathcal{K}' \times \mathcal{H} \to \mathcal{K}' \times \mathcal{T}$ denote a $\epsilon^{\mathsf{bind}}_{\mathsf{cPRF}}$-$\mathsf{bind}$ secure function for some spaces $\mathcal{H}$ and $\mathcal{T}$. We then construct an $\mathsf{AEAD} = (\mathsf{AEAD}.\mathsf{KGen}, \mathsf{AEAD}.\mathsf{Enc}, \mathsf{AEAD}.\mathsf{Dec})$ with spaces $\mathcal{K} = \mathcal{N} = \mathcal{K}' = \mathcal{M}' = \{0,1\}^t$ from SKE and cPRF as follows:

   (a) $\mathsf{AEAD}.\mathsf{KGen}()$: is identical to $\mathsf{AEAD}'.\mathsf{KGen}()$
   (b) $\mathsf{AEAD}.\mathsf{Enc}(k, n, h, m)$: computes $(y, y') \leftarrow \mathsf{cPRF}(k \oplus n, h)$ and $c' \leftarrow \mathsf{AEAD}'.\mathsf{Enc}(y, m \oplus n)$, followed by outputting $c = (c', y')$.
   (c) $\mathsf{AEAD}.\mathsf{Dec}(k, n, h, c)$: parses $(c', y') \leftarrow c$ and verifies whether $(y, y') = \mathsf{cPRF}(k \oplus n, h)$ for some y. If the verification fails, then outputs $\bot$. Otherwise, outputs $\mathsf{AEAD}'.\mathsf{Dec}(y, c') \oplus n$.

   We first prove that AEAD is $\epsilon^{\mathsf{KC}}_{\mathsf{AEAD}}$-$\mathsf{KC}$ secure for any $q \geq 2$, where $\epsilon^{\mathsf{KC}}_{\mathsf{AEAD}} \leq \epsilon^{\mathsf{bind}}_{\mathsf{cPRF}}$. Suppose an attacker $\mathcal{A}$ that breaks the KC security of AEAD, then we can construct an attacker $\mathcal{B}$ that breaks bind security of the underlying cPRF. $\mathcal{B}$ simply invokes $\mathcal{A}$ and honestly simulates the KC experiment to $\mathcal{A}$. If $\mathcal{A}$ wins, then there must exist $(k_1, n_1, h_1, m_1, c_1), (k_2, n_2, h_2, m_2, c_2) \in \mathcal{L}$ such that

   (a) $k_1 \neq k_2$
   (b) $n_1 = n_2$
   (c) $c_1 = c_2 \neq \bot$
   (d) $m_1 \neq \bot$ and $m_2 \neq \bot$

This implies that $(k_1 \oplus n_1, h_1) \neq (k_2 \oplus n_2, h_2)$. Moreover, for $c_1 = (c_1', y_1')$ and $c_2 = (c_2', y_2')$, the condition $c_1 = c_2$ implies that $y_1' = y_2'$. Then, $\mathcal{B}$ can simply checks all elements in the list $\mathcal{L}$ for such $k_1, k_2, n_1, n_2, y_1', y_2'$ and outputs $(k_1 \oplus n_1, y_1', k_2 \oplus n_2, y_2')$. After that, $\mathcal{B}$ wins whenever $\mathcal{A}$ wins.

We then prove that AEAD is not ϵ'-k-FROB for any negligible ϵ'. An attacker $\mathcal{A}$ can simply execute following steps:

(a) samples $k_1, k_2 \xleftarrow{\$} \mathcal{K}$ such that $k_1 \neq k_2$, $n_1 \xleftarrow{\$} \mathcal{N}$, $h_1 = h_2 \xleftarrow{\$} \mathcal{H}$

(b) computes $n_2 = k_1 \oplus k_2 \oplus n_1$

(c) picks any message $m \in \mathcal{M}$ and computes $c \leftarrow \mathsf{AEAD}(k_1, n_1, h_1, m)$

(d) outputs $\big(c, (k_1, n_1, h_1), (k_2, n_2, h_2)\big)$

It is straightforward that for $k_1 \neq k_2$, $m_1 = m \neq \perp$, and $m_2 = m_1 \oplus n_1 \oplus n_2 \neq \perp$ for $m_1 \leftarrow \mathsf{AEAD.Dec}(k_1, n_1, h_1, c)$ and $m_2 \leftarrow \mathsf{AEAD.Dec}(k_2, n_2, h_2, c)$. Thus, $\mathcal{A}$ always wins.

□

4.5.5 Proof of Theorem 16

Proof. We prove each of these statements in turn.

1. Statement 1: We prove this statement by reduction. If there exists an attacker $\mathcal{A}$ that breaks the MKCR security of AEAD with probability ϵ, then we can construct an attacker $\mathcal{B}$ that breaks the KC security of AEAD also with probability ϵ. The attacker $\mathcal{B}$ simply invokes $\mathcal{A}$. When $\mathcal{A}$ outputs $(\mathcal{K}^\star, n^\star, h^\star, c^\star)$, $\mathcal{B}$ picks two arbitrary $k_1, k_2 \in \mathcal{K}^\star$ with $k_1 \neq k_2$. Then, $\mathcal{B}$ queries $\mathcal{O}_{\mathsf{Dec}}$ oracle twice, respectively with inputs $(k_1, n^\star, h^\star, c^\star)$ and $(k_2, n^\star, h^\star, c^\star)$. If $\mathcal{A}$ wins, then it must hold that $\perp \neq m_1 = \mathsf{AEAD.Dec}(k_1, n^\star, h^\star, c^\star)$ and $\perp \neq m_2 = \mathsf{AEAD.Dec}(k_2, n^\star, h^\star, c^\star)$. Thus, $\mathcal{B}$ always wins.

2. Statement 2: We prove this statement by giving a counter-example. Let $\mathsf{SKE} = (\mathsf{AEAD'.KGen}, \mathsf{AEAD'.Enc}, \mathsf{AEAD'.Dec})$ be an one-time pad with spaces $\mathcal{K}' = \mathcal{M}' = \{0,1\}^t$ for some $t > 0$. Let $\mathsf{cPRF} : \mathcal{K}' \times \mathcal{N} \to \mathcal{K}' \times \mathcal{T}$ denote a $\epsilon^{\mathsf{bind}}_{\mathsf{cPRF}}$-$\mathsf{cPRF}$ secure function for some spaces $\mathcal{N}$ and $\mathcal{T}$. We then construct an $\mathsf{AEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$ with spaces $\mathcal{K} = h$ from SKE and F as follows:

   (a) $\mathsf{AEAD.KGen}()$: is identical to $\mathsf{AEAD'.KGen}()$

   (b) $\mathsf{AEAD.Enc}(k, n, h, m)$: computes $(y, y') \leftarrow \mathsf{cPRF}(k \oplus h, n)$ and $c' \leftarrow \mathsf{AEAD'.Enc}(y, m)$, followed by outputting $c = (c', y')$.

   (c) $\mathsf{AEAD.Dec}(k, n, h, c)$: parses $(c', y') \leftarrow c$ and verifies whether $(y, y') = \mathsf{cPRF}(k \oplus h, n)$ for some y. If the verification fails, then outputs $\perp$. Otherwise, outputs $\mathsf{AEAD'.Dec}(y, c')$.

We first prove that AEAD is ϵ-MKCR secure, where $\epsilon = \epsilon^{\mathsf{bind}}_{\mathsf{cPRF}}$. Suppose an attacker $\mathcal{A}$ that breaks the MKCR security of AEAD, we then construct an attacker $\mathcal{B}$ that breaks the bind security of the underlying cPRF. $\mathcal{B}$ simply invokes $\mathcal{A}$ and honestly simulates the MKCR experiment for $\kappa = 2$. The attacker $\mathcal{A}$ wins if it can output $(\mathcal{K}^\star, n^\star, h^\star, c^\star)$ such that

(a) $|\mathcal{K}^\star| \geq 2$, and

(b) $\mathsf{AEAD.Dec}(k, n^\star, h^\star, c^\star) \neq \perp$ for all $k \in \mathcal{K}^\star$

Then, $\mathcal{B}$ simply picks any $k_1 \neq k_2$ from space $\mathcal{K}^\star$. It must hold that $(k_1 \oplus h^\star, n^\star) \neq (k_2 \oplus h^\star, n^\star)$. The decryption $\mathsf{AEAD.Dec}(k_1, n^\star, h^\star, c^\star) \neq \perp$ and $\mathsf{AEAD.Dec}(k_2, n^\star, h^\star, c^\star) \neq \perp$ indicates that the verification inside the decryption never fails. This means, for $(y_1, y_1') \leftarrow \mathsf{cPRF}(k_1 \oplus h^\star, n^\star)$ and $(y_2, y_2') \leftarrow \mathsf{cPRF}(k_2 \oplus h^\star, n^\star)$ it must hold that $y_1' = y_2'$. Thus, $\mathcal{B}$ always wins if it outputs $(k_1 \oplus h^\star, n^\star, k_2 \oplus h^\star, n^\star)$.

Then, we prove that AEAD is not ϵ'-KC secure for any non-negligible ϵ'. An attacker $\mathcal{A}$ can simply pick arbitrary $(k_1, n_1, h_1, m) \in \mathcal{K} \times \mathcal{N} \times \mathcal{H} \times \mathcal{M}$ and invokes $\mathcal{O}_{\mathsf{Enc}}$ oracle with the input (k_1, n_1, h_1, m) for a ciphertext c_1. Then, $\mathcal{A}$ invokes $\mathcal{O}_{\mathsf{Dec}}$ oracle with input (k_2, n_2, h_2, c_2) for m_2 such that $k_1 \neq k_2$, $k_2 \oplus h_2 = k_1 \oplus h_1$, $n_1 = n_2$, and $c_1 = c_2$. It is straightforward that $m_1 = m_2 \neq \perp$ and $\mathcal{A}$ always wins, which concludes the proof.

□

4.5.6 Proof of Theorem 17

Proof. We prove each of these statements in turn.

1. Statement 1: For any (k, n, h, c) output by $\mathcal{A}$ and $m \leftarrow \mathsf{AEAD.Dec}(k, n, h, c)$, $\mathcal{A}$ can win only when

   (a) $m \neq \perp$, and

   (b) $m \neq \mathsf{AEAD.Dec}(k, n, h, c)$.

   The second condition contracts to the fact that $m \leftarrow \mathsf{AEAD.Dec}(k, n, h, c)$. Thus, $\mathcal{A}$ always loses.

2. Statement 2: We prove this claim by reduction. If there exists an attacker $\mathcal{A}$ that breaks the ϵ-r-BIND security of ccAEAD[AEAD], then we can construct an attacker $\mathcal{B}$ that breaks the ϵ'-X-FROB security of AEAD for $(h, m) \subseteq \mathsf{X}$ and $\epsilon' = \epsilon$. When $\mathcal{A}$ outputs $\Big(c, (k_1, n_1, h_1, m_1), (k_2, n_2, h_2, m_2)\Big)$, $\mathcal{A}$ wins if

   (a) $\perp \notin \{k_1, n_1, h_1, m_1, k_2, n_2, h_2, m_2\}$

   (b) $(h_1, m_1) \neq (h_2, m_2)$

   (c) $m_1 = \mathsf{AEAD.Dec}(k_1, n_1, h_1, c)$

(d) $m_2 = \mathsf{AEAD.Dec}(k_2, n_2, h_2, c)$

This in particular indicates that

(a) $\perp \notin \{k_1, n_1, h_1, k_2, n_2, h_2\}$,

(b) $m_1 = \mathsf{AEAD.Dec}(k_1, n_1, h_1, c)$

(c) $m_2 = \mathsf{AEAD.Dec}(k_2, n_2, h_2, c)$

(d) $f_{\mathsf{X}}(k_1, n_1, h_1, m_1) \neq f_{\mathsf{X}}(k_2, n_2, h_2, m_2)$ for any $(h, m) \subseteq \mathsf{X}$

(e) $m_1 \neq \perp$ and $m_2 \neq \perp$

Thus, $\mathcal{B}$ can simply output $\Big(c, (k_1, n_1, h_1), (k_2, n_2, h_2)\Big)$ and win whenever $\mathcal{A}$ wins.

3. Statement 3: We prove this claim by reduction. If there exists an attacker $\mathcal{A}$ that breaks the ϵ-X-FROB security of ccAEAD[AEAD] for $\mathsf{X} \subseteq (h, m)$, then we can construct an attacker $\mathcal{B}$ that breaks the ϵ'-r-BIND security of ccAEAD[AEAD] and $\epsilon' = \epsilon$. When $\mathcal{A}$ outputs $\Big(c, (k_1, n_1, h_1), (k_2, n_2, h_2)\Big)$, $\mathcal{A}$ wins if

   (a) $\perp \notin \{k_1, n_1, h_1, k_2, n_2, h_2\}$,

   (b) for $m_1 := \mathsf{AEAD.Dec}(k_1, n_1, h_1, c)$ and $m_2 := \mathsf{AEAD.Dec}(k_2, n_2, h_2, c)$ it holds that $f_{\mathsf{X}}(k_1, n_1, h_1, m_1) \neq f_{\mathsf{X}}(k_2, n_2, h_2, m_2)$ for any $\mathsf{X} \subseteq (h, m)$

   (c) $m_1 \neq \perp$ and $m_2 \neq \perp$

   This in particular indicates that

   (a) $\perp \notin \{k_1, n_1, h_1, m_1, k_2, n_2, h_2, m_2\}$

   (b) $(h_1, m_1) \neq (h_2, m_2)$

   (c) $m_1 = \mathsf{AEAD.Dec}(k_1, n_1, h_1, c)$

   (d) $m_2 = \mathsf{AEAD.Dec}(k_2, n_2, h_2, c)$

   Thus, $\mathcal{B}$ can simply output $\Big(c, (k_1, n_1, h_1, m_1), (k_2, n_2, h_2, m_2)\Big)$, where $m_1 := \mathsf{AEAD.Dec}(k_1, n_1, h_1, c)$ and $m_2 := \mathsf{AEAD.Dec}(k_2, n_2, h_2, c)$, and win whenever $\mathcal{A}$ wins.

□

4.5.7 Proof of Theorem 18

Proof. We prove each of these statements in turn.

1. Statement 1 and 2: We prove these statements by a counter-example AEAD, which is identical to the one in the proof of Statement 2 in Theorem 15. As shown in Theorem 15, we know that AEAD is KC secure. By Theorem 16, we know that AEAD is also MKCR secure. Below, we prove that ccAEAD[AEAD] is not ϵ'-r-BIND secure for any negligible ϵ'.

An attacker $\mathcal{A}$ can simply pick arbitrary $k_1, k_2 \in \mathcal{K}$, $n_1, n_2 \in \mathcal{N}$, $h_1, h_2 \in \mathcal{H}$, $m_1 \in \mathcal{M}$ such that $k_1 \neq k_2$, $n_2 = n_1 \oplus k_1 \oplus k_2$, $h_1 = h_2$. Then, $\mathcal{A}$ computes $c \leftarrow \mathsf{AEAD.Enc}(k_1, n_1, h_1, m_1)$ and $m_2 \leftarrow \mathsf{AEAD.Dec}(k_2, n_2, h_2, c)$. It holds that $m_2 = m_1 \oplus n_1 \oplus n_2$. By $k_1 \neq k_2$, we know that $m_2 = m_1 \oplus n_1 \oplus n_2 = m_1 \oplus k_1 \oplus k_2 \neq m_1$ and therefore $(h_1, m_1) \neq (h_2, m_2)$. Finally, $\mathcal{A}$ outputs $\Big(c, (k_1, n_1, h_1, m_1), (k_2, n_2, h_2, m_2)\Big)$ and always wins.

2. Statement 3: We prove this statement by a counter-example $\mathsf{ccAEAD}[\mathsf{AEAD}]$. Let $\mathsf{SKE} = (\mathsf{AEAD'.KGen}, \mathsf{AEAD'.Enc}, \mathsf{AEAD'.Dec})$ denote one-time pad with spaces $\mathcal{K}' = \mathcal{M}' = \{0,1\}^t$ for some $t > 0$. We then define $\mathsf{AEAD} = (\mathsf{AEAD.KGen}, \mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$ with space $\mathcal{K} = \{0,1\}^t$ and $\mathcal{M} = \{0,1\}^{\frac{t}{2}}$ from SKE and a collision-resistant function $\mathsf{F} : \mathcal{H} \times \mathcal{M} \to \mathcal{T}$ for some space $\mathcal{T}$ as follows:

   - $\mathsf{AEAD.KGen}()$: identical to $k \xleftarrow{\$} \mathsf{AEAD'.KGen}()$.
   - $\mathsf{AEAD.Enc}(k, n, h, m)$: runs $c' \leftarrow \mathsf{AEAD'.Enc}(k, m \parallel m)$ and $t \leftarrow \mathsf{F}(h, m)$, followed by outputting $c \leftarrow c' \parallel t$.
   - $\mathsf{AEAD'.Dec}(k, n, h, c)$: parses $c' \parallel t \leftarrow c$ and runs $m \parallel m' \leftarrow \mathsf{AEAD.Dec}(k, c')$, followed by outputting $\perp$ if $t \neq \mathsf{F}(h, m)$ and m otherwise.

   We first prove that $\mathsf{ccAEAD}[\mathsf{AEAD}]$ is $\mathsf{r\text{-}BIND}$: Note that an attacker $\mathcal{A}$ can break the $\mathsf{r\text{-}BIND}$ security of AEAD only when $\mathcal{A}$ outputs $(h_1, m_1) \neq (h_2, m_2)$. By the collision resistance of the underlying F, we know that $\mathsf{F}(h_1, m_1) \neq \mathsf{F}(h_2, m_2)$ except negligible probability. This further implies that for any $c = c' \parallel t$, at least one of the conditions $t = \mathsf{F}(h_1, m_1)$ and $t = \mathsf{F}(h_2, m_2)$ cannot hold except negligible probability. Thus, $\mathcal{A}$ can never win the $\mathsf{r\text{-}BIND}$ experiment with non-negligible probability.

   Below, we prove that $\mathsf{ccAEAD}[\mathsf{AEAD}]$ is not ϵ'-KC secure for any negligible ϵ'. An attacker $\mathcal{A}$ can simply pick arbitrary $(k_1, n_1, h_1, m_1) \in \mathcal{K} \times \mathcal{N} \times \mathcal{H} \times \mathcal{M}$ and invoke $\mathcal{O}_{\mathsf{Enc}}$ oracle with input (k_1, n_1, h_1, m_1) for a ciphertext c. Then, $\mathcal{A}$ sets k_2 identical to k_1 except flipping the final bit, $n_2 = n_1$, and $h_2 = h_1$, followed by querying $\mathcal{O}_{\mathsf{Dec}}$ oracle with input (k_2, n_2, h_2, c) for a message m_2. It is straightforward that $m_2 = m_1 \neq \perp$ and $\mathcal{A}$ always wins.

3. Statement 4: We prove this statement by a counter-example $\mathsf{ccAEAD}[\mathsf{AEAD}]$, which is identical to the one in above proof of Statement 3. From the above statement, we know that $\mathsf{ccAEAD}[\mathsf{AEAD}]$ is $\mathsf{r\text{-}BIND}$ secure. Below, we prove that $\mathsf{ccAEAD}[\mathsf{AEAD}]$ is not ϵ'-MKCR secure for any $\kappa \geq 2$ and any negligible ϵ'.

   An attacker $\mathcal{A}$ can easily pick $(k^\star, n^\star, h^\star, m^\star) \in \mathcal{K} \times \mathcal{N} \times \mathcal{H} \times \mathcal{M}$ and compute $c^\star \leftarrow \mathsf{AEAD.Enc}(k^\star, n^\star, h^\star, m^\star)$. Next, $\mathcal{A}$ sets $\mathcal{K}^\star = \{k : k \text{ and } k^\star \text{ have the same first half bits}\}$. Finally, $\mathcal{A}$ outputs $(\mathcal{K}^\star, n^\star, h^\star, c^\star)$. We have that for any $k \in \mathcal{K}^\star$, $\mathsf{AEAD.Dec}(k, n^\star, h^\star, c^\star) = m^\star \neq \perp$. Moreover, recall that $\mathcal{K} = \{0,1\}^t$ and $\mathcal{M} = \{0,1\}^{\frac{t}{2}}$ for arbitrary $t > 0$. For any $\kappa \geq 2$, $\mathcal{A}$ can always find suitable $t > 0$ such that $|\mathcal{K}^\star| = 2^{\frac{t}{2}} \geq \kappa$. Thus, $\mathcal{A}$ always wins, which concludes the proof. □

Chapter 5

Provable Security of FIDO2, CTAP2.1, and WebAuthn 2

This chapter is based on the paper:

Nina Bindel, Cas Cremers and Mang Zhao, "FIDO2, CTAP 2.1, and WebAuthn 2: Provable Security and Post-Quantum Instantiation," in 2023 2023 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, US, 2023 pp. 1471-1490. DOI: 10.1109/SP46215.2023.00039, URL: `https://doi.ieeecomputersociety.org/10.1109/SP46215.2023.00039`

This paper was joint work with Nina Bindel and Cas Cremers. All authors actively contributed to the completion of this work. The contributions in this chapter related to CTAP, the security proofs for WebAuthn, and the provable security analysis of the composition of CTAP and WebAuthn are my own. In the case of WebAuthn, the security model, the discovery of the downgrade attack, and its resolution were the result of collaborative research efforts.

5.1 Introduction

One of the largest projects globally to mitigate the problems of weak passwords is the FIDO protocol by the Fast Identity Online (FIDO) Alliance. The alliance has brought together over forty key companies in the online authentication space, including Amazon, Apple, Google, Intel, Microsoft, RSA, VISA, and Yubico, and has brought security devices to the wider public to improve the security of important logins.

The FIDO2 standard – the latest of the protocols – is built around two sub-protocols that are critical for enabling security-device supported logins. The first one is *WebAuthn*, which is a protocol between web applications, web browsers, and authenticator hardware tokens. At its core, WebAuthn allows a website (a Relying Party) to perform a *passwordless* challenge-response protocol with a token (an Authenticator) – where the browser acts as an intermediary – and challenges are signed by credential keys generated and stored in the token. The protocol supports multiple optional modes and features, such as attestation and user involvement.

The second relevant protocol is *CTAP* (Client To Authenticator Protocol), which is a protocol between an authenticator (e.g., a hardware security token) and a client (e.g., a browser). The goal of the protocol is to bind (and thus to restrict) which clients can use the authenticator's API (Application Programming Interface). To enable API access, the client asks the user to enter the authenticator's PIN; this PIN is checked by the token, and a shared secret is established that represents the binding and is used to authenticate all subsequent client accesses to the authenticator.

The FIDO2 standard, while already widely deployed, is subject to ongoing development. Previous versions of these standards have been studied. However, as we will see later, the main study has made strong assumptions that do no hold for the majority of deployed systems, such as relying on the attestation[1] mode to prove core properties. Moreover, the recently proposed CTAP 2.1 [59] includes a completely new base protocol that has not yet been analyzed in any framework.

Notably, the most recent version of the FIDO2 standard with CTAP 2.1 and WebAuthn 2 [107] appears to be "post-quantum ready", because it enables a mode of operation that only uses on symmetric cryptographic primitives, digital signatures, and KEMs (Key Encapsulation Mechanisms). However, no post-quantum instantiations have been proposed, nor has the CTAP 2.1 protocol received any analysis. In this work we set out to fill this gap: analyse the newest version and assess its post-quantum security.

[1]In the context of WebAuthn, "attestation" means identification of device type/manufacturer, and notably does not imply any check of the software that is being executed.

Contributions

1. We prove that FIDO2 with WebAuthn 2 and CTAP 2.1 is provably secure against classical attackers in a fine-grained security and protocol model. Our security models are more fine-grained or cover other aspects than previous versions such as [21, 103]. For example, we add important aspects such as algorithm negotiation, required user actions, and token binding. For CTAP 2.1, our security proofs confirm the stronger containment properties (reduced "blast radius") offered by the protocol compared to CTAP 2.0. Our analysis of WebAuthn 2 also has new implications for WebAuthn 1: we provide the first guarantees of the most widely used `None` attestation mode, user verification, user presence, and token binding.
2. We prove that if FIDO2 with WebAuthn 2 and CTAP 2.1 is instantiated with post-quantum (PQ) secure KEMs and signatures, then it is secure against quantum attackers in the same model. We give concrete suggestions for PQ secure algorithm and negotiation design choices, including classical-PQ hybrids as suggested by standardization agencies, such as NIST (National Institute for Standards and Technology) [69].
3. We propose a simple improvement to WebAuthn 2 that improves its resilience to certain types of downgrade attack. While these can only occur for strong threat models, these improvements yield stronger classical security against broken cryptographic primitives, and are even more relevant for their PQ instantiations.

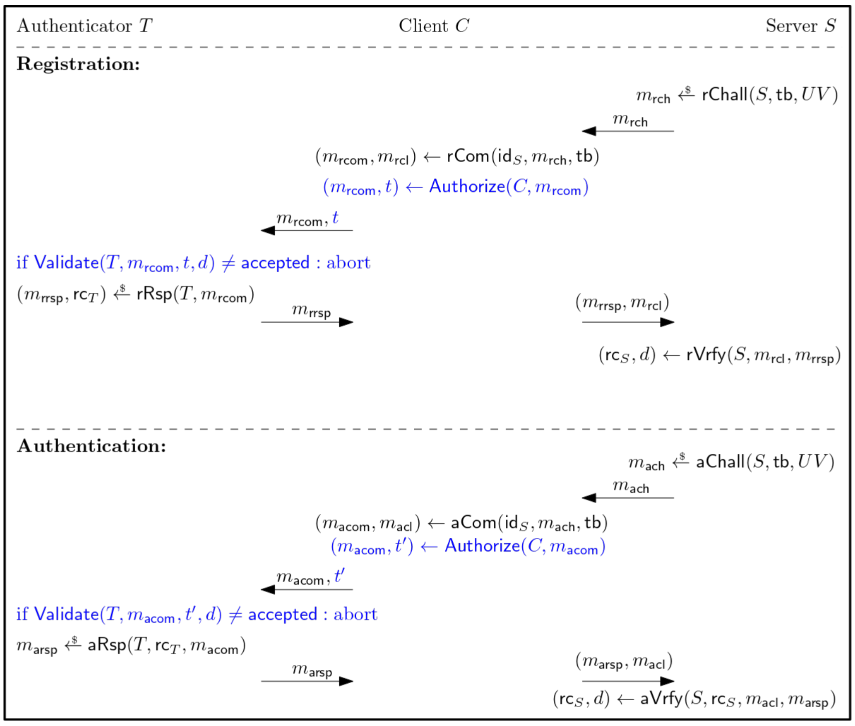
Overview

We provide a high-level background on FIDO2's CTAP and WebAuthn protocols, and previous analysis models, in Section 5.2. Next, we define additional notions and preliminaries in Section 5.3. Afterwards, we first present the analysis of WebAuthn 2 in Section 5.4, and then that of CTAP 2.1 in Section 5.5. We prove the security of their composition in FIDO2 in Section 5.6. We then return to related work in Section 5.7, and describe limitations and future work in Section 5.8. Finally, we give the full proofs of all theorems in this chapter in Section 5.9.

5.2 Background

5.2.1 High-level overview of FIDO2

The FIDO2 protocol incorporates the two sub-protocols WebAuthn and CTAP, and involves four main types of parties: relying parties (e.g., a server, online service, or an operating system feature), authenticators (e.g., token or security key), clients (e.g., web browsers or other applications), and users. WebAuthn typically leaves the users implicit in the description of the authenticator.



Authenticator T
Client C
Server S
Registration:
$m_{\mathsf{rch}} \xleftarrow{\$} \mathsf{rChall}(S, \mathsf{tb}, UV)$
m_{rch}
$(m_{\mathsf{rcom}}, m_{\mathsf{rcl}}) \leftarrow \mathsf{rCom}(\mathsf{id}_S, m_{\mathsf{rch}}, \mathsf{tb})$
$(m_{\mathsf{rcom}}, t) \leftarrow \mathsf{Authorize}(C, m_{\mathsf{rcom}})$
m_{rcom}, t
if $\mathsf{Validate}(T, m_{\mathsf{rcom}}, t, d) \neq \mathsf{accepted}$: abort
$(m_{\mathsf{rrsp}}, \mathsf{rc}_T) \xleftarrow{\$} \mathsf{rRsp}(T, m_{\mathsf{rcom}})$
m_{rrsp}
$(m_{\mathsf{rrsp}}, m_{\mathsf{rcl}})$
$(\mathsf{rc}_S, d) \leftarrow \mathsf{rVrfy}(S, m_{\mathsf{rcl}}, m_{\mathsf{rrsp}})$
Authentication:
$m_{\mathsf{ach}} \xleftarrow{\$} \mathsf{aChall}(S, \mathsf{tb}, UV)$
m_{ach}
$(m_{\mathsf{acom}}, m_{\mathsf{acl}}) \leftarrow \mathsf{aCom}(\mathsf{id}_S, m_{\mathsf{ach}}, \mathsf{tb})$
$(m_{\mathsf{acom}}, t') \leftarrow \mathsf{Authorize}(C, m_{\mathsf{acom}})$
m_{acom}, t'
if $\mathsf{Validate}(T, m_{\mathsf{acom}}, t', d) \neq \mathsf{accepted}$: abort
$m_{\mathsf{arsp}} \xleftarrow{\$} \mathsf{aRsp}(T, \mathsf{rc}_T, m_{\mathsf{acom}})$
m_{arsp}
$(m_{\mathsf{arsp}}, m_{\mathsf{acl}})$
$(\mathsf{rc}_S, d) \leftarrow \mathsf{aVrfy}(S, \mathsf{rc}_S, m_{\mathsf{acl}}, m_{\mathsf{arsp}})$

Figure 5.1: The main message flow of FIDO2 with WebAuthn 2 with attestation type `None` is shown in black. The blue flows indicate interaction with the third CTAP 2.1 phase (i.e. after CTAP 2.1's setup and binding phases.) The user is left implicit in the flow of the authenticator token. For registration, the server generates a challenge. This is forwarded through the client to the token (possibly authorized through CTAP 2.1), which returns a public credential key and additional data, which is stored by the server. Afterwards, for each authentication, a similar process occurs, but the token now signs challenge and data with the with the signing key corresponding to the public credential key that was registered previously.

Initially, the client and authenticator run the setup and binding phase of the CTAP 2.1 protocol. Once this is completed, relying parties can register and authenticate authenticators by running WebAuthn 2 through CTAP 2.1, which we depict in Figure 5.1. WebAuthn 2 is used in two phases: in the registration phase, an authenticator produces a fresh credential key pair whose public key is sent to the relying party and stored. Afterwards, each time the relying party wants to authenticate a user, it performs a challenge-response protocol with the authenticator who signs the challenge using the credential private key, which is then verified by the relying party. We next expand the two protocols in more detail.

5.2.1.1 WebAuthn

The goal of WebAuthn is to enable relying parties to authenticate users through authenticator tokens using a challenge-response protocol. WebAuthn is specified as an API rather than as a protocol; in practice, a common scenario is that the relying party is an online service with server backend code and Javascript running in the browser, and the server's Javascript then uses the WebAuthn API supported by the browser to communicate with the token. The first interaction, when the server communicates with the token, is called registration phase. In this phase, the server S sends a challenge message (m_{rch}) to the token through the client C. This challenge contains a random nonce, parameters such as whether user verification (UV) is required, and optionally a value `tb` that uniquely identifies the underlying channel (in practice typically identifying a unique Transport Layer Security (TLS) connection, which can provide channel binding to prevent some types of man-in-the-middle attacks).

The client C parses the challenge message and turns it into a command message (m_{rcom}) and a client message (m_{rcl}), and forwards the command message to the token T. The token T produces a credential public-private key pair, which is bound to the server S and enables S to perform verification during the following authentication phase, and outputs a response message (m_{rrsp}). The client then returns this together with the client message to the server S. The response message specifies the type of "attestation statement" selected by the token, which enables the server S to perform verification during the registration phase, and includes the credential public key. WebAuthn 2 supports five attestation types; these include `Basic` and `None`[2]. Tokens that support type `Basic` are equipped with an attestation key pair, which is specific to the token model, but not unique: by design, the attestation key pair is shared by a batch of tokens[3]. The `None` mode provides no token-specific information and is supported by all tokens.

The authentication phase is executed after the completion of the registration in a slightly different way. When the client parses the challenge message (m_{ach}) from the server S and turns it into a command message (m_{acom}) and a client message (m_{acl}), followed by sending the command message to the token T. The token T produces a response message (m_{arsp}) signed using the credential private key, and bound to the server S. The server S finally accepts a response message and a client message only when they pass verification using the corresponding credential public key.

5.2.1.2 CTAP

The CTAP protocol allows the client (e.g., a browser) to communicate with the authenticator. Using only WebAuthn, any application might try to access a token to request credential keys or responses to challenges. In practice, we would like to limit the client applications that are allowed to use the token's API. One of the goals of the CTAP protocol is to limit this access.

CTAP proceeds in three phases. In the setup phase, a client C' initializes a PIN, which is collected from the user, into the token T. In the binding phase, the client C (not necessarily same as C') and the token T exchange a shared binding state, if the client C

[2]The remaining three modes are: `Self`, `AttCA`, and `AnonCA`, which are less common and out of scope of this work.

[3]The number of tokens in each batch is at least 100,000, cf. [107, Section 14.4.1].

is able to provide information about the PIN stored on the token T. The binding state is expected to uniquely bind the client C to the token T. If the client C fails 3 times consecutively, the token T is rebooted and all previously established binding states are reset. If the client C fails 8 times in total, the token T is blocked. When the above preparation is done, the client C authorizes any command message by outputting a tag t, which is forward to the token T along with the command message itself. The token T only proceeds upon the positive decision d from the user, e.g., by pressing a button, and then validates the command message and the tag. In particular, a token only produces a response message in WebAuthn when its validation process in CTAP succeeds. Note that the binding state is repeatedly used during a period, the length of which depends on the concrete CTAP version and the type of token devices, and will be blocked afterwards.

5.2.2 Previous analysis by Barbosa et al. [20, 21]

Barbosa et al. [20, 21] gave the first formal analysis of FIDO2, and in particular the version with CTAP 2.0 and WebAuthn 1. We recall some important conclusions.

1. **WebAutnn**: Barbosa et al. formalize WebAuthn 1 as a *passwordless authentication* (PlA) protocol. Assuming the uniqueness of each attestation key pair, they then prove that WebAuthn 1 with attestation type `Basic` provides *secure passwordless authentication*. However, since each attestation key pair is in fact necessarily shared by a large batch of tokens (often called *batch attestation*), their main theorem establishes uniqueness properties of partnering for each batch of tokens that share the same attestation key pair instead of each single token. Moreover, their analysis has no clear implications for the `None` mode.
2. **CTAP**: Barbosa et al. formalize CTAP 2.0 as a *PIN-based access control for authenticators* (PACA) protocol. Then, they prove the *Unforgeability with trusted binding* (UF-t) of CTAP 2.0. In Section 5.7.1 we show that the difference between CTAP 2.0 and CTAP 2.1 is substantial, which means the previous results cannot simply be translated.

Thus, Barbosa et al. [20] provided the first formal analysis of FIDO2 with CTAP 2.0 and WebAuthn 1, which was ground-breaking in many ways, but as a first attempt also left open many questions and subtle proof issues. We provide a detailed comparison between [20] and our work in Section 5.7.2.

5.3 Additional Preliminaries

5.3.1 Additional Security Definitions

We define a new customized notion for SKE: the IND-CPA security with respect to function H (IND-1CPA-H). Compared to IND-1CPA security, the attacker additionally obtains a challenge tag that is produced by applying H to both a symmetric key, which is same as the one used by the SKE, and the challenge ciphertext.

Definition 46. *Let* $\mathsf{SKE} = (\mathsf{SKE.KGen}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ *be a symmetric key encryption scheme with symmetric key space* $\mathcal{K}$*. We say* SKE *is* ϵ*-one time* $\mathsf{IND\text{-}CPA}$ *secure with respect to function* H *(denoted by* $\mathsf{IND\text{-}1CPA\text{-}H}$*) secure, if the blow defined advantage of every (potential quantum) attacker* $\mathcal{A}$ *against* $\mathrm{Expr}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CPA\text{-}H}}$ *experiment in Figure 5.2 is bounded by,*

$$\mathsf{Adv}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CPA\text{-}H}}(\mathcal{A}) := \left| \Pr[\mathrm{Expr}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CPA\text{-}H}}(\mathcal{A}) = 1] - \frac{1}{2} \right| \leq \epsilon.$$

We further extend $\mathsf{IND\text{-}1\$PA}$ security to a new notion $\mathsf{IND\text{-}1\$PA\text{-}LPC}$ that additionally gives the attacker the access to a challenge plaintext checking oracle, which returns whether the input ciphertext can be decrypted to the challenge plaintext.

Definition 47. *Let* $\mathsf{SKE} = (\mathsf{SKE.KGen}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ *be a symmetric key encryption scheme with symmetric key space* $\mathcal{K}$*. We say* SKE *is* ϵ*-*$\mathsf{IND\text{-}1\$PA\text{-}LPC}$ *secure, if the blow defined advantage of every (potential quantum) attacker* $\mathcal{A}$ *against* $\mathrm{Expr}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LPC}}$ *experiment in Figure 5.2 is bounded by,*

$$\mathsf{Adv}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LPC}}(\mathcal{A}) := \left| \Pr[\mathrm{Expr}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LPC}}(\mathcal{A}) = 1] - \frac{1}{2} \right| \leq \epsilon.$$

```
$\mathrm{Expr}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CPA\text{-}H}}(\mathcal{A})$:
    $\mathsf{b} \xleftarrow{\$} \{0,1\}$
    $K \xleftarrow{\$} \mathsf{SKE.KGen}()$
    $(m_0^\star, m_1^\star) \xleftarrow{\$} \mathcal{A}()$
    if $|m_0^\star| \neq |m_1^\star|$
        return 0
    $c^\star \xleftarrow{\$} \mathsf{SKE.Enc}(K, m_{\mathsf{b}}^\star)$
    $t^\star \leftarrow \mathsf{H}(K, c^\star)$
    $\mathsf{b}' \xleftarrow{\$} \mathcal{A}(c^\star, t^\star)$
    return $[\![\mathsf{b} = \mathsf{b}']\!]$
```

```
$\mathrm{Expr}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LPC}}(\mathcal{A})$:
    $\mathsf{b} \xleftarrow{\$} \{0,1\}$
    $K \xleftarrow{\$} \mathsf{SKE.KGen}()$
    $(m_0^\star, m_1^\star) \xleftarrow{\$} \mathcal{A}()$
    if $|m_0^\star| \neq |m_1^\star|$
        return 0
    $c^\star \xleftarrow{\$} \mathsf{SKE.Enc}(K, m_{\mathsf{b}}^\star)$
    $\mathsf{b}' \xleftarrow{\$} \mathcal{A}^{\textsc{Rand},\textsc{Lpc}}(c^\star)$
    return $[\![\mathsf{b} = \mathsf{b}']\!]$
```

```
$\textsc{Rand}(l)$:
    $m_0' \xleftarrow{\$} \{0,1\}^l$
    $m_1' \xleftarrow{\$} \{0,1\}^l$
    $c' \xleftarrow{\$} \mathsf{SKE.Enc}(K, m_{\mathsf{b}}')$
    return $(m_0', m_1', c')$
$\textsc{Lpc}(c)$:
    if $c = c^\star$
        return 0
    return $[\![m_0^\star = \mathsf{SKE.Dec}(K, c)]\!]$
```

Figure 5.2: $\mathsf{IND\text{-}1CPA\text{-}H}$ and $\mathsf{IND\text{-}1\$PA\text{-}LPC}$ experiments for a $\mathsf{SKE} = (\mathsf{SKE.KGen}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ scheme.

5.3.2 CBC Mode and $\mathsf{IND\text{-}1\$PA\text{-}LPC}$ Security

The Cipher Block Chaining (CBC) mode is a block cipher mode of operation invented by Ehrsam et al. in 1976 [87]. The CBC can be divided into two categories: CBC_0, whose initial vector is a string of zero bits, and CBC_R, whose initial vector is a random bit string. We first recall CBC as an instance of symmetric key encryption. Let $\mathcal{K} := \{0,1\}^{f_1(\lambda)}$,

$\mathcal{M} := \{0,1\}^{f_2(\lambda)}$, and $\mathcal{O} := \{0,1\}^{f_2(\lambda)}$ respectively denote the symmetric key space, message space, and output space of an invertible function $\mathsf{F} : \mathcal{K} \times \mathcal{M} \to \mathcal{O}$, where f_1 and f_2 denote arbitrary polynomial functions. Then, both CBC_0 and CBC_R are defined in Figure 5.3. Here, we simply assume that the input message m of the encryption algorithm always has the length of a multiple of $f_2(\lambda)$. It is straightforward that CBC_0 is a deterministic encryption scheme.

```
SKE.KGen(1^λ):
    $K \xleftarrow{\$} \mathcal{K}$
    return $K$
SKE.Enc(K, m):
    $x_1 \| ... \| x_n \leftarrow m$ s.t. $|x_i| = f_2(\lambda)\ \forall i \in [n]$
    $y_0 \xleftarrow{\$} \mathsf{SetIV}()$
    for $i = 1, ..., n$
        $y_i \leftarrow \mathsf{F}(K, y_{i-1} \oplus x_i)$
    $y \leftarrow y_0 \| \cdots \| y_n$
    return $y$

SKE.Dec(K, c):
    $y_0 \| ... \| y_n \leftarrow c$ s.t. $|y_i| = f_2(\lambda)\ \forall i \in [n]$
    for $i = 1, ..., n$
        $x_i \leftarrow y_{i-1} \oplus \mathsf{F}^{-1}(K, y_i)$
    $m \leftarrow x_1 \| \cdots \| x_n$
    return $m$
```

Figure 5.3: CBC mode $\mathsf{SKE} = (\mathsf{SKE.KGen}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ with symmetric key space $\mathcal{K} := \{0,1\}^{f_1(\lambda)}$ for arbitrary polynomial function f_1. If $\mathsf{SKE} = \mathsf{CBC}_0$, then $\mathsf{SetIV}()$ outputs a string of zero bits of length $f_2(\lambda)$. If $\mathsf{SKE} = \mathsf{CBC}_R$, then $\mathsf{SetIV}()$ outputs a random string of length $f_2(\lambda)$.

The IND-1\$PA security of the deterministic CBC_0 was proven by Barbosa et al. [20]. Moreover, the IND-CPA security of the randomized CBC_R was proven by Bellare et al. [25]. Below, we prove the IND-1\$PA-LPC security of both CBC_0 and CBC_R based on above two security conclusions.

Theorem 19 (IND-CPA $\implies$ IND-1\$PA). *Let* $\mathsf{SKE} = (\mathsf{SKE.KGen}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ *denote a symmetric encryption scheme. If* SKE *is* $\epsilon^{\text{ind-cpa}}_{\mathsf{SKE}}$-IND-CPA *secure, then* SKE *is* $\epsilon^{\text{ind-1\$pa}}_{\mathsf{SKE}}$-IND-1\$PA *secure such that* $\epsilon^{\text{ind-1\$pa}}_{\mathsf{SKE}} \le \epsilon^{\text{ind-cpa}}_{\mathsf{SKE}}$.

Theorem 20 (IND-1\$PA $\implies$ IND-1\$PA-LPC). *Let* $\mathsf{SKE} = (\mathsf{SKE.KGen}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ *denote* CBC_0 *or* CBC_R. *If* SKE *is* $\epsilon^{\text{ind-1\$pa}}_{\mathsf{SKE}}$-IND-1\$PA *secure and the underlying function* $\mathsf{F} : \{0,1\}^{f_1(\lambda)} \times \{0,1\}^{f_2(\lambda)} \to \{0,1\}^{f_2(\lambda)}$ *is* $\epsilon^{\text{prp}}_{\mathsf{F}}$-prp *secure, then* SKE *is* $\epsilon^{\text{ind-1\$pa-lpc}}_{\mathsf{SKE}}$-IND-1\$PA-LPC *secure such that*

$$\epsilon^{\text{ind-1\$pa-lpc}}_{\mathsf{SKE}} \le 2\epsilon^{\text{prp}}_{\mathsf{F}} + q_{\textsc{Lpc}} 2^{-f_2(\lambda)} + q_{\textsc{Rand}} \lceil \frac{l_{\max}}{f_2(\lambda)} \rceil 2^{-f_2(\lambda)} + \epsilon^{\text{ind-1\$pa}}_{\mathsf{SKE}}$$

where $q_{\mathcal{O}}$ *denotes the maximal number of queries to* $\mathcal{O} \in \{\textsc{Rand}, \textsc{Lpc}\}$ *oracles and* $l_{\max}$ *denotes the maximal input to the* $\textsc{Rand}$ *oracle.*

5.4 WebAuthn 2 and Extended Passwordless Authentication Protocols

For our analysis of WebAuthn 2 and its PQ instantiation, we follow the high-level approach from [20, 21], which proposed the class of PlA protocols that generalizes WebAuthn 1, and proposed a corresponding security notion. We provide a more fine-grained model of WebAuthn 2, notably including the default mode `None` in which no attestation is performed, as well as the user presence and user verification checks, and a stronger threat model. We compare the details in our work and [20] in Section 5.7.2 These aspects and their security cannot be captured in the PlA class without modification. In this section, we therefore first extend [20]'s formalisation and propose the *extended* PlA (ePlA) protocol class, and instantiate WebAuthn 2 as an ePlA protocol. We then introduce our new model to define *secure passwordless authentication* (auth) for ePlA protocols and prove that WebAuthn 2 satisfies it. We then show how to instantiate PQ-WebAuthn 2. Our proof of auth implies PQ security against a QPT attacker if the schemes used in a session are PQ secure. We return to downgrade attacks in Section 5.4.6.

5.4.1 Extended Passwordless Authentication Protocols (ePlA)

Similar to the PlA model from [20], we define our *extended passwordless authentication protocol* ePlA by two phases, $\mathsf{Register}$ and $\mathsf{Authenticate}$:

$\mathsf{Register}$: a two-pass challenge-response protocol run between a token T, a client C, and a server S, which is run at most once per tuple (T, S) (i.e., not for additional clients). At the end, both T and S hold registration contexts, which are relevant for subsequent authentications. $\mathsf{Register}$ can be decomposed into the following algorithms:

- rChall: inputs a server S, a token binding state tb, and a user verification condition $UV \in \{\mathsf{true}, \mathsf{false}\}$, and outputs a challenge message m_{rch}, i.e., $m_{\mathsf{rch}} \xleftarrow{\$} \mathsf{rChall}(S, \mathsf{tb}, UV)$.
- rCom: inputs the intended server identity id_S, a challenge message m_{rch}, and a token binding state tb, and outputs a client message m_{rcl} and a command message m_{rcom}, i.e., $(m_{\mathsf{rcom}}, m_{\mathsf{rcl}}) \leftarrow \mathsf{rCom}(\mathsf{id}_S, m_{\mathsf{rch}}, \mathsf{tb})$.
- rRsp: inputs a token T and a command message m_{rcom} and outputs a response message m_{rrsp} and an token-associated registration context rc_T, i.e., $(m_{\mathsf{rrsp}}, \mathsf{rc}_T) \xleftarrow{\$} \mathsf{rRsp}(T, m_{\mathsf{rcom}})$.
- rVrfy: inputs a server S, a client message m_{rcl}, and a response message m_{rrsp}, and outputs a server-associated registration context rc_S and a decision bit $d \in \{0, 1\}$ to indicate whether the registration request was accepted ($d = 1$) or not ($d = 0$), i.e., $(\mathsf{rc}_S, d) \leftarrow \mathsf{rVrfy}(S, m_{\mathsf{rcl}}, m_{\mathsf{rrsp}})$.

Authenticate: a two-pass challenge-response protocol run between a token T, a client C, and a server S after a successful run of Register, in which both T and S generated their registration contexts. At the end, S either accepts or rejects the authentication attempt. Similarly to Register, Authenticate can be decomposed into four algorithms:

aChall: inputs a server S, a token binding state tb, and a user verification condition $UV \in \{\mathsf{true}, \mathsf{false}\}$, and outputs a challenge message m_{ach}, i.e., $m_{\mathsf{ach}} \xleftarrow{\$} \mathsf{aChall}(S, \mathsf{tb}, UV)$.

aCom: inputs the intended server identity id_S, a challenge message m_{ach}, and a token binding state tb, and outputs a client message m_{acl} and a command message m_{acom}, i.e., $(m_{\mathsf{acl}}, m_{\mathsf{acom}}) \leftarrow \mathsf{aCom}(\mathsf{id}_S, m_{\mathsf{ach}}, \mathsf{tb})$.

aRsp: inputs a token T along with its associated registration context rc_T, and a command message m_{acom}, and outputs a response message m_{arsp} and the updated registration context rc_T, i.e., $(m_{\mathsf{arsp}}, \mathsf{rc}_T) \xleftarrow{\$} \mathsf{aRsp}(T, \mathsf{rc}_T, m_{\mathsf{acom}})$.

aVrfy: inputs a server S along with its associated registration context rc_S, a client message m_{acl}, and a response message m_{arsp}, and outputs the updated registration context rc_S and a decision bit $d \in \{0, 1\}$ indicating whether the authentication request was accepted by the user (output 1) or not (output 0), i.e., $(\mathsf{rc}_S, d) \leftarrow \mathsf{aVrfy}(S, \mathsf{rc}_S, m_{\mathsf{acl}}, m_{\mathsf{arsp}})$.

To model concurrent or sequential sessions of a server S (associated with ID id_S) and sequential sessions of a token T, we use π_S^i and π_T^j to denote their i-th and j-th instances respectively, i.e., $S = \{\pi_S^i\}_i$ and $T = \{\pi_T^j\}_j$. Our new abstraction retains the black message flow from Figure 5.1.

5.4.2 WebAuthn 2 is an ePlA Protocol

We use the following session variables for WebAuthn 2.

$\pi_S^i.\mathsf{ch}$: challenge nonce sampled in this session

$\pi_S^i.\mathsf{uid}$: user identifier sampled in this session

$\pi_S^i.\mathsf{tb}$: token binding state used in this session

$\pi_S^i.UV$: user verification condition, indicating whether the user should be verified, e.g., via PIN or Biometrics

$\pi_S^i.UP$: user presence condition, indicating whether the presence of the user is sufficient; constant true value

$\pi_S^i.\mathsf{pkCP}$: list of digital signature schemes accepted by S

$\pi_T^j.\mathsf{suppUV}$: indicates whether T supports user verification

$\pi_S^i.\mathsf{st}_{\mathsf{exe}}, \pi_T^j.\mathsf{st}_{\mathsf{exe}} \in \{\perp, \mathsf{running}, \mathsf{accepted}\}$: execution state of each session

$\pi_S^i.\mathsf{agCon}, \pi_T^j.\mathsf{agCon}$: the content that is expected to be agreed with other parties. These variables are protocol-specific. In WebAuthn 2, both variables include the server identifier, the hash of the client messages, the UP and UV conditions, and other session-specific data.

$\pi_S^i.\mathsf{sid}, \pi_T^j.\mathsf{sid}$: session identifiers. Two distinct sessions that have communicated with each other are expected to own the identical session identifiers.

These variables are protocol-specific. In WebAuthn 2, both variables include the hash of the server identifier and other session-specific data.

WebAuthn 2 Protocol Intuition. Intuitively, the registration phase starts with the execution of rChall algorithm, where the server S samples random challenge nonce $\pi_S^i.\mathsf{ch}$ and user identifier $\pi_S^i.\mathsf{uid}$, initializes the user verification condition $\pi_S^i.UV$, and outputs the challenge message m_{rch}, which includes the above data as well as the server domain id_S and accepted list $\pi_S^i.\mathsf{pkCP}$. Additionally, the server also stores the token binding states $\pi_S^i.\mathsf{tb}$, which is shared with a client. Receiving m_{rch}, the client is supposed to verify the server domain followed by computing the hash value h of the client message $m_{\mathsf{rcl}} := (\mathsf{ch}, \mathsf{tb})$. Compared with m_{rch}, the output command message m_{rcom} replaces ch with h and add a constant user presence condition $UP := \mathsf{true}$. Receiving m_{rcom}, the token T picks a suitable signature scheme DS in the list pkCP (if available) and checks whether the user verification mechanism is supported (if required). After that, T samples a public-private key pair (vk, sk) of DS and a credential identifier cid, followed by initializing the associated registration context $\mathsf{rc}_T[\mathsf{id}_S]$ and the agreed content $\pi_T^j.\mathsf{agCon}$. The session identifier $\pi_T^j.\mathsf{sid}$ is set to the hash of the server domain, the credential identifier, and the initial counter $n := 0$. The output response message m_{rrsp} includes the session identifier $\pi_T^j.\mathsf{sid}$ as well as vk, DS, UP and UV. The server S finally inputs both m_{rcl} and m_{rrsp} and executes a number of checks. If all checks pass, S also initializes its associated registration context $\mathsf{rc}_S[\mathsf{cid}]$ and the agreed content $\pi_S^i.\mathsf{agCon}$. The session identifier $\pi_S^i.\mathsf{sid}$ is identical to $\pi_T^j.\mathsf{sid}$.

The authentication phase is very similar. The crucial difference is that the token outputs a signature, which signs the $\pi_S^i.\mathsf{agCon}$-relevant data using the private key sk of DS. Moreover, the session identifiers of token and servers additionally include the hash of the client message m_{acl}.

We give the concrete definition of algorithms of WebAuthn 2 with the default attestation type `None` in Section 5.4.3.

5.4.3 Detailed Description of WebAuthn 2

In Figure 5.4, the security parameter $\lambda = 128$. For each server S, the associated identifier id_S is its effective domain. The official supported signature algorithms are RSASSA–PKCS1–v1_5 and RSASSA–PSS. Later in Section 5.4.5, we will show that the list of

```
Register
$\mathsf{rChall}(\pi_S^i, \mathsf{tb}, UV)$: // 1. Server
1  $\pi_S^i.\mathsf{ch} \xleftarrow{\$} \{0,1\}^{\geq\lambda}$, $\pi_S^i.\mathsf{tb} \leftarrow \mathsf{tb}$, $\pi_S^i.UV \leftarrow UV$
2  $\pi_S^i.\mathsf{uid} \xleftarrow{\$} \{0,1\}^{\leq 4\lambda}$
3  $m_{\mathsf{rch}} \leftarrow (\mathsf{id}_S, \pi_S^i.\mathsf{ch}, \pi_S^i.\mathsf{uid}, \pi_S^i.\mathsf{pkCP}, \pi_S^i.UV)$
4  $\pi_S^i.\mathsf{st_{exe}} \leftarrow \mathsf{running}$
5  return $m_{\mathsf{rch}}$
$\mathsf{rCom}(\mathsf{id}_S, m_{\mathsf{rch}}, \mathsf{tb})$: // 2. Client
6  $(\mathsf{id}, \mathsf{ch}, \mathsf{uid}, \mathsf{pkCP}, UV) \leftarrow m_{\mathsf{rch}}$
7  if $\mathsf{id} \neq \mathsf{id}_S$: return $\perp$
8  $m_{\mathsf{rcl}} \leftarrow (\mathsf{ch}, \mathsf{tb})$
9  $UP \leftarrow \mathsf{true}$, $h \leftarrow \mathsf{H}(m_{\mathsf{rcl}})$
10 $m_{\mathsf{rcom}} \leftarrow (\mathsf{id}, \mathsf{uid}, h, \mathsf{pkCP}, UP, UV)$
11 return $(m_{\mathsf{rcom}}, m_{\mathsf{rcl}})$
$\mathsf{rRsp}(\pi_T^j, m_{\mathsf{rcom}})$: // 3. Token
12 $(\mathsf{id}, \mathsf{uid}, h, \mathsf{pkCP}, UP, UV) \leftarrow m_{\mathsf{rcom}}$
13 if at least one algorithm in $\mathsf{pkCP}$ is supported
14   $\mathsf{DS} \leftarrow \mathsf{pkCP}[i]$ with smallest $i$ possible
15 else return $(\perp, \perp)$
16 if $\pi_T^j.\mathsf{suppUV} = \mathsf{false}$ and $UV = \mathsf{true}$
17   return $(\perp, \perp)$
18 $(vk, sk) \xleftarrow{\$} \mathsf{DS.KGen}()$, $\mathsf{cid} \xleftarrow{\$} \{0,1\}^{\geq\lambda}$, $n \leftarrow 0$
19 $m_{\mathsf{rrsp}} \leftarrow (\mathsf{H}(\mathsf{id}), n, \mathsf{cid}, vk, \mathsf{DS}, UP, UV)$
20 $\boxed{\mathsf{h_{CP}} \leftarrow \mathsf{H}(\mathsf{pkCP})}$
21 $\mathsf{rc}_T[\mathsf{id}] \leftarrow (\mathsf{uid}, \mathsf{cid}, sk, n, \mathsf{DS}, \boxed{\mathsf{h_{CP}}})$
22 $\pi_T^j.\mathsf{agCon} \leftarrow (\mathsf{id}, h, \mathsf{cid}, n, \mathsf{pkCP}, vk, \mathsf{DS}, UV, UP)$
23 $\pi_T^j.\mathsf{sid} \leftarrow (\mathsf{H}(\mathsf{id}), \mathsf{cid}, n)$
24 $\pi_T^j.\mathsf{st_{exe}} \leftarrow \mathsf{accepted}$
25 return $(m_{\mathsf{rrsp}}, \mathsf{rc}_T)$
$\mathsf{rVrfy}(\pi_S^i, m_{\mathsf{rcl}}, m_{\mathsf{rrsp}})$: // 4. Server
26 $(\mathsf{ch}, \mathsf{tb}) \leftarrow m_{\mathsf{rcl}}$, $(h, n, \mathsf{cid}, vk, \mathsf{DS}, UP, UV) \leftarrow m_{\mathsf{rrsp}}$
27 if $h \neq \mathsf{H}(\mathsf{id}_S)$ or $n \neq 0$ or $\mathsf{ch} \neq \pi_S^i.\mathsf{ch}$ or $\mathsf{tb} \neq \pi_S^i.\mathsf{tb}$ or
   $\mathsf{DS} \notin \pi_S^i.\mathsf{pkCP}$ or $UP \neq \mathsf{true}$ or $UV \neq \pi_S^i.UV$
28   return $(\perp, 0)$
29 $\boxed{\mathsf{h_{CP}} \leftarrow \mathsf{H}(\pi_S^i.\mathsf{pkCP})}$
30 $\mathsf{rc}_S[\mathsf{cid}] \leftarrow (\pi_S^i.\mathsf{uid}, vk, n, \mathsf{DS}, \boxed{\mathsf{h_{CP}}})$
31 $\pi_S^i.\mathsf{agCon} \leftarrow (\mathsf{id}_S, \mathsf{H}(m_{\mathsf{rcl}}), \mathsf{cid}, n, \pi_S^i.\mathsf{pkCP}, vk, \mathsf{DS}, UV, UP)$
32 $\pi_S^i.\mathsf{sid} \leftarrow (\mathsf{H}(\mathsf{id}), \mathsf{cid}, n)$
33 $\pi_S^i.\mathsf{st_{exe}} \leftarrow \mathsf{accepted}$
34 return $(\mathsf{rc}_S, 1)$
```

```
Authenticate
$\mathsf{aChall}(\pi_S^i, \mathsf{tb}, UV)$: // 1. Server
35 $\pi_S^i.\mathsf{ch} \xleftarrow{\$} \{0,1\}^{\geq\lambda}$, $\pi_S^i.\mathsf{tb} \leftarrow \mathsf{tb}$, $\pi_S^i.UV \leftarrow UV$
36 $m_{\mathsf{ach}} \leftarrow (\mathsf{id}_S, \pi_S^i.\mathsf{ch}, \pi_S^i.UP, \pi_S^i.UV)$
37 $\pi_S^i.\mathsf{st_{exe}} \leftarrow \mathsf{running}$
38 return $m_{\mathsf{ach}}$
$\mathsf{aCom}(\mathsf{id}_S, m_{\mathsf{ach}}, \mathsf{tb})$: // 2. Client
39 $(\mathsf{id}, \mathsf{ch}, UV) \leftarrow m_{\mathsf{ach}}$
40 if $\mathsf{id} \neq \mathsf{id}_S$: return $\perp$
41 $m_{\mathsf{acl}} \leftarrow (\mathsf{ch}, \mathsf{tb})$
42 $UP \leftarrow \mathsf{true}$, $h \leftarrow \mathsf{H}(m_{\mathsf{acl}})$
43 $m_{\mathsf{acom}} \leftarrow (\mathsf{id}, h, UP, UV)$
44 return $(m_{\mathsf{acom}}, m_{\mathsf{acl}})$
$\mathsf{aRsp}(\pi_T^j, \mathsf{rc}_T, m_{\mathsf{acom}})$: // 3. Token
45 $(\mathsf{id}, h, UP, UV) \leftarrow m_{\mathsf{acom}}$
46 if $\mathsf{rc}_T[\mathsf{id}] = \perp$: return $(\perp, \mathsf{rc}_T)$
47 if $\pi_T^j.\mathsf{suppUV} = \mathsf{false}$ and $UV = \mathsf{true}$
48   return $(\perp, \mathsf{rc}_T)$
49 $\mathsf{rc}_T[\mathsf{id}].n \leftarrow \mathsf{rc}_T[\mathsf{id}].n + 1$
50 $ad \leftarrow (\mathsf{H}(\mathsf{id}), \mathsf{rc}_T[\mathsf{id}].n, UP, UV)$
51 $\sigma \xleftarrow{\$} \mathsf{rc}_T[\mathsf{id}].\mathsf{DS.Sign}(\mathsf{rc}_T[\mathsf{id}].sk, (ad, h))$
52 $m_{\mathsf{arsp}} \leftarrow (\mathsf{rc}_T[\mathsf{id}].\mathsf{cid}, ad, \boxed{\mathsf{rc}_T[\mathsf{id}].\mathsf{h_{CP}},} \sigma, \mathsf{rc}_T[\mathsf{id}].\mathsf{uid})$
53 $\pi_T^j.\mathsf{agCon} \leftarrow (\mathsf{id}, h, \mathsf{rc}_T[\mathsf{id}].n, \boxed{\mathsf{rc}_T[\mathsf{id}].\mathsf{h_{CP}},} UV, UP)$
54 $\pi_T^j.\mathsf{sid} \leftarrow (\mathsf{H}(\mathsf{id}), \mathsf{rc}_T[\mathsf{id}].\mathsf{cid}, h, n)$
55 $\pi_T^j.\mathsf{st_{exe}} \leftarrow \mathsf{accepted}$
56 return $(m_{\mathsf{arsp}}, \mathsf{rc}_T)$
$\mathsf{aVrfy}(\pi_S^i, \mathsf{rc}_S, m_{\mathsf{acl}}, m_{\mathsf{arsp}})$: // 4. Server
57 $(\mathsf{ch}, \mathsf{tb}) \leftarrow m_{\mathsf{acl}}$, $(\mathsf{cid}, ad, \boxed{\mathsf{h_{CP}},} \sigma, \mathsf{uid}) \leftarrow m_{\mathsf{arsp}}$
58 $(h, n, UP, UV) \leftarrow ad$
59 if $\mathsf{rc}_S[\mathsf{cid}] = \perp$: return $(\mathsf{rc}_S, 0)$
60 $\boxed{\text{if } \mathsf{h_{CP}} \neq \mathsf{rc}_S[\mathsf{cid}].\mathsf{h_{CP}}\text{: } \mathsf{rc}_S[\mathsf{cid}] \leftarrow \perp \text{ and return } (\mathsf{rc}_S, 0)}$
61 if $\pi_S^i.\mathsf{ch} \neq \mathsf{ch}$ or $\pi_S^i.\mathsf{tb} \neq \mathsf{tb}$ or $h \neq \mathsf{H}(\mathsf{id}_S)$
   or $UP \neq \mathsf{true}$ or $UV \neq \pi_S^i.UV$ or
   $\mathsf{rc}_S[\mathsf{cid}].\mathsf{DS.Vfy}(\mathsf{rc}_S[\mathsf{cid}].vk, (ad, \mathsf{H}(m_{\mathsf{acl}})), \sigma) = 0$ or
   $n \leq \mathsf{rc}_S[\mathsf{cid}].n$: return $(\mathsf{rc}_S, 0)$
62 $\mathsf{rc}_S[\mathsf{cid}].n \leftarrow n$
63 $\pi_S^i.\mathsf{agCon} \leftarrow (\mathsf{id}_S, \mathsf{H}(m_{\mathsf{acl}}), n, \boxed{\mathsf{h_{CP}},} UV, UP)$
64 $\pi_S^i.\mathsf{sid} \leftarrow (h, \mathsf{cid}, \mathsf{H}(m_{\mathsf{acl}}), n)$
65 $\pi_S^i.\mathsf{st_{exe}} \leftarrow \mathsf{accepted}$
66 return $(\mathsf{rc}_S, 1)$
```

Figure 5.4: Instantiation of $\mathsf{ePIA} = (\mathsf{Register}, \mathsf{Authenticate})$ with WebAuthn 2 (and WebAuthn 2^+ that includes boxed operations) with attestation type `None`, where $\mathsf{Register} = (\mathsf{rChall}, \mathsf{rCom}, \mathsf{rRsp}, \mathsf{rVrfy})$ and $\mathsf{Authenticate} = (\mathsf{aChall}, \mathsf{aCom}, \mathsf{aRsp}, \mathsf{aVrfy})$. Recall that λ is the implicit security parameter.

signature schemes can be extended by PQ compatible hybrid signature scheme. The underlying hash function H is SHA-256. We assume that each token has a unique user and can be registered at most once per server. The $\mathsf{Register} = (\mathsf{rChall}, \mathsf{rCom}, \mathsf{rRsp}, \mathsf{rVrfy})$ sub-protocol is executed as follows.

- $\mathsf{rChall}(\pi_S^i, \mathsf{tb}, UV)$: The server S samples a random challenge nonce $\pi_S^i.\mathsf{ch}$ and a user identifier $\pi_S^i.\mathsf{uid}$ and initializes the token biding state $\pi_S^i.\mathsf{tb}$ and user verification condition $\pi_S^i.UV$. Finally, S sets $\pi_S^i.\mathsf{st_{exe}}$ to $\mathsf{running}$ and outputs a challenge message, see Line 3.

- $\mathsf{rCom}(\mathsf{id}_S, m_{\mathsf{rch}}, \mathsf{tb})$: The client parses m_{rch} into a server identifier id, a challenge nonce ch, a user identifier uid, a supported signature list pkCP and a user verification condition UV. Next, the client aborts if $\mathsf{id} \neq \mathsf{id}_S$. Otherwise, the client sets the user presence condition UP to true and computes the hash h of client message m_{rcl}, which is defined in Line 8. Finally, the client outputs the client and command messages m_{rcl} and m_{rcom}, respectively, see Line 10.
- $\mathsf{rRsp}(\pi_T^j, m_{\mathsf{rcom}})$: The token T first parses m_{rcom} into a server identifier id, a user identifier uid, a hash value h, a signature list pkCP, and the user presence and user verification conditions UP and UV, respectively. Next, T picks one supported signature scheme DS in pkCP with the highest preference, i.e., with the smallest index possible. Afterwards, T checks whether it can support the required user verification condition UV. If either step fails, the token aborts. Otherwise, T generates a public-private key pair using the key generation algorithm of DS, initializes the counter n to 0, samples a random credential identifier cid, and sets its execution state to $\mathsf{accepted}$. Finally, T extends the registration context as in Line 21, and outputs it together with a response message m_{rrsp}, as defined in Line 19. The agreed content includes the server identifier id, the hash value h, the credential identifier cid, the counter n, the list pkCP, the public key vk, the signature scheme DS, and the user presence UP and verification UV conditions. The session identifier is the tuple of the hash of server identifier id, the credential identifier cid, and the counter n.
- $\mathsf{rVrfy}(\pi_S^i, m_{\mathsf{rcl}}, m_{\mathsf{rrsp}})$: The server S parses the client message m_{rcl} and the response message m_{rrsp} and executes a few checks as in Line 27. It outputs abort and decision $d = 0$ if any check fails. Otherwise, S sets the execution state to $\mathsf{accepted}$. Finally, S extends the registration context as in Line 30 and outputs it together with decision $d = 1$. The agreed content and the session identifier are defined as the ones in the rRsp algorithm.

$\mathsf{Authenticate} = (\mathsf{aChall}, \mathsf{aCom}, \mathsf{aRsp}, \mathsf{aVrfy})$ is defined next.

- $\mathsf{aChall}(\pi_S^i, \mathsf{tb}, UV)$: The server S samples a random challenge nonce $\pi_S^i.\mathsf{ch}$ and initializes its token binding state $\pi_S^i.\mathsf{tb}$ and user condition $\pi_S^i.UV$. Finally, S sets π_S^i to $\mathsf{running}$ and outputs a challenge message, see Line 36.
- $\mathsf{aCom}(\mathsf{id}_S, m_{\mathsf{ach}}, \mathsf{tb})$: The client parses m_{ach} into an identifier id, a challenge nonce ch, and user verification condition UV. Next, the client aborts if $\mathsf{id} \neq \mathsf{id}_S$. Otherwise, the client sets the user presence condition UP to true and compute the hash h of the client message m_{acl}, which is defined in Line 41. Finally, the client outputs the client message m_{acl} and command message m_{acom}, see Line 43.
- $\mathsf{aRsp}(\pi_T^j, \mathsf{rc}_T, m_{\mathsf{acom}})$: The token T first parses the command message m_{acom} into a server identifier id, a hash value h, and user presence and user verification conditions UP and UV. Next, T checks whether the corresponding registration context exists and whether

it can satisfy the user verification requirement. T aborts if either of the above steps fails. Then, T increments the counter $\mathsf{rc}_T[\mathsf{id}].n$ by 1 and defines the associated data ad that includes the hash of id, the counter $\mathsf{rc}_T[\mathsf{id}].n$, and the conditions UP and UV, followed by computing a signature σ on ad and h using the signing key $\mathsf{rc}_T[\mathsf{id}].sk$. Finally, T sets its execution state to $\mathsf{accepted}$ and outputs the response message m_{arsp} defined in Line 52 along with rc_T. The agreed context is defined as the tuple of the server identifier id, the value h, the counter $\mathsf{rc}_T[\mathsf{id}].n$, and the user conditions UV and UP. The session identifier is defined as the tuple of the hash of the server identifier id, the credential identifier $\mathsf{rc}_T[\mathsf{id}].\mathsf{cid}$, the hash value h, and the counter n.

- $\mathsf{aVrfy}(\pi_S^i, \mathsf{rc}_S, m_{\mathsf{acl}}, m_{\mathsf{arsp}})$: The server S parses the client message m_{acl} and the response message m_{arsp} and executes checks as in Line 61 if the corresponding registration context exists. It aborts and produces decision $d = 0$ if any check fails. Otherwise, S updates the counter in the registration context and sets the execution state to $\mathsf{accepted}$ and outputs rc_S together with decision $d = 1$. The agreed context and the session identifier are the same as in aRsp.

5.4.4 Security Experiment for ePIA

The desired security property is that a server accepts an authentication response if and only if it was generated by a unique honest partnered token session. We capture it by our auth security experiment in Figure 5.5.

Threat Model To closely capture the official security statement[4], we assume that all communication channels in the registration phase are authenticated. In contrast, there are no security assumptions on the communication channels between token, client, and server in the authentication phase. We assume that the users always provide the user presence or user verification confirmation when it is required and leave the users implicit in the security model. We assume the identifier id_S of each server S is unique. Unlike [20], we do *not* assume tokens to be "tamper-proof", i.e., the attacker is allowed to corrupt locally stored registration contexts.

Oracles During the game execution the attacker $\mathcal{A}$ can create new servers and tokens through the oracles NewS and NewTPlA. In particular, the attacker can customize the concrete setting of the created parties, i.e., the supported signature list of the server and whether the token supports user verification. By invoking the Register oracle, $\mathcal{A}$ is able to eavesdrop on honest registrations between servers and tokens of its choice. Moreover, via the oracles Challenge, Response and Complete, $\mathcal{A}$ can actively interfere during authentication. Note that sessions which have accepted or rejected can no longer be

[4]"Under the assumption that a registration ceremony is completed securely, and that the authenticator maintains confidentiality of the credential private key, subsequent authentication ceremonies using that public key credential are resistant to man-in-the-middle attacks" [107, Section 13.4.4]

queried. Furthermore, the attacker $\mathcal{A}$ can also query the $\textsc{Corrupt}$ oracle to reveal a token's registration context related to a server.

Session Partnering Partnering identifies token and server sessions that are successfully communicating with each other as expected, and is encoded through matching session identifiers. More precisely, we say a server session π_S^i *partners with a token session* π_T^j if and only if $\pi_S^i.\mathsf{sid} = \pi_T^j.\mathsf{sid} \neq \perp$. We say a server session π_S^i *partners with a token* T if it partners with one of T's sessions. We say a token T is the *registration partner* of a server S, if the registration context of T at S has been set, i.e., $\mathsf{rc}_T[\mathsf{id}_S] \neq \perp$.

Winning Conditions We call a server session a *test session* if it accepts a response message. We say that the secure passwordless authentication for an ePIA holds if for all test sessions π_S^i none of the following winning conditions holds:

1. the non-$\perp$ session identifiers of two token sessions collide.
2. the non-$\perp$ session identifiers of two server sessions collide.
3. π_S^i does not partner with T and $\textsc{Corrupt}(S, T)$ was not queried (i.e., the registration context of T at S has not been revealed), where T is any registration partner of S.
4. the agreed contents of a pair of partnered server session $\pi_{S'}^{i'}$ and token sessions $\pi_{T'}^{j'}$ are distinct and $\textsc{Corrupt}(S', T')$ has not been queried.

Definition 48 (Secure passwordless authentication (auth) for ePIA)**.** *Let* $\mathsf{Compl} \in \{\mathsf{PPT}, \mathsf{QPT}\}$*. Let* $\mathsf{ePIA} = (\mathsf{Register}, \mathsf{Authenticate})$ *be an extended passwordless authentication protocol. We say that* ePIA *provides* secure passwordless authentication*, or* auth *for short, if for all* Compl *attackers* $\mathcal{A}$ *the advantage*

$$\mathsf{Adv}_{\mathsf{ePIA}}^{\mathsf{auth}}(\mathcal{A}) := \Pr\left[\mathrm{Expr}_{\mathsf{ePIA}}^{\mathsf{auth}}(\mathcal{A}) = 1\right]$$

in winning the game $\mathrm{Expr}_{\mathsf{ePIA}}^{\mathsf{auth}}$ *defined in Figure 5.5 is negligible in the implicit security parameter* λ*.*

Conversely, we say a Compl *attacker* $\mathcal{A}$ *breaks the secure passwordless authentication of* ePIA *for some test session* π*, if* $\mathcal{A}$ *wins* $\mathrm{Expr}_{\mathsf{ePIA}}^{\mathsf{auth}}$ *game via* π*.*

In the following theorem, we show that WebAuthn 2 satisfies the defined security property auth. We sketch the proof here and give the full proof in Section 5.9.3.

Theorem 21 (PPT/QPT security of WebAuthn 2)**.** *Let* $\mathsf{Compl} \in \{\mathsf{PPT}, \mathsf{QPT}\}$*. Let* $\mathsf{ePIA} = (\mathsf{Register}, \mathsf{Authenticate})$ *denote the WebAuthn 2 protocol depicted in Figure 5.4. Assume that the underlying function* H *is* $\epsilon_{\mathsf{H}}^{\mathsf{coll\text{-}res}}$*-collision resistant. If there exists a* Compl *attacker* $\mathcal{A}$ *that breaks the secure passwordless authentication of* ePIA *for a test session*

```
$\mathrm{Expr}^{\mathsf{auth}}_{\mathsf{ePIA}}(\mathcal{A})$:
  $\mathcal{L}_{\mathsf{frsh}} \leftarrow \emptyset$
  $\mathsf{win\text{-}auth} \leftarrow 0$
  $() \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}()$
  return $\mathsf{win\text{-}auth}$
```

```
$\mathsf{regPartner}(S)$:
  if $\exists T$ such that $\mathsf{rc}_T[\mathsf{id}_S] \neq \perp$
    return $T$
  return $\perp$
```

```
$\mathsf{Win\text{-}auth}(S, i)$:
  if $\exists (T_1, j_1), (T_2, j_2)$ such that $(T_1, j_1) \neq (T_2, j_2)$ and $\pi^{j_1}_{T_1}.\mathsf{sid} = \pi^{j_2}_{T_2}.\mathsf{sid} \neq \perp$ : return 1
  if $\exists (S_1, i_1), (S_2, i_2)$ such that $(S_1, i_1) \neq (S_2, i_2)$ and $\pi^{i_1}_{S_1}.\mathsf{sid} = \pi^{i_2}_{S_2}.\mathsf{sid} \neq \perp$ : return 1
  $T \leftarrow \mathsf{regPartner}(S)$
  if $(S, T) \in \mathcal{L}_{\mathsf{frsh}}$ and $\neg\exists j$ such that $\pi^i_S.\mathsf{sid} = \pi^j_T.\mathsf{sid}$: return 1
  if $\exists (S', i'), (T', j')$ such that $\pi^{i'}_{S'}.\mathsf{sid} = \pi^{j'}_{T'}.\mathsf{sid} \neq \perp$ and $(S', T') \in \mathcal{L}_{\mathsf{frsh}}$ and $\pi^{i'}_{S'}.\mathsf{agCon} \neq$
      $\pi^{j'}_{T'}.\mathsf{agCon}$: return 1
  return 0
```

```
$\textsc{Register}((S, i), (T, j), \mathsf{tb}, UV)$:
  if $\mathsf{pkCP}_S = \perp$ or $\mathsf{suppUV}_T = \perp$ or $\pi^i_S \neq \perp$ or $\pi^j_T \neq \perp$ or $\mathsf{rc}_T[S] \neq \perp$
    return $\perp$
  $\pi^i_S.\mathsf{pkCP} \leftarrow \mathsf{pkCP}_S$
  $\pi^j_T.\mathsf{suppUV} \leftarrow \mathsf{suppUV}_T$
  $m_{\mathsf{rch}} \xleftarrow{\$} \mathsf{rChall}(\pi^i_S, \mathsf{tb}, UV)$
  $(m_{\mathsf{rcom}}, m_{\mathsf{rcl}}) \leftarrow \mathsf{rCom}(\mathsf{id}_S, m_{\mathsf{rch}}, \mathsf{tb})$
  $(m_{\mathsf{rrsp}}, \mathsf{rc}_T) \xleftarrow{\$} \mathsf{rRsp}(\pi^j_T, m_{\mathsf{rcom}})$
  $(\mathsf{rc}_S, d) \xleftarrow{\$} \mathsf{rVrfy}(\pi^i_S, m_{\mathsf{rcl}}, m_{\mathsf{rrsp}})$
  $\mathcal{L}_{\mathsf{frsh}} \leftarrow \mathcal{L}_{\mathsf{frsh}} \cup \{(S, T)\}$
  return $(m_{\mathsf{rch}}, m_{\mathsf{rcl}}, m_{\mathsf{rcom}}, m_{\mathsf{rrsp}}, d)$
```

```
$\textsc{NewS}(S, \mathsf{pkCP})$:
  if $\mathsf{pkCP}_S \neq \perp$
    return
  $\mathsf{pkCP}_S \leftarrow \mathsf{pkCP}$
  return
```

```
$\textsc{NewTPla}(T, \mathsf{suppUV})$:
  if $\mathsf{suppUV}_T \neq \perp$
    return
  $\mathsf{suppUV}_T \leftarrow \mathsf{suppUV}$
  return
```

```
$\textsc{Challenge}((S, i), \mathsf{tb}, UV)$:
  if $\mathsf{pkCP}_S = \perp$ or $\pi^i_S \neq \perp$
    return $\perp$
  $\pi^i_S.\mathsf{pkCP} \leftarrow \mathsf{pkCP}_S$
  $m_{\mathsf{ach}} \leftarrow \mathsf{aChall}(\pi^i_S, \mathsf{tb}, UV)$
  return $m_{\mathsf{ach}}$
```

```
$\textsc{Response}((T, j), m_{\mathsf{acom}})$:
  if $\mathsf{suppUV}_T = \perp$ or $\pi^j_T \neq \perp$
    return $\perp$
  $\pi^j_T.\mathsf{suppUV} \leftarrow \mathsf{suppUV}_T$
  $(m_{\mathsf{arsp}}, \mathsf{rc}_T) \xleftarrow{\$} \mathsf{aRsp}(\pi^j_T, \mathsf{rc}_T, m_{\mathsf{acom}})$
  return $m_{\mathsf{arsp}}$
```

```
$\textsc{Complete}((S, i), m_{\mathsf{acl}}, m_{\mathsf{arsp}})$:
  if $\pi^i_S = \perp$ or $\pi^i_S.\mathsf{st}_{\mathsf{exe}} \neq \mathsf{running}$
    return $\perp$
  $(\mathsf{rc}_S, d) \xleftarrow{\$} \mathsf{aVrfy}(\pi^i_S, \mathsf{rc}_S, m_{\mathsf{acl}}, m_{\mathsf{arsp}})$
  if $d = 1$
    $\mathsf{win\text{-}auth} \leftarrow \mathsf{Win\text{-}auth}(S, i)$
  return $d$
```

```
$\textsc{Corrupt}(S, T)$:
  if $\mathsf{rc}_T[S] = \perp$
    return $\perp$
  $\mathcal{L}_{\mathsf{frsh}} \leftarrow \mathcal{L}_{\mathsf{frsh}} \setminus \{(S, T)\}$
  return $\mathsf{rc}_T[S]$
```

Figure 5.5: auth security experiment for extended Passwordless Authentication Protocols ePIA = (Register, Authenticate), where $\mathcal{O}$ = {NewS, NewTPla, Corrupt, Register, Challenge, Response, Complete} and Compl ∈ {PPT, QPT}. We highlight the difference to PIA from [20] in blue. The variables agCon and sid are instance-specific, see Section 5.4.2.

π and the digital signature scheme DS *used in π is $\epsilon_{\mathsf{DS}}^{\mathsf{euf\text{-}cma}}$*-euf-cma *secure against* Compl *attackers, then it holds that*

$$\mathsf{Adv}_{\mathsf{ePIA}}^{\mathsf{auth}}(\mathcal{A}) \leq \binom{q_{\textsc{Register}}}{2} 2^{-\lambda} + \binom{q_{\textsc{Challenge}}}{2} 2^{-\lambda} + \epsilon_{\mathsf{H}}^{\mathsf{coll\text{-}res}} + 2q_{\textsc{Register}}\epsilon_{\mathsf{DS}}^{\mathsf{euf\text{-}cma}}$$

where $q_{\mathcal{O}}$ denotes the number of $\mathcal{A}$'s queries to $\mathcal{O} \in \{\textsc{Register}, \textsc{Challenge}\}$.

Proof Sketch. Notice that the token session identifiers include credential identifiers, which are sampled of length $\geq \lambda$ for different tokens only in the $\textsc{Register}$ queries, and a counter n, which is incremented in each sessions of the same token. The attacker $\mathcal{A}$ cannot win via winning condition in Line 8 except probability $\binom{q_{\textsc{Register}}}{2} 2^{-\lambda}$. Note that the server session identifiers include the hash of server id, which is assumed to be unique for each server. Note also that the server session identifiers in the authentication phases additionally includes the hash of the token binding state tb and challenge nonces ch, which are of length $\geq \lambda$ and sampled only in the $\textsc{Challenge}$ queries. The attacker $\mathcal{A}$ cannot win via winning condition in Line 9 except with probability $\binom{q_{\textsc{Challenge}}}{2} 2^{-\lambda} + \epsilon_{\mathsf{H}}^{\mathsf{coll\text{-}res}}$. Finally, observe that the registration phases are authenticated and that the identifier of each server session in the authentication phases is set only when the corresponding server session accepts a signature, which signs the hash of the unique server id, the counter n, the hash of the client message m_{acl}, UP, and UV. Moreover, there are at most $q_{\textsc{Register}}$ private signing keys in the experiment. The winning conditions in Line 11 and Line 12 indicate that the attacker $\mathcal{A}$ can forge any signature of DS without corrupting the private signing key of any token, which happens with probability at most $2\epsilon_{\mathsf{DS}}^{\mathsf{euf\text{-}cma}}$ for each token and thus in total $2q_{\textsc{Register}}\epsilon_{\mathsf{DS}}^{\mathsf{euf\text{-}cma}}$. □

Theorem 21 shows that no polynomial-time attackers against WebAuthn 2 in the auth experiment can trigger any winning condition, through which the following aspects are captured. Conditions 1 and 2 capture the uniqueness of each session identifiers. i.e., if two sessions are partnered with each other, they are each other's unique partners. Condition 3 encodes the official security statement (see footnote 4). Condition 4 ensures that under the same assumption, the token and server sessions in the subsequent authentication ceremonies using that public key credential must agree on the server identifier id_S, the hash value $\mathsf{H}(\mathsf{ch}, \mathsf{tb})$, the local counter n, and the user presence UP and verification UV conditions. As a corollary, if the underlying hash function H is collision resistant, then the token and server sessions also implicitly agree on the token binding state tb.

5.4.5 Post-Quantum Instantiation of WebAuthn 2

To add the ability to authenticate using PQ or hybrid signature schemes with minimal changes to the WebAuthn 2 protocol, we propose to only extend the supported digital signature list pkCP (encoding an "or" choice) and explicitly allowing hybrid schemes (to encode "and", e.g., for classical and PQ schemes).

Following the WebAuthn 2 specification, the server has the option to include RSASSA–PKCS1–v1_5, RSASSA–PSS [123], or/and ECDSA–P256 [165] in pkCP, see Section 5.2 for an explanation of pkCP. Recall that the auth security of the WebAuthn 2 is proven in the standard model in Theorem 21. Therefore, the auth security for WebAuthn 2 also holds against quantum attackers, assuming that $\epsilon_{\mathsf{H}}^{\mathsf{coll\text{-}res}}$ and $\epsilon_{\mathsf{DS}}^{\mathsf{euf\text{-}cma}}$ are sufficiently small against quantum attackers, i.e., are instantiated with PQ secure algorithms. Instead of accepting only plain PQ signatures schemes, the server could also select hybrid signature schemes for pkCP as below.

Let DS_1 and DS_2 be signature schemes. We write $\mathcal{C}[\mathsf{DS}_1, \mathsf{DS}_2] = (\mathsf{DS}_{\mathcal{C}}.\mathsf{KGen}, \mathsf{DS}_{\mathcal{C}}.\mathsf{Sign}, \mathsf{DS}_{\mathcal{C}}.\mathsf{Vrfy})$ for the hybrid signature schemes constructed from DS_1 and DS_2[5]. $\mathsf{DS}_{\mathcal{C}}.\mathsf{KGen}$ simply returns the concatenation of the two ingredient public and secret keys. Similarly, the signature returned by $\mathsf{DS}_{\mathcal{C}}.\mathsf{Sign}$ is the concatenation of the ingredient signatures over the same message. $\mathsf{DS}_{\mathcal{C}}.\mathsf{Vrfy}$ returns 1 if and only if both ingredient signatures are valid. Otherwise it returns 0. The ingredient schemes could either be instantiated with different PQ (PQ-PQ hybrid), or with one classical and one PQ signature scheme (classical-PQ hybrid). Note that many other combiners exists, such as nested approaches that have been formalized in [51], which are particularly well suited to achieve backwards compatibility in, e.g., X.509 certificates.

In case of WebAuthn 2, backwards compatibility is important as not all authenticators, e.g., USB tokens, can be updated to support new algorithms via software updates. To offer backwards compatibility, the server includes classical algorithms in pkCP as less preferred algorithms and PQ/hybrid schemes with higher preference, e.g., $\mathsf{pkCP} = \{\mathsf{DS}_1 = \mathcal{C}[\mathsf{DS}_2, \mathsf{DS}_3], \mathsf{DS}_2, \mathsf{DS}_3\}$ with $\mathsf{DS}_3 \in$ {RSASSA–PKCS1–v1_5, RSASSA–PSS, ECDSA–P256}. Then, the (honest) token would always choose the more preferred hybrid or PQ algorithms for the PQ security, unless they are not supported.

5.4.6 Stronger Downgrade Protection

Our WebAuthn 2 results in the previous sections assume that the registration phase is authenticated (as in the standard), which means that the supported schemes list cannot be modified, and thus basic scheme downgrade attacks are impossible. On the other end of the spectrum, if an active attacker interferes continuously with *all* phases, we cannot detect or prevent downgrades.

However, there is an intermediate threat model, for which WebAuthn 2 could, but does not, provide downgrade protection. Note that the (ordered) list of the relying party's accepted signature algorithms $\pi_S^i.\mathsf{pkCP}$ is sent in plain from the relying party to the

[5]This description can easily be extended to more than two ingredient schemes.

authenticator via the client (see Figure 5.4). The credential keys are then generated using the first algorithm in the *received* pkCP that is supported by the authenticator, see [107, Section 6.3.2.7.1]. During rVrfy, the relying party checks that the used signature scheme DS is in $\pi_S^i.\mathsf{pkCP}$. Hence, if the communication in the registration phase is not authenticated, an attacker can easily change the list pkCP during transmission to the authenticator. For example, during the PQ transition, ideally security is based on classical and PQ algorithms in a backwards compatible way. While we explain how to achieve backwards compatibility with authenticators that only support classical algorithms in Section 5.4.5, a quantum attacker is able to break RSA or ECDSA might change pkCP such that the authenticator only has the choice between classical algorithms.

Consider an attacker that can forge signatures of one of the accepted and supported algorithms. Moreover, assume this attacker is able to compromise the browser or control the network used during registration but not the ones used for authentication, e.g., in an internet cafe a compromised machine is used for registration but others for authentication. Then tricking the authenticator to choose the vulnerable algorithm (and create a corresponding credential key pair) is beneficial because it allows the attacker to forge authentications later on even if they do not control the network anymore.

If the attacker has permanent control of the machine used for registration and authentication, and can forge signatures of an algorithm that is accepted and supported by the relying party and the authenticator, respectively, this attack cannot be prevented. Moreover, it is impossible to prevent the authenticator being tricked into using a less preferred algorithm without substantial changes to the WebAuthn 2 protocol and the public-key infrastructure within. However, we suggest changes that enable *detecting* such an event with high probability, calling the resulting protocol WebAuthn 2^+, if at least one message without interference of the attacker is sent. We depict the changes as boxed operations in Figure 5.4. Essentially, the idea is to include the hash $\mathsf{h_{CP}}$ of the *received* list of accepted algorithms pkCP' during registration, in the authentication response. The relying party compares $\mathsf{H}(\mathsf{pkCP})$ with $\mathsf{h_{CP}}$ to detect whether authenticator and relying party agree on the list of algorithms. To enable the above changes, both the relying party and the authenticator must store respective lists; we suggest to include them in the registration context.

If an attacker changed the list pkCP during registration in WebAuthn 2^+, the attacker would need to change the value $\mathsf{h_{CP}}$ during every authentication response to avoid detection of the attack. We stress that it would not be sufficient to only reject authentications when such an attack is detected, since the honest authenticator would then be unable to communicate with the relying party due to the disagreement on the list pkCP. Even worse, only those authentication responses in which the attacker successfully switched the value $\mathsf{h_{CP}}$ would be accepted. Thus, the detection of this downgrade attack should

```
$\mathrm{Expr}^{\mathsf{AlgAgree}}_{\mathrm{WebAuthn\ 2^+}}(\mathcal{A})$:
$(S, i, T, j, \mathsf{tb}, UV) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}()$
$m^\star_{\mathsf{ach}} \xleftarrow{\$} \textsc{Challenge}((S,i), \mathsf{tb}, UV)$
$(m^\star_{\mathsf{acom}}, m^\star_{\mathsf{acl}}) \leftarrow \mathsf{aCom}(\mathsf{id}_S, m^\star_{\mathsf{ach}}, \mathsf{tb})$
$m^\star_{\mathsf{arsp}} \xleftarrow{\$} \textsc{Response}((T,j), m^\star_{\mathsf{acom}})$
$d^\star_a \xleftarrow{\$} \textsc{Complete}((S,i), m^\star_{\mathsf{acl}}, m^\star_{\mathsf{arsp}})$
$\mathsf{Supp} \leftarrow$ list of supported algorithms by $T$
$d^\star_{T,S} \leftarrow [\![(\mathsf{pkCP}_S \cap \mathsf{Supp})[1] \neq \mathsf{rc}_T[S].\mathsf{DS}]\!]$
return $[d^\star_{(T,S)} = 1 \wedge d^\star_a = 1]$

$\textsc{rChallenge}((S,i), \mathsf{tb}, UV)$:
if $\mathsf{pkCP}_S = \perp$ or $\pi^i_S \neq \perp$
    return $\perp$
$\pi^i_S.\mathsf{pkCP} \leftarrow \mathsf{pkCP}_S$ //ordered list of accepted algorithms
$m_{\mathsf{rrsp}} \leftarrow \mathsf{rChall}(\pi^i_S, \mathsf{tb}, UV)$
return $m_{\mathsf{rrsp}}$

$\textsc{rResponse}((T,j), m_{\mathsf{rcom}})$:
if $\mathsf{suppUV}_T = \perp$ or $\pi^j_T \neq \perp$
    return $\perp$
$\pi^j_T.\mathsf{suppUV} \leftarrow \mathsf{suppUV}_T$
$(m_{\mathsf{rrsp}}, \mathsf{rc}_T) \xleftarrow{\$} \mathsf{rRsp}(\pi^j_T, m_{\mathsf{rcom}})$
return $m_{\mathsf{rrsp}}$

$\textsc{rComplete}((S,i), m_{\mathsf{rcl}}, m_{\mathsf{rrsp}})$:
if $\mathsf{pkCP}_S = \perp$ or $\pi^i_S = \perp$
or $\pi^i_S.\mathsf{st}_{\mathsf{exe}} \neq \mathsf{running}$ : return $\perp$
$(\mathsf{rc}_S, d) \xleftarrow{\$} \mathsf{rVrfy}(\pi^i_S, m_{\mathsf{rcl}}, m_{\mathsf{rrsp}})$
return $d$
```

Figure 5.6: Game $\mathrm{Expr}^{\mathsf{AlgAgree}}_{\mathrm{WebAuthn\ 2^+}}$ and oracles rChallenge, rResponse, rComplete; note that NewS, NewTPla, Challenge, Response, and Complete are given in Figure 5.5.

trigger deregistering the authenticator by the relying party and notifying the user (ideally out-of-band).

More formally, we say that WebAuthn 2^+ satisfies our property *Algorithm Agreement* (AlgAgree) against $\mathsf{Compl} \in \{\mathsf{PPT}, \mathsf{QPT}\}$ attackers $\mathcal{A}$ if the advantage

$$\mathsf{Adv}^{\mathsf{AlgAgree}}_{\mathrm{WebAuthn\ 2^+}}(\mathcal{A}) := \Pr\left[\mathrm{Expr}^{\mathsf{AlgAgree}}_{\mathrm{WebAuthn\ 2^+}}(\mathcal{A}) = 1\right]$$

in winning the game $\mathrm{Expr}^{\mathsf{AlgAgree}}_{\mathrm{WebAuthn\ 2^+}}$ (defined in Figure 5.6) is negligible in the implicit security parameter λ. We view WebAuthn 2^+ as an instantiation of an ePIA and give the attacker access to the following oracles: rChallenge, rResponse, and rComplete given in Figure 5.6, and NewS, NewTPla, Challenge, Response, and Complete given in Figure 5.5.

The attacker wins the game $\mathrm{Expr}^{\mathsf{AlgAgree}}_{\mathrm{WebAuthn\ 2^+}}$ if the generated key pair is not of the most preferred server's algorithm that is supported by the token (i.e., it is not the first element in the intersection of the supported and the preferred algorithms, see line 7 in Figure 5.6), and honestly generated authentications are always accepted by the server (see line 5 in Figure 5.6). It is important to emphasize that our threat model here is different than the one for Section 5.4.4. Namely, we assume that the communication channels in the registration and authentication phase are unauthenticated with one exception. We assume that there is at least one honest authentication, i.e., during this one authentication the attacker does not actively interfere with the communication between the three parties.

We can show that WebAuthn 2^+ satisfies the above property if H is a collision resistant hash function. The proof sketch is as follows. Assume the attacker $\mathcal{A}$ wins $\mathrm{Expr}^{\mathsf{AlgAgree}}_{\mathrm{WebAuthn\ 2^+}}$

(i.e., $d^\star_{(T,S)} = 1$ and $d^\star_a = 1$). This implies that the attacker is able to successfully register the token T at server S such that the chosen signature algorithm is supported by the token, accepted by the server, and not the most preferred algorithm in the intersection of supported and accepted algorithms. Furthermore, it means that line 60 in Figure 5.4 holds, i.e., that the hash value $\mathsf{h_{CP}}$ over the received list pkCP' (computed and sent by the token) is the same as the hash value $\mathsf{rc}_S[\mathsf{cid}].\mathsf{h_{CP}}$ over the original pkCP. This contradicts the collision-resistance of H, as $\mathsf{pkCP} \neq \mathsf{pkCP}'$.

5.5 CTAP 2.1 and Extended Pin-based Access Control for Authenticator Protocols

In this section, we first define the *extended PIN-based Access Control for Authenticators* (ePACA) protocol following [20] and describe CTAP 2.1 as an ePACA instance. Next, we present a variant of the strong unforgeability with trust-binding (SUF-t′) experiment. Finally, we extend CTAP 2.1 for PQ compatibility and formally prove the SUF-t′ security of the extension.

5.5.1 Extended Pin-based Access Control for Authenticator Protocols

An *extended PIN-based Access Control for Authenticators* protocol $\mathsf{ePACA} = (\mathsf{Reboot}, \mathsf{Setup}, \mathsf{Bind}, \mathsf{Auth}, \mathsf{Validate})$ is an interactive protocol between a client C, an authenticator token T, and a user U, specified by the following algorithms:

$\mathsf{Reboot}(T)$: runs at each power-up of the token T and initializes the inherent state with a mandatory user interaction. This algorithm is expected to be invoked to power up T and initialize the local state before the execution of any other algorithms on T.

$\mathsf{Setup}(T, C, U)$: inputs a token T, a client C, and a user U and outputs the transcript trans. During this interactive sub-protocol, U securely transfers the PIN to T via C. Note that this algorithm is invoked on each token T at most once. We write $\mathsf{trans} \xleftarrow{\$} \mathsf{Setup}(T, C, U)$.

$\mathsf{Bind}(T, C, U)$: During this interactive sub-protocol, the client C is bound to the token T under the confirmation of the user U. This sub-protocol is further divided into two algorithms:

- **$\mathsf{Bind\text{-}C}(C, U, m)$:** inputs a client C, a user U, and an incoming message m and outputs an outgoing message m'. During this algorithm, C processes m under the confirmation from U. We write $m' \xleftarrow{\$} \mathsf{Bind\text{-}C}(C, U, m)$.
- **$\mathsf{Bind\text{-}T}(T, m)$:** inputs a token T and an incoming message m, and outputs an outgoing message m'. We write $m' \xleftarrow{\$} \mathsf{Bind\text{-}T}(T, m)$.

$\mathsf{Auth}(C, M)$: inputs a client C and a command M, and outputs both the command M and its authorization tag t. We write $(M, t) \xleftarrow{\$} \mathsf{Auth}(C, M)$.

$\mathsf{Validate}(T, M, t, d)$: inputs a token T, a command M, an authorization tag t, and a user decision $d \in \{\mathsf{accepted}, \mathsf{rejected}\}$, and outputs $\mathsf{status} \in \{\mathsf{accepted}, \mathsf{rejected}\}$ indicating whether the authorization can be verified or not. We write $\mathsf{status} \xleftarrow{\$} \mathsf{Validate}(T, M, t, d)$.

5.5.2 CTAP 2.1 is an ePACA protocol

CTAP 2.1 [59] is a substantial change from CTAP 2.0 [60] in terms of generalization and modularity. More concretely, CTAP 2.1 makes use of a generic stateful so-called *Pin/Uv Auth Protocol* $\mathsf{puvProtocol} = (\mathsf{initialize}, \mathsf{regenerate}, \mathsf{resetpuvToken}, \mathsf{getPublicKey}, \mathsf{encapsulate}, \mathsf{decapsulate}, \mathsf{encrypt}, \mathsf{decrypt}, \mathsf{authenticate}, \mathsf{verify})$, which can be instantiated using $\mathsf{puvProtocol}_1$ and $\mathsf{puvProtocol}_2$ from the standard that we depict in Section 5.5.4. Additionally, we here propose a third instantiation $\mathsf{puvProtocol}_3$ that allows for PQ security in Section 5.5.5. Each $\mathsf{puvProtocol}$ has its internal state including a public-private key pair (pk, sk) and a string pt.

Similar to the treatment in Section 5.4, we use π_T^i and π_C^j to denote token T's i-th and client C's j-th instance respectively. In addition, each T has a token-associated state st_T that is shared by all of T's instances. Namely, we have $T = \{\mathsf{st}_T\} \cup \{\pi_T^i\}_i$ and $C = \{\pi_C^j\}_j$. We use pin_U to denote U's unique PIN. In addition, we define the following variables for tokens T or clients C:

$\mathsf{st}_T.\mathsf{version} \in \{2.0, 2.1\}$: denotes the CTAP version.

$\mathsf{st}_T.\mathsf{puvProtocol}$: denotes a stateful Pin/Uv Auth Protocol.

$\mathsf{st}_T.\mathsf{puvProtocolList}$: denotes the list of Pin/Uv Auth Protocol instantiations that T supports.

$\mathsf{st}_T.\mathsf{pinHash} \in \{0,1\}^\star \cup \{\perp\}$: denotes the hash of a user PIN. This variable is expected to be set during Setup.

$\mathsf{st}_T.\mathsf{pinRetries} \in \{0, ..., \mathsf{pinRetriesMax}\}$: denotes the number of remaining tries for clients to deliver a $\mathsf{pinHash}$, where $\mathsf{pinRetriesMax}$ denotes the maximal number of tries.

$\mathsf{st}_T.m \in \{0, ..., 3\}$: denotes the remaining consecutive tries for clients to deliver $\mathsf{pinHash}$.

$\pi_T^i.\mathsf{st}_{\mathsf{exe}}, \pi_C^j.\mathsf{st}_{\mathsf{exe}} \in \{\mathsf{waiting}, \mathsf{bindStart}, \mathsf{bindDone}, \perp\}$: denotes the execution state of a token/client session.

$\pi_T^i.\mathsf{bs}, \pi_C^j.\mathsf{bs} \in \{0,1\}^\star \cup \{\perp\}$: denotes the binding state. This variable is expected to be set during Bind.

$\pi_T^i.\mathsf{sid}, \pi_C^j.\mathsf{sid} \in \{0,1\}^\star \cup \{\perp\}$: denotes the session identifiers; defined as the full transcript of the Bind execution.

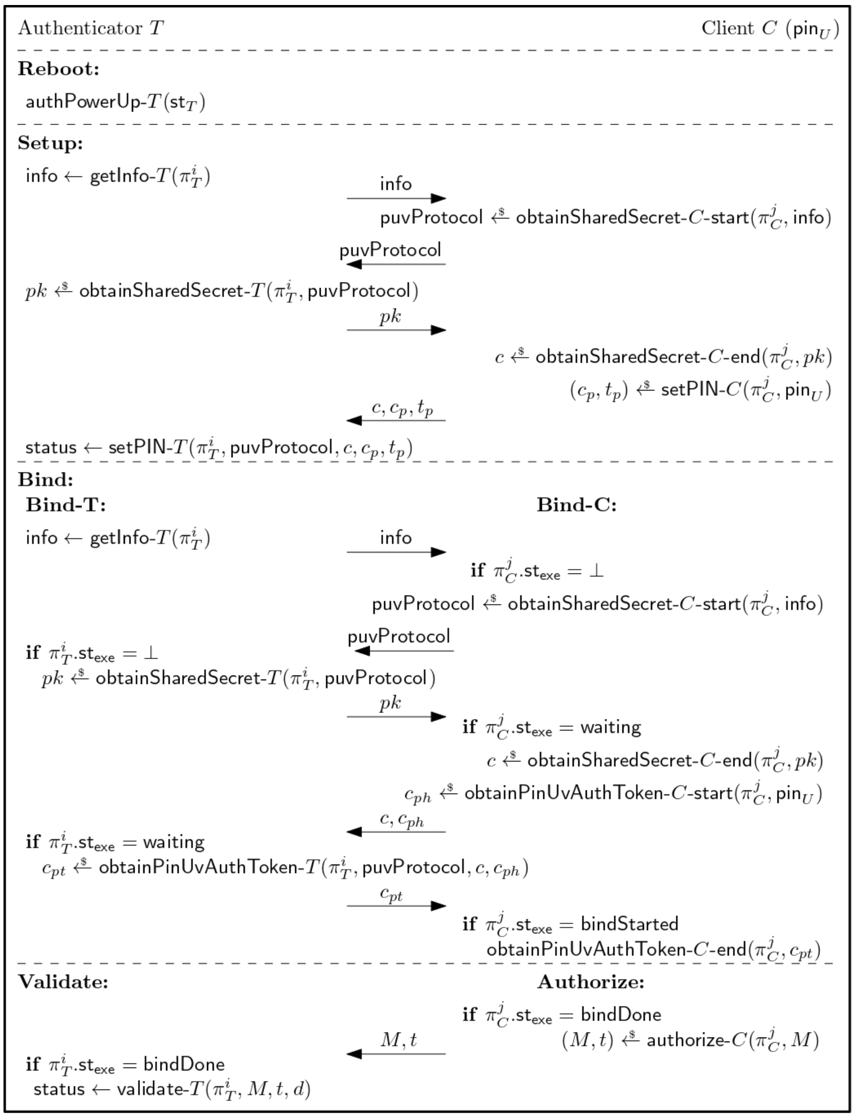

Authenticator T
Client C (pin_U)
Reboot:
$\mathsf{authPowerUp}\text{-}T(\mathsf{st}_T)$
Setup:
$\mathsf{info} \leftarrow \mathsf{getInfo}\text{-}T(\pi_T^i)$
info
$\mathsf{puvProtocol} \xleftarrow{\$} \mathsf{obtainSharedSecret}\text{-}C\text{-}\mathsf{start}(\pi_C^j, \mathsf{info})$
$\mathsf{puvProtocol}$
$pk \xleftarrow{\$} \mathsf{obtainSharedSecret}\text{-}T(\pi_T^i, \mathsf{puvProtocol})$
pk
$c \xleftarrow{\$} \mathsf{obtainSharedSecret}\text{-}C\text{-}\mathsf{end}(\pi_C^j, pk)$
$(c_p, t_p) \xleftarrow{\$} \mathsf{setPIN}\text{-}C(\pi_C^j, \mathsf{pin}_U)$
c, c_p, t_p
$\mathsf{status} \leftarrow \mathsf{setPIN}\text{-}T(\pi_T^i, \mathsf{puvProtocol}, c, c_p, t_p)$
Bind:
Bind-T:
Bind-C:
$\mathsf{info} \leftarrow \mathsf{getInfo}\text{-}T(\pi_T^i)$
info
if $\pi_C^j.\mathsf{st}_{\mathsf{exe}} = \bot$
$\mathsf{puvProtocol} \xleftarrow{\$} \mathsf{obtainSharedSecret}\text{-}C\text{-}\mathsf{start}(\pi_C^j, \mathsf{info})$
$\mathsf{puvProtocol}$
if $\pi_T^i.\mathsf{st}_{\mathsf{exe}} = \bot$
$pk \xleftarrow{\$} \mathsf{obtainSharedSecret}\text{-}T(\pi_T^i, \mathsf{puvProtocol})$
pk
if $\pi_C^j.\mathsf{st}_{\mathsf{exe}} = \mathsf{waiting}$
$c \xleftarrow{\$} \mathsf{obtainSharedSecret}\text{-}C\text{-}\mathsf{end}(\pi_C^j, pk)$
$c_{ph} \xleftarrow{\$} \mathsf{obtainPinUvAuthToken}\text{-}C\text{-}\mathsf{start}(\pi_C^j, \mathsf{pin}_U)$
c, c_{ph}
if $\pi_T^i.\mathsf{st}_{\mathsf{exe}} = \mathsf{waiting}$
$c_{pt} \xleftarrow{\$} \mathsf{obtainPinUvAuthToken}\text{-}T(\pi_T^i, \mathsf{puvProtocol}, c, c_{ph})$
c_{pt}
if $\pi_C^j.\mathsf{st}_{\mathsf{exe}} = \mathsf{bindStarted}$
$\mathsf{obtainPinUvAuthToken}\text{-}C\text{-}\mathsf{end}(\pi_C^j, c_{pt})$
Validate:
Authorize:
if $\pi_C^j.\mathsf{st}_{\mathsf{exe}} = \mathsf{bindDone}$
$(M, t) \xleftarrow{\$} \mathsf{authorize}\text{-}C(\pi_C^j, M)$
M, t
if $\pi_T^i.\mathsf{st}_{\mathsf{exe}} = \mathsf{bindDone}$
$\mathsf{status} \leftarrow \mathsf{validate}\text{-}T(\pi_T^i, M, t, d)$

Figure 5.7: CTAP 2.1 is an ePACA = (Reboot, Setup, Bind, Auth, Validate) protocol. All algorithms are formally defined in Section 5.5.3.

$\pi_C^j.\mathsf{selectedpuvProtocol}$**:** denotes the puvProtocol instantiation chosen by the client.

$\pi_C^j.k \in \{0,1\}^\star \cup \{\bot\}$**:** denotes the shared key with a token.

CTAP2.1 Protocol Intuition. Next, we formalize CTAP 2.1 as an ePACA protocol. Overall CTAP 2.1 includes 12 algorithms[6]. We depict the communication flow of CTAP 2.1 in Figure 5.7. Intuitively, the Reboot algorithm initializes the underlying puvProtocols and resets the remaining consecutive tries $\mathsf{st}_T.m$ to 3.

In the Setup interaction, the token T first outputs its information info, which includes the supported list $\mathsf{st}_T.\mathsf{puvProtocolList}$. Next, the client selects and initializes one

[6]Similar to the treatment in [20], we omit the algorithms for PIN reset and leave it for future work. The suffix -T and -C in the names of algorithms indicates the algorithm executor to be either a token or a client. The suffix -start and -end indicates that this algorithm is the first or the final step in an interactive execution.

π_C^j.selectedpuvProtocol from the received list followed by sending its choice back to T. Then, the token T returns the public key pk of the chosen st_T.puvProtocol. Afterwards, the client runs the encapsulation of its π_C^j.selectedpuvProtocol upon pk for a key $\pi_C^j.k$, which is then used to encrypt and authenticate the PIN pin_U collected from user U, and forwards all derived ciphertexts and tags to T. The token T finally decapsulates the key, followed by verifying the ciphertext, and recovers pin_U. The local st_T.pinHash stores the hash of pin_U and the remaining retries st_T.pinRetries is set to pinRetriesMax.

The Bind interaction is identical to Setup until the client derives the key $\pi_C^j.k$. Then, the client uses $\pi_C^j.k$ to encrypt the hash of the user pin pin_U and sends the ciphertext to T. If st_T.pinRetries does not reach 0, the token decapsulates the key and recovers a pinHash. If the pinHash does not match the hash of the local st_T.pinHash, the underlying st_T.puvProtocol re-generates the public key pk. If the remaining consecutive retries meanwhile arrive at 0, the token is forced to reboot. If the pinHash matches the hash of the local st_T.pinHash, the remaining retries $\mathsf{st}_T.m$ and st_T.pinRetries are reset to their maximal values. The pts of all underlying st_T.puvProtocol are re-sampled. The token finally sets the binding state π_T^i.bs to the pt of the current st_T.puvProtocol, which is then encrypted using the decapsulated key. The client eventually recovers the pt and sets it to π_C^j.bs.

After the negotiation for the binding states, the client can invoke Auth algorithm to authorize command M using its binding state π_T^i.bs. Similarly, the token can invoke the Validate algorithm to verify the authorized command using π_C^j.bs.

Below, we give the detailed description of the 12 algorithms in the following Section 5.5.3 and the official instantiations of Pin/Uv Auth Protocol in Section 5.5.4.

5.5.3 Description of CTAP 2.1 Algorithms

authPowerUp-T: inputs a token state st_T and resets each underlying Pin/Uv Auth Protocol puvProtocol. The counter m for the consecutive tries for binding phase is set to its maximum of 3.

getInfo-T: inputs a token session π_T^i and outputs its version and the list of the supported Pin/Uv Auth Protocol. We write info $\leftarrow$ getInfo-$T(\pi_T^i)$.

obtainSharedSecret-C-start: inputs a client session π_C^j and token information info $=$ (version, puvProtocolList) and aborts if version $= 2.0$. Otherwise, the client session π_C^j selects a Pin/Uv Auth Protocol puvProtocol from the list puvProtocolList and initializes it locally. The execution state of π_C^j is set to waiting. Finally, this algorithm outputs the selected Pin/Uv Auth Protocol puvProtocol. We write puvProtocol $\xleftarrow{\$}$ obtainSharedSecret-C-start(π_C^j, info).

obtainSharedSecret-T: inputs a token session π_T^i and a Pin/Uv Auth Protocol puvProtocol aborts if puvProtocol is not supported by the token T. Otherwise, this algorithm simply outputs the public key of the local instance of puvProtocol. During the execution, the

```
authPowerUp-$T$(st$_T$):
  foreach puvProtocol $\in$ st$_T$.puvProtocolList
    st$_T$.puvProtocol.initialize()
  st$_T$.$m \leftarrow 3$

obtainSharedSecret-$C$-start($\pi_C^j$, info):
  Parse (version, puvProtocolList) $\leftarrow$ info
  if version $= 2.0$
    return $\perp$
  select puvProtocol $\leftarrow$ puvProtocolList
  $\pi_C^j$.selectedpuvProtocol $\leftarrow$ puvProtocol
  $\pi_C^j$.selectedpuvProtocol.initialize()
  $\pi_C^j$.st$_{\text{exe}}$ $\leftarrow$ waiting
  $\pi_C^j$.sid $\leftarrow$ $\pi_C^j$.sid $\|$ info $\|$ puvProtocol
  return puvProtocol

obtainSharedSecret-$T$($\pi_T^i$, puvProtocol) :
  if puvProtocol $\notin$ st$_T$.puvProtocolList
    return $\perp$
  $pk_T \leftarrow$ st$_T$.puvProtocol.getPublicKey()
  $\pi_T^i$.st$_{\text{exe}}$ $\leftarrow$ waiting
  $\pi_T^i$.sid $\leftarrow$ $\pi_T^i$.sid $\|$ puvProtocol $\| pk_T$
  return $pk_T$

obtainSharedSecret-$C$-end($\pi_C^j, pk$) :
  $(c, K) \xleftarrow{\$} \pi_C^j$.selectedpuvProtocol.encapsulate($pk$)
  $\pi_C^j.k \leftarrow K$
  $\pi_C^j$.sid $\leftarrow$ $\pi_C^j$.sid $\| pk \| c$
  return $c$

setPIN-$C$($\pi_C^j$, pin) :
  if pin $\notin \mathcal{PIN}$
    return $\perp$
  $c_p \xleftarrow{\$} \pi_C^j$.selectedpuvProtocol.encrypt($\pi_C^j.k$, pin)
  $t_p \xleftarrow{\$} \pi_C^j$.selectedpuvProtocol.authenticate($\pi_C^j.k, c_p$)
  return $(c_p, t_p)$

setPIN-$T$($\pi_T^i$, puvProtocol, $c, c_p, t_p$):
  if puvProtocol $\notin$ st$_T$.puvProtocolList or st$_T$.pinHash $\neq \perp$
    return $\perp$
  $K \leftarrow$ st$_T$.puvProtocol.decapsulate($c$)
  if $K = \perp$ or st$_T$.puvProtocol.verify($K, c_p, t_p$) = false
    return $\perp$
  pin $\leftarrow$ st$_T$.puvProtocol.decrypt($K, c_p$)
  if pin $\notin \mathcal{PIN}$
    return $\perp$
  st$_T$.pinHash $\leftarrow$ H(pin)
  st$_T$.pinRetries $\leftarrow$ pinRetriesMax
  return accepted

getInfo-$T$($\pi_T^i$):
  info $\leftarrow$ (st$_T$.version, st$_T$.puvProtocolList)
  $\pi_T^i$.sid $\leftarrow$ $\pi_T^i$.sid $\|$ info
  return info

obtainPinUvAuthToken-$C$-start($\pi_C^j$, pin):
  pinHash $\leftarrow$ H(pin)
  $c_{ph} \xleftarrow{\$} \pi_C^j$.selectedpuvProtocol.encrypt($\pi_C^j.k$, pinHash)
  $\pi_C^j$.st$_{\text{exe}}$ $\leftarrow$ bindStart
  $\pi_C^j$.sid $\leftarrow$ $\pi_C^j$.sid $\| c_{ph}$
  return $c_{ph}$

obtainPinUvAuthToken-$T$($\pi_T^i$, puvProtocol, $c, c_{ph}$):
  if puvProtocol $\notin$ st$_T$.puvProtocolList or st$_T$.pinRetries $= 0$
    return ($\perp$, false)
  $K \leftarrow$ st$_T$.puvProtocol.decapsulate($c$)
  if $K = \perp$
    return ($\perp$, false)
  st$_T$.pinRetries $\leftarrow$ st$_T$.pinRetries $- 1$
  pinHash $\leftarrow$ st$_T$.puvProtocol.decrypt($K, c_{ph}$)
  if pinHash $\neq$ st$_T$.pinHash
    st$_T$.puvProtocol.regenerate()
    if st$_T$.$m = 0$
      authPowerUp-$T$(st$_T$)
      return ($\perp$, true)
  st$_T$.$m \leftarrow 3$, st$_T$.pinRetries $\leftarrow$ pinRetriesMax
  foreach puvProtocol$'$ $\in$ st$_T$.puvProtocolList
    st$_T$.puvProtocol$'$.resetpuvToken()
  $\pi_T^i$.bs $\leftarrow$ $\pi_T^i$.puvProtocol.$pt$
  $c_{pt} \xleftarrow{\$}$ st$_T$.puvProtocol.encrypt($K, \pi_T^i$.bs)
  $\pi_T^i$.st$_{\text{exe}}$ $\leftarrow$ bindDone
  $\pi_T^i$.sid $\leftarrow$ $\pi_T^i$.sid $\|$ puvProtocol $\| c \| c_{ph} \| c_{pt} \|$ false
  return ($c_{pt}$, false)

obtainPinUvAuthToken-$C$-end($\pi_C^j, c_{pt}$):
  $\pi_C^j$.bs $\leftarrow$ $\pi_C^j$.selectedpuvProtocol.decrypt($\pi_C^j.k, c_{pt}$)
  $\pi_C^j$.st$_{\text{exe}}$ $\leftarrow$ bindDone
  $\pi_C^j$.sid $\leftarrow$ $\pi_C^j$.sid $\| c$
  return

auth-$C$($\pi_C^j, M$):
  $t \xleftarrow{\$} \pi_C^j$.selectedpuvProtocol.authenticate($\pi_C^j$.bs, $M$)
  return $(M, t)$

validate-$T$($\pi_T^i, M, t, d$):
  if st$_T$.puvProtocol.verify($\pi_T^i$.bs, $M, t$) = true
    return $d$
  return rejected
```

Figure 5.8: CTAP 2.1 is an ePACA = (Reboot, Setup, Bind, Auth, Validate) protocol. The flow of ePACA protocol is given in Figure 5.7.

status of the token session is set to waiting. We write $pk \leftarrow$ obtainSharedSecret-T(π_T^i, puvProtocol).

obtainSharedSecret-C-end: inputs a client session π_C^j and a public key pk. During the execution, the client session produces a shared secret K and a ciphertext c, followed by storing the secret K locally in $\pi_C^j.k$. This algorithm outputs the ciphertext c. We write $c \xleftarrow{\$} \mathsf{obtainSharedSecret}\text{-}C\text{-}\mathsf{end}(\pi_C^j, pk)$.

$\mathsf{setPIN}\text{-}C$: inputs a client session π_C^j and a PIN pin and aborts if pin is not in the PIN domain $\mathcal{PIN}$. Otherwise, π_C^j encrypts this pin and authenticates the encryption using the selected Pin/Uv Auth Protocol and the locally stored shared secret $\pi_C^j.k$. This algorithm outputs the ciphertext c and the authentication tag t. We write $(c, t) \xleftarrow{\$} \mathsf{setPIN}\text{-}C(\pi_C^j, \mathsf{pin})$.

$\mathsf{setPIN}\text{-}T$: inputs a token session π_T^i, a Pin/Uv Auth Protocol puvProtocol, two ciphertexts c and c_p, and an authentication tag t_p. It aborts if puvProtocol is not supported or the local pinHash has been set. Then, the token decapsulates c for a shared secret K and verifies the ciphertext c_p and tag t using K. If K cannot be correctly decapsulated or the verification falls, then this algorithm aborts. If a PIN pin can be correctly decrypted, then the local pinHash $\pi_T^i.\mathsf{pinHash}$ is set to hash of pin and the local counter pinRetries is set to the maximum. Otherwise, this algorithm aborts. In the end, this algorithm outputs a status $\mathsf{status} \in \{\mathsf{accepted}, \mathsf{rejected}\}$ indicating success or failure. [7] We write $\mathsf{status} \leftarrow \mathsf{setPIN}\text{-}T(\pi_T^i, \mathsf{puvProtocol}, c, c_p, t_p)$.

$\mathsf{obtainPinUvAuthToken}\text{-}C\text{-}\mathsf{start}$: inputs a client session π_C^j and a PIN pin. The client session π_C^j computes the hash of pin and encrypts it using the selected Pin/Uv Auth Protocol and the locally stored share secret $\pi_C^j.k$. This algorithm outputs the encryption c. During the execution, the status of the client session is set to bindStart. We write $c \xleftarrow{\$} \mathsf{obtainPinUvAuthToken}\text{-}C\text{-}\mathsf{start}(\pi_C^j, \mathsf{pin})$.

$\mathsf{obtainPinUvAuthToken}\text{-}T$: inputs a token session π_T^i, a Pin/Uv Auth Protocol puvProtocol, and two ciphertexts c and c_{ph}. It aborts if puvProtocol is not supported by T or if the local counter pinRetries is 0. Otherwise, session π_T^i decapsulates c for a key K and aborts if a failure happens during the decapsulation. Then, π_T^i decrements the counter pinRetries by 1 and decrypts c_ph using K for a hash value pinHash. If pinHash matches the locally stored $\mathsf{st}_T.\mathsf{pinHash}$, then the counter m and pinRetries is set to their maximum. Otherwise, the local instance puvProtocol regenerates its key pair. If the counter for the consecutive retries reaches 0, then the token is rebooted. In all cases, the token resets the pts in all Pin/Uv Auth Protocol instances. Then, the session π_T^i sets the pt underlying puvProtocol as the binding state $\pi_T^i.\mathsf{bs}$ and encrypts it using K for a ciphertext c_{pt}. This algorithm outputs c_{pt} and a boolean value calledReboot indicating whether $\mathsf{authPowerUp}\text{-}T$ is invoked or not. After the successful completion, the status of the token session is set to bindDone. We write $(c_{pt}, \mathsf{calledReboot}) \xleftarrow{\$} \mathsf{obtainPinUvAuthToken}\text{-}T(\pi_T^i, \mathsf{puvProtocol}, c, c_{ph})$.

$\mathsf{obtainPinUvAuthToken}\text{-}C\text{-}\mathsf{end}$: inputs a client session π_C^j and a ciphertext c_{pt}. During the execution, the client decrypts the binding state $\pi_C^j.\mathsf{bs}$ from c_{pt} and the status of the client session is set to bindDone.

[7] In practice, the user confirmation is required in this step. Here, we simply assume the user confirmation and omit it in the algorithm.

```
$\mathsf{initialize}_1()$:
    $\mathsf{regenerate}_1()$
    $\mathsf{resetpuvToken}_1()$
$\mathsf{regenerate}_1()$:
    $(pk, sk) \xleftarrow{\$} \mathsf{ECDH.KG}()$
$\mathsf{resetpuvToken}_1()$:
    $pt \xleftarrow{\$} \{0,1\}^{\mu\lambda}$
$\mathsf{encapsulate}_1(pk')$:
    $Z \leftarrow \mathsf{XCoordinateOf}(sk \cdot pk')$
    $K \leftarrow \mathsf{H}_1(Z)$
    $c \leftarrow pk$
    $\textbf{return } (c, K)$
$\mathsf{decapsulate}_1(c)$:
    $Z \leftarrow \mathsf{XCoordinateOf}(sk \cdot c)$
    $K \leftarrow \mathsf{H}_1(Z)$
    $\textbf{return } K$

$\mathsf{getPublicKey}_1()$:
    $\textbf{return } pk$
$\mathsf{encrypt}_1(K, m)$:
    $c \leftarrow \mathsf{SKE}_1.\mathsf{Enc}(K, m)$
    $\textbf{return } c$
$\mathsf{decrypt}_1(K, c)$:
    $m \leftarrow \mathsf{SKE}_1.\mathsf{Dec}(K, c)$
    $\textbf{return } m$
$\mathsf{authenticate}_1(K', m)$:
    $t \leftarrow \mathsf{H}_2(K', m)$
    $\textbf{return } t$
$\mathsf{verify}_1(K', m, t)$:
    $t' \leftarrow \mathsf{H}_2(K', m)$
    $\textbf{return } [\![t = t']\!]$
```

Figure 5.9: The first instantiation of PIN/UV Auth Protocol $\mathsf{puvProtocol}_1$. The operation $\cdot$ denotes scalar multiplication.

$\mathsf{auth}\text{-}C$: inputs a client session π_C^j and a command M. The client session authenticates M using the selected Pin/Uv Auth Protocol and the local binding state for a tag t. This algorithm then outputs M and an authorized tag t[8]. We write $(M, t) \xleftarrow{\$} \mathsf{auth}\text{-}C(\pi_C^j, M)$.

$\mathsf{validate}\text{-}T$: inputs a token session π_T^i, a command M, an authorized tag t, and a user decision $d \in \{\mathsf{accepted}, \mathsf{rejected}\}$, and outputs status $\mathsf{status} = \mathsf{accepted}$ if $d = \mathsf{accepted}$ and M and t can be verified using the binding state $\pi_T^i.\mathsf{bs}$ and the Pin/Uv Auth Protocol, which is specified by the tag t (See Footnote 8); and $\mathsf{rejected}$ otherwise.

5.5.4 Official Instantiations of Pin/Uv Auth Protocol

CTAP 2.1 officially introduces two instantiations of Pin/Uv Auth Protocol $\mathsf{puvProtocol}$, as in Figure 5.9 and Figure 5.10. The first, $\mathsf{puvProtocol}_1$, runs $\mathsf{initialize}_1$ by simply invoking $\mathsf{regenerate}_1$ and $\mathsf{resetpuvToken}_1$, which further samples a public-private key pair from ECDH over curve NIST P-256 and samples a random pt with length $\mu\lambda$ for $\mu \in \{1, 2\}$ and $\lambda = 128$ bits. $\mathsf{getPublicKey}_1$ outputs the internal public key pk. $\mathsf{encapsulate}_1$ computes the key exchange using as input ECDH public key and its internal private key and applies $\mathsf{H}_1 = $ SHA-256 to the x-coordinate of the key exchange result for a shared K, followed by outputting its internal public key and K. $\mathsf{decapsulate}_1$ recovers the shared secret K from ciphertext c using its internal private key sk. $\mathsf{encrypt}_1$ encrypts a message m using SKE_1 and a symmetric key K, where SKE_1 denotes AES-256-CBC encryption using an all-zero initial vector IV. $\mathsf{decrypt}_1$ recovers the message from ciphertext c by using SKE_1 and key

[8]In practice, this authorized tag t also includes information that specifies the index of Pin/Uv Auth Protocol. Here, we omit this.

```
initialize$_2$():
    regenerate$_2$()
    resetpuvToken$_2$()
regenerate$_2$():
    $(pk, sk) \xleftarrow{\$} \mathsf{ECDH.KG}()$
resetpuvToken$_2$():
    $pt \xleftarrow{\$} \{0,1\}^{2\lambda}$
encapsulate$_2$($pk'$):
    $Z \leftarrow \mathsf{XCoordinateOf}(sk \cdot pk')$
    $K_1 \leftarrow \mathsf{H}_3(Z, \text{“CTAP2 HMAC key”})$
    $K_2 \leftarrow \mathsf{H}_3(Z, \text{“CTAP2 AES key”})$
    $K \leftarrow (K_1, K_2)$
    $c \leftarrow pk$
    return $(c, K)$
decapsulate$_2$($c$):
    $Z \leftarrow \mathsf{XCoordinateOf}(sk \cdot c)$
    $K_1 \leftarrow \mathsf{H}_3(Z, \text{“CTAP2 HMAC key”})$
    $K_2 \leftarrow \mathsf{H}_3(Z, \text{“CTAP2 AES key”})$
    $K \leftarrow (K_1, K_2)$
    return $K$
getPublicKey$_2$():
    return $pk$
encrypt$_2$($K, m$):
    Parse $(K_1, K_2) \leftarrow K$ s.t. $|K_1| = 2\lambda$
    $c \leftarrow \mathsf{SKE}_2.\mathsf{Enc}(K_2, m)$
    return $c$
decrypt$_2$($K, c$):
    Parse $(K_1, K_2) \leftarrow K$ s.t. $|K_1| = 2\lambda$
    $m \leftarrow \mathsf{SKE}_2.\mathsf{Dec}(K_2, c)$
    return $m$
authenticate$_2$($K', m$):
    Parse $(K'_1, K'_2) \leftarrow K'$ s.t. $|K'_1| = 2\lambda$
    $t \leftarrow \mathsf{H}_4(K'_1, m)$
    return $t$
verify$_2$($K', m, t$):
    Parse $(K'_1, K'_2) \leftarrow K'$ s.t. $|K'_1| = 2\lambda$
    $t' \leftarrow \mathsf{H}_4(K'_1, m)$
    return $[\![t = t']\!]$
```

Figure 5.10: The second instantiation of PIN/UV Auth Protocol puvProtocol$_2$. The operation $\cdot$ denotes the scalar-multiplication.

K. authenticate$_1$ authenticates a message m using K' by applying H_2 to both, where H_2 runs HMAC-SHA-256 and truncates the result to the first 128 bits. verify$_1$ outputs true if $t = \mathsf{H}_2(K', m)$, and false otherwise[9].

The second instantiation puvProtocol$_2$ runs initialize$_2$, regenerate$_2$, and getPublicKey$_2$ identical to the ones in puvProtocol$_1$. The resetpuvToken$_2$ algorithm outputs a pt with fixed 256 bits length. The algorithm encapsulate$_2$ first computes the x coordinate of the ECDH exchange of input public key and internal private key, denoted by Z, followed by applying H_3 to Z and “CTAP2 HMAC key” for a HMAC key K_1 and to Z and “CTAP2 AES key” for a AES key K_2. Finally, encapsulate$_2$ outputs its internal public key as ciphertext as well as K_1 and K_2. decapsulate$_2$ recovers HMAC key K_1 and AES key K_2 from the input ciphertext c using its internal private key. encrypt$_2$ splits the input K into two sub-keys K_1 and K_2 where K_1 has length of 256 bits. Then, it encrypts a message m using SKE_2 on key K_2, where SKE_2 denotes AES-256-CBC encryption using a randomized initial vector IV. decrypt$_2$ recovers the message m from ciphertext c using the key K_2, where K_2 discards the first 256 bits of K. authenticate$_2$ applies H_4 to key K'_1 and a message m to produce a tag t, where H_4 denotes HMAC-SHA-256 and K'_1 is the first 256 bits of the input K'. verify$_2$ on a key K', a message m, and a tag t, verifies whether the tag t matches $\mathsf{H}_4(K'_1, m)$, where

[9]In practice, if $K' = pt$, then verify$_1$ also outputs fails if pt is not in-use. Note that the usage time of the pt is out of the scope of this paper. We omit this here and in the following verify$_2$ in puvProtocol$_2$.

```
initialize_3():
    regenerate_3()
    resetpuvToken_3()
regenerate_3():
    $(pk_1, sk_1) \xleftarrow{\$} \mathsf{ECDH.KG}()$
    $(pk_2, sk_2) \xleftarrow{\$} \mathsf{KEM.KGen}()$
    $pk \leftarrow (pk_1, pk_2)$
    $sk \leftarrow (sk_1, sk_2)$
encrypt_3(K, m):
    $(K_1, K_2) \leftarrow K$ s.t. $|K_1| = \mu'\lambda$
    $c \leftarrow \mathsf{SKE}_3.\mathsf{Enc}(K_2, m)$
    return $c$
decrypt_3(K, c):
    $(K_1, K_2) \leftarrow K$ s.t. $|K_1| = \mu'\lambda$
    $m \leftarrow \mathsf{SKE}_3.\mathsf{Dec}(K_2, c)$
    return $m$
authenticate_3(K', m):
    $(K'_1, K'_2) \leftarrow K'$ s.t. $|K'_1| = \mu'\lambda$
    $t \leftarrow \mathsf{H}_7(K'_1, m)$
    return $t$
verify_3(K', m, t):
    $(K'_1, K'_2) \leftarrow K'$ s.t. $|K'_1| = \mu'\lambda$
    $t' \leftarrow \mathsf{H}_7(K', m)$
    return $[\![t = t']\!]$
getPublicKey_3():
    return $pk$
resetpuvToken_3():
    $pt \xleftarrow{\$} \{0,1\}^{\mu'\lambda}$
encapsulate_3(pk'):
    $(pk'_1, pk'_2) \leftarrow pk'$
    $(sk_1, sk_2) \leftarrow sk$
    $Z_1 \leftarrow \mathsf{XCoordinateOf}(sk_1 \cdot pk'_1)$
    $(c_2, Z_2) \leftarrow \mathsf{KEM.Encaps}(pk'_2)$
    $Z \leftarrow \mathsf{H}_5(Z_1, Z_2)$
    $K_1 \leftarrow \mathsf{H}_6(Z, \text{"CTAP2 HMAC key"})$
    $K_2 \leftarrow \mathsf{H}_6(Z, \text{"CTAP2 AES key"})$
    $K \leftarrow (K_1, K_2)$
    $c \leftarrow (pk, c_2)$
    return $(c, K)$
decapsulate_3(c):
    Parse $(c_1, c_2) \leftarrow c$
    Parse $(sk_1, sk_2) \leftarrow sk$
    $Z_1 \leftarrow \mathsf{XCoordinateOf}(sk_1 \cdot c_1)$
    $Z_2 \leftarrow \mathsf{KEM.Decaps}(sk_2, c_2)$
    $Z \leftarrow \mathsf{H}_5(Z_1, Z_2)$
    $K_1 \leftarrow \mathsf{H}_6(Z, \text{"CTAP2 HMAC key"})$
    $K_2 \leftarrow \mathsf{H}_6(Z, \text{"CTAP2 AES key"})$
    $K \leftarrow (K_1, K_2)$
    return $K$
```

Figure 5.11: The third instantiation of PIN/UV Auth Protocol $\mathsf{puvProtocol}_3$. The operation $\cdot$ denotes the scalar-multiplication.

K'_1 is the first 256 bits of K'.

5.5.5 Our Post-Quantum Instantiation of CTAP 2.1

We propose a third instantiation of the Pin/Uv Auth Protocol in Figure 5.11 that aims at PQ compatibility in a hybrid manner. Compared to $\mathsf{puvProtocol}_2$, the most important changes made to achieve PQ security are as follows. First, in addition to an ECDH (over curve NIST P-256) key pair, a key pair of a PQ secure KEM is sampled during $\mathsf{regenerate}_3$. Second, the algorithm $\mathsf{encapsulate}_3$ executes both the ECDH key exchange and the encapsulation of the PQ KEM to derive a hybrid ciphertext c and key $K = (K_1, K_2)$. Finally, the algorithm $\mathsf{decapsulate}_3$ correspondingly recovers the hybrid key $K = (K_1, K_2)$ from the ciphertext c.

5.5.6 Security Model of ePACA Protocols

Moving forward, we model the security of ePACA protocols as security experiment $\mathrm{Expr}^{\mathsf{SUF\text{-}t'}}_{\mathsf{ePACA}}$. The security goal is to ensure that a token can only accept a command that has been authorized by a trusted client under user permission.

Trust Model Similarly to [20], we assume "trust-on-first-use", which means that the interactive execution of Setup is authenticated without any active interference of an eavesdropping attacker. Moreover, we assume no active attacks against clients during the interactive execution of Bind, while active attacks against tokens are allowed. More concretely, active attacks against clients are allowed only when the execution state of the clients turns from waiting to bindStart. However, active attacks against tokens are allowed even if the execution state of tokens is still waiting. We further assume that each user holds a unique PIN pin_U that is independently sampled from the domain $\mathcal{PIN}$[10] following some distribution $\mathfrak{D}$ with min-entropy $\alpha_{\mathfrak{D}}$. All tokens are assumed to share a common pinRetriesMax. We assume that each ECDH point is bijective to its x-coordinate.

Experiment-specific Variables Each session π is associated with a variable $\mathsf{isValid} \in \{\mathsf{true}, \mathsf{false}, \perp\}$ that denotes whether the session is still accessible (by users or attackers) or not. Each token session π^i_T is associated with a variable $\mathsf{pinCorr} \in \{\mathsf{true}, \mathsf{false}\}$ that indicates whether the setup user PIN of T has been corrupted.

Oracles The oracles in our security experiment (see Figure 5.12) are defined similarly to the ones in [20]. More concretely, the oracles NewT and NewU create new tokens and users, respectively. In particular, the attacker can customize the token with specific initial data when querying NewT. The Reboot(T) oracle invokes Reboot and marks all previously established sessions of T as invalid. The oracle Setup runs the authenticated interaction of Setup. The oracle Execute captures that the Bind interaction is partially authenticated until the client's execution state is set to bindStart and the remaining interaction of Bind is not authenticated, as the attacker can deliver messages to token and client by Send-Bind-T and Send-Bind-C oracles respectively. The Auth and Validate oracles simulate the Auth and Validate execution of clients and tokens, respectively. Furthermore, querying CorruptUser and Compromise reveals a user's PIN and a client's binding state, respectively. Notably, whenever Reboot or Bind are completed on a token T, we mark all of T's previously established sessions as invalid.

Session Partnering Partnering identifies the sessions of a token T and a client C that successfully completed $\mathsf{Bind}(T, C, U)$ for some user U. We call a token session π^i_T *partnered* with a client session π^j_C if and only if $\pi^i_T.\mathsf{sid} = \pi^j_C.\mathsf{sid} \neq \perp$.

[10]In practice, each PIN must have a maximal length of 63 bytes and a minimal length of four code points (on tokens) or four unicode characters (on client).

```
$\mathrm{Expr}^{\mathsf{SUF\text{-}t'}}_{\mathsf{ePACA}}(\mathcal{A})$:
  $\mathcal{L}_{\textsc{Auth}} \leftarrow \emptyset$
  $\mathsf{win\text{-}SUF\text{-}t'} \leftarrow 0$
  $() \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}()$
  return $\mathsf{win\text{-}SUF\text{-}t'}$
```

```
$\mathsf{bindPartner}(T, i)$:
  if $\exists (C, j)$ s.t. $\pi_T^i.\mathsf{sid} = \pi_C^j.\mathsf{sid}$
    return $(C, j)$
  return $(\perp, \perp)$
```

```
$\mathsf{Win\text{-}SUF\text{-}t'}(T, i, M, t, d)$:
  if $d \neq \mathsf{accepted}$:  return 1
  if $\exists (C_1, j_1), (C_2, j_2)$ s.t. $(C_1, j_1) \neq (C_2, j_2)$ and $\pi_{C_1}^{j_1}.\mathsf{st_{exe}} = \pi_{C_2}^{j_2}.\mathsf{st_{exe}} = \mathsf{bindDone}$ and $\pi_{C_1}^{j_1}.\mathsf{sid} = \pi_{C_2}^{j_2}.\mathsf{sid}$:  return 1
  if $\exists (T_1, i_1), (T_2, i_2)$ s.t. $(T_1, i_1) \neq (T_2, i_2)$ and $\pi_{T_1}^{i_1}.\mathsf{st_{exe}} = \pi_{T_2}^{i_2}.\mathsf{st_{exe}} = \mathsf{bindDone}$ and $\pi_{T_1}^{i_1}.\mathsf{sid} = \pi_{T_2}^{i_2}.\mathsf{sid}$:  return 1
  $(C, j) \leftarrow \mathsf{bindPartner}(T, i)$
  if $(C, j, M, t) \notin \mathcal{L}_{\textsc{Auth}}$
    if $(C, j) = (\perp, \perp)$ or $\pi_C^j.\mathsf{compromised} = \mathsf{false}$
      if $\pi_T^i.\mathsf{pinCorr} = \mathsf{false}$: return 1
  return 0
```

```
$\textsc{NewT}(T, \mathsf{initialData})$:
  if $\mathsf{st}_T \neq \perp$: return $\perp$
  $(\mathsf{version}, \mathsf{puvProtocolList}) \leftarrow \mathsf{initialData}$
  $\mathsf{st}_T.\mathsf{version} \leftarrow \mathsf{version}$
  $\mathsf{st}_T.\mathsf{puvProtocolList} \leftarrow \mathsf{puvProtocolList}$
  $\mathsf{Reboot}(\mathsf{st}_T)$
  return
```

```
$\textsc{NewU}(U)$:
  if $\mathsf{pin}_U = \perp$
    $\mathsf{pin}_U \xleftarrow{\$} \mathcal{PIN}$
  return
```

```
$\textsc{CorruptUser}(U)$:
  $\mathsf{corr}_U \leftarrow \mathsf{true}$
  return $\mathsf{pin}_U$
```

```
$\textsc{Compromise}(C, j)$:
  if $\pi_C^j = \perp$ or $\pi_C^j.\mathsf{st_{exe}} \neq \mathsf{bindDone}$
    return $\perp$
  $\pi_C^j.\mathsf{compromised} \leftarrow \mathsf{true}$
  return $\pi_C^j.\mathsf{bs}$
```

```
$\textsc{Reboot}(T)$:
  if $\mathsf{st}_T = \perp$
    return
  foreach $i$ s.t. $\pi_T^i \neq \perp$
    $\pi_T^i.\mathsf{isValid} \leftarrow \mathsf{false}$
  $\mathsf{Reboot}(\mathsf{st}_T)$
  return
```

```
$\textsc{Setup}(T, i, C, j, U)$:
  if $\mathsf{st}_T = \perp$ or $\pi_T^i \neq \perp$ or $\pi_C^j \neq \perp$ or $\mathsf{pin}_U = \perp$
    return $\perp$
  $\pi_T^i \leftarrow \mathsf{st}_T$
  $\mathsf{trans} \xleftarrow{\$} \mathsf{Setup}(\pi_T^i, \pi_C^j, \mathsf{pin}_U)$
  $\pi_T^i.\mathsf{isValid}, \pi_C^j.\mathsf{isValid} \leftarrow \mathsf{false}$
  $\mathsf{st}_T.\mathsf{user} \leftarrow U$
  return $\mathsf{trans}$
```

```
$\textsc{Send-Bind-T}(T, i, m)$:
  if $\mathsf{st}_T = \perp$ or $\pi_T^i = \perp$ or $\pi_T^i.\mathsf{st_{exe}} \neq \mathsf{waiting}$ or $\pi_T^i.\mathsf{isValid} = \mathsf{false}$
    return $\perp$
  $\pi_T^i.\mathsf{pinCorr} \leftarrow \mathsf{corr}_{\mathsf{st}_T.\mathsf{user}}$
  $m' \xleftarrow{\$} \mathsf{Bind\text{-}T}(\pi_T^i, m)$
  $c_{pt} \,\|\, \mathsf{calledReboot} \leftarrow m'$
  if $\mathsf{calledReboot} = \mathsf{true}$
    foreach $i'$ s.t. $\pi_T^{i'} \neq \perp$
      $\pi_T^{i'}.\mathsf{isValid} \leftarrow \mathsf{false}$
  elseif $\pi_T^i.\mathsf{st_{exe}} = \mathsf{bindDone}$
    foreach $i' \neq i$ and $\pi_T^{i'} \neq \perp$
      $\pi_T^{i'}.\mathsf{isValid} \leftarrow \mathsf{false}$
  return $m'$
```

```
$\textsc{Send-Bind-C}(C, j, m)$:
  if $\pi_C^j = \perp$ or $\pi_C^j.\mathsf{st_{exe}} \neq \mathsf{bindStart}$ or $\pi_C^j.\mathsf{isValid} = \mathsf{false}$
    return $\perp$
  return $\mathsf{Bind\text{-}C}(\pi_C^j, m)$
```

```
$\textsc{Execute}(T, i, C, j, U)$:
  if $\mathsf{st}_T = \perp$ or $\pi_T^i \neq \perp$ or $\pi_C^j \neq \perp$ or $\mathsf{pin}_U = \perp$
    return $\perp$
  $\pi_T^i \leftarrow \mathsf{st}_T$
  $\mathsf{trans}, m_C \leftarrow \perp$
  while $\pi_C^j.\mathsf{st_{exe}} \neq \mathsf{bindStart}$
    $m_T \xleftarrow{\$} \mathsf{Bind\text{-}T}(\pi_T^i, m_C)$
    $m_C \xleftarrow{\$} \mathsf{Bind\text{-}C}(\pi_C^j, U, m_T)$
    $\mathsf{trans} \leftarrow \mathsf{trans} \,\|\, m_T \,\|\, m_C$
  return $\mathsf{trans}$
```

```
$\textsc{Auth}(C, j, M)$:
  if $\pi_C^j = \perp$ or $\pi_C^j.\mathsf{st_{exe}} \neq \mathsf{bindDone}$
    return $\perp$
  $(M, t) \xleftarrow{\$} \mathsf{auth\text{-}}C(\pi_C^j, M)$
  $\mathcal{L}_{\textsc{Auth}} \leftarrow \mathcal{L}_{\textsc{Auth}} \cup \{(C, j, M, t)\}$
  return $(M, t)$
```

```
$\textsc{Validate}(T, i, M, t, d)$:
  if $\pi_T^i = \perp$ or $\pi_T^i.\mathsf{st_{exe}} \neq \mathsf{bindDone}$ or $\pi_T^i.\mathsf{isValid} = \mathsf{false}$
    return $\perp$
  $\mathsf{status} \leftarrow \mathsf{validate}\text{-}T(\pi_T^i, M, t, d)$
  if $\mathsf{status} = \mathsf{accepted}$
    $\mathsf{win\text{-}SUF\text{-}t'} \leftarrow \mathsf{Win\text{-}SUF\text{-}t'}(T, i, M, t, d)$
  return $\mathsf{status}$
```

Figure 5.12: Security experiment for extended PIN-based Access Control Authenticators Protocol for ePACA = (Reboot, Setup, Bind, Auth, Validate), where $\mathcal{O}$ = {NewT, NewU, Compromise, CorruptUser, Reboot, Setup, Execute, Send-Bind-T, Send-Bind-C, Auth, Validate}. We highlight differences to the SUF-t security game from [20] in blue.

Winning Conditions We call a token session *test session* if it accepts an authorized command-tag pair under some user decision. An attacker $\mathcal{A}$ wins $\mathrm{Expr}^{\mathsf{SUF\text{-}t'}}_{\mathsf{ePACA}}$ if there exists a test session π^i_T that accepts an authorized command (M, t) with user decision d and any of the following conditions holds:

1. the user decision $d \neq \mathsf{accepted}$.
2. two distinct client sessions that completed Bind have the same session identifiers.
3. two distinct token sessions that completed Bind have the same session identifiers.
4. (M, t) was not output by any of π^i_T's uncompromised valid partners π^j_C before the corruption of the user PIN that was setup on the token T.

Definition 49 ($\mathsf{SUF\text{-}t'}$ security of ePACA)**.** *Let* $\mathsf{Compl} \in \{\mathsf{PPT}, \mathsf{QPT}\}$*. Let* $\mathsf{ePACA} = (\mathsf{Reboot}, \mathsf{Setup}, \mathsf{Bind}, \mathsf{Auth}, \mathsf{Validate})$ *be an extended PIN-based Access Control for Authenticators protocol. We say that* ePACA *is* strongly unforgeable with trusted binding*, or is* $\mathsf{SUF\text{-}t'}$*-secure for short, if for all* Compl *attackers* $\mathcal{A}$

$$\mathsf{Adv}^{\mathsf{SUF\text{-}t'}}_{\mathsf{ePACA}}(\mathcal{A}) := \Pr[\mathrm{Expr}^{\mathsf{SUF\text{-}t'}}_{\mathsf{ePACA}}(\mathcal{A}) = 1]$$

in winning the game $\mathrm{Expr}^{\mathsf{SUF\text{-}t'}}_{\mathsf{ePACA}}$ *as described in Figure 5.12 is negligible in the implicit security parameter* λ*.*

5.5.7 Security Conclusions for CTAP 2.1

After having defined security for ePACA protocols above, we now present the security statements for CTAP 2.1. We give the full proofs of our two theorems (against PPT and QPT attackers) in Section 5.9.4 and Section 5.9.5.

Our first theorem shows the $\mathsf{SUF\text{-}t'}$ security of CTAP 2.1 against PPT attackers.

Theorem 22 (PPT security of CTAP 2.1)**.** *Let* $\mathsf{ePACA} = (\mathsf{Reboot}, \mathsf{Setup}, \mathsf{Bind}, \mathsf{Auth}, \mathsf{Validate})$ *denote the CTAP 2.1 protocol described in Section 5.5.2. Assume that* ePACA *supports* $\mathsf{puvProtocol}_i$ *for* $i \in \{1, 2, 3\}$*. If the hash function* H *is* $\epsilon^{\mathsf{coll\text{-}res}}_{\mathsf{H}}$ *collision resistant,* $\mathsf{H}_i : \{0,1\}^\star \to \{0,1\}^{l_i}$ *is modeled as independent random oracle for* $i \in \{1, ..., 7\}$*,* SKE_1 *is* $\epsilon^{\mathsf{ind\text{-}1cpa\text{-}H_2}}_{\mathsf{SKE}_1}$-$\mathsf{IND\text{-}1CPA\text{-}H_2}$ *and* $\epsilon^{\mathsf{ind\text{-}1\$pa\text{-}lpc}}_{\mathsf{SKE}_1}$-$\mathsf{IND\text{-}1\$PA\text{-}LPC}$ *secure,* SKE_i *is* $\epsilon^{\mathsf{ind\text{-}1cpa}}_{\mathsf{SKE}_i}$-$\mathsf{IND\text{-}1CPA}$ *and* $\epsilon^{\mathsf{ind\text{-}1\$pa\text{-}lpc}}_{\mathsf{SKE}_i}$-$\mathsf{IND\text{-}1\$PA\text{-}LPC}$ *secure for* $i \in \{2, 3\}$*, and the* sCDH *problem over* ECDH *with prime order* q *is* $\epsilon^{\mathsf{sCDH}}_{\mathsf{ECDH}}$ *hard, then the advantage of any* PPT *attacker* $\mathcal{A}$ *that breaks*

$\mathsf{SUF\text{-}t'}$ *security of* ePACA *is bounded by*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{SUF\text{-}t'}}_{\mathsf{ePACA}}(1^\lambda) \leq &(q_{\textsc{Setup}} + q_{\textsc{Execute}})\epsilon^{\mathsf{sCDH}}_{\mathsf{ECDH}} + \epsilon^{\mathsf{coll\text{-}res}}_{\mathsf{H}} \\ &+ \binom{q_{\textsc{Setup}} + q_{\textsc{Execute}}}{2}(2^{2-\min(l_1,l_3,l_5,l_6)} + 2^{1-q}) \\ &+ q_{\textsc{NewU}} 2^{-\alpha_{\mathfrak{D}}} + \binom{q_{\textsc{Send-Bind-T}}}{2} 2^{-\min(\mu,2,\mu')\lambda} \\ &+ q_{\textsc{Setup}} \max(\epsilon^{\mathsf{ind\text{-}1cpa\text{-}H_2}}_{\mathsf{SKE_1}}, \epsilon^{\mathsf{ind\text{-}1cpa}}_{\mathsf{SKE_2}}, \epsilon^{\mathsf{ind\text{-}1cpa}}_{\mathsf{SKE_3}}) \\ &+ q_{\textsc{Execute}} \max(\epsilon^{\mathsf{ind\text{-}1\$pa\text{-}lpc}}_{\mathsf{SKE_1}}, \epsilon^{\mathsf{ind\text{-}1\$pa\text{-}lpc}}_{\mathsf{SKE_2}}, \epsilon^{\mathsf{ind\text{-}1\$pa\text{-}lpc}}_{\mathsf{SKE_3}}) \\ &+ q_{\textsc{Setup}} \mathsf{pinRetriesMax} 2^{-\alpha_{\mathfrak{D}}} \\ &+ q_{\textsc{Validate}} 2^{-\min(\mu\lambda, 2\lambda, \mu'\lambda, l_2, l_4, l_7)}\end{aligned}$$

where $q_{\mathcal{O}}$ *denotes the number of queries to* $\mathcal{O} = \{\textsc{Setup}, \textsc{Execute}, \textsc{Validate}\}$ *and* q_i *denotes the number of queries to random oracle* H_i *for* $i \in \{1, ..., 7\}$.

Proof Sketch. The proof is divided into the following steps: (1) By the random oracle H_i for $i \in \{1, 3, 5, 6\}$ and the sCDH assumption on the underlying ECDH, all keys K derived from the encapsulation of the underlying $\mathsf{puvProtocol}$ in the $\mathsf{obtainSharedSecret}\text{-}C\text{-}\mathsf{end}$ algorithm, which is only invoked in the $\textsc{Setup}$ and $\textsc{Execute}$ oracles, are distinct except probability $(q_{\textsc{Setup}} + q_{\textsc{Execute}})\epsilon^{\mathsf{sCDH}}_{\mathsf{ECDH}} + \binom{q_{\textsc{Setup}}+q_{\textsc{Execute}}}{2} 2^{2-\min(l_1,l_3,l_5,l_6)}$. (2) By the entropy of the user PIN $\alpha_{\mathfrak{D}}$, none of the user PIN sampled in $\textsc{NewU}$ oracle is predicable except with probability $q_{\textsc{NewU}} 2^{-\alpha_{\mathfrak{D}}}$. (3) By the collision-resistance of H and the entropy of the Diffie-Hellman public keys 2^q and of pt values $2^{\max(\mu,2,\mu')\lambda}$, we have the all $\mathsf{H}(\mathsf{pin})$, Diffie-Hellman public keys, pt values, are respectively distinct except probability in total $\epsilon^{\mathsf{coll\text{-}res}}_{\mathsf{H}} + \binom{q_{\textsc{Setup}+\textsc{Execute}}}{2} 2^{1-q} + \binom{q_{\textsc{Send-Bind-T}}}{2} 2^{-\min(\mu,2,\mu')\lambda}$. (4) By the $\mathsf{IND\text{-}1CPA\text{-}H_2}$ security of $\mathsf{SKE_1}$ and the $\mathsf{IND\text{-}1CPA}$ security of $\mathsf{SKE_2}$ and $\mathsf{SKE_3}$, the pins encrypted by the underlying $\mathsf{puvProtocol}$ in the $\mathsf{setPIN}\text{-}C$ algorithm, which is only invoked in the $\textsc{Setup}$ oracle, are indistinguishable from random except probability $q_{\textsc{Setup}} \max(\epsilon^{\mathsf{ind\text{-}1cpa\text{-}H_2}}_{\mathsf{SKE_1}}, \epsilon^{\mathsf{ind\text{-}1cpa}}_{\mathsf{SKE_2}}, \epsilon^{\mathsf{ind\text{-}1cpa}}_{\mathsf{SKE_3}})$. (5) By the $\mathsf{IND\text{-}1\$PA\text{-}LPC}$ security of SKE_i for $i \in \{1, 2, 3\}$, the $\mathsf{pinHash}$s encrypted by the underlying $\mathsf{puvProtocol}$ in the $\mathsf{obtainPinUvAuthToken}\text{-}C\text{-}\mathsf{start}$ algorithm, which is invoked only in the $\textsc{Execute}$ oracle, are indistinguishable from random except probability $q_{\textsc{Execute}} \max(\epsilon^{\mathsf{ind\text{-}1\$pa\text{-}lpc}}_{\mathsf{SKE_1}}, \epsilon^{\mathsf{ind\text{-}1\$pa\text{-}lpc}}_{\mathsf{SKE_2}}, \epsilon^{\mathsf{ind\text{-}1\$pa\text{-}lpc}}_{\mathsf{SKE_3}})$.

Finally, the attacker $\mathcal{A}$ cannot trigger the flip of the $\mathsf{win\text{-}SUF\text{-}t'}$ predicate in Figure 5.12 via condition (i) in Line 8, due to the design of CTAP 2.1, see $\mathsf{validate}\text{-}T$ algorithm in CTAP 2.1. (ii) in Line 9, due to the distinction of Diffie-Hellman public keys, (iii) in Line 10, due to the distinction of Diffie-Hellman public keys and pts, (iv) in Line 11-14, since $\mathcal{A}$ obtains no information about pins or pts and can only win by randomly guessing the pin in the $\textsc{Setup}$ oracle maximal $\mathsf{pinRetriesMax}$ times for each token session, or the pt values or the tags t in the $\mathsf{Validate}$ algorithm in the $\textsc{Validate}$ oracle, which happens with probability except $q_{\textsc{Setup}}\mathsf{pinRetriesMax}2^{-\alpha_{\mathfrak{D}}} + q_{\textsc{Validate}} 2^{-\min(\mu\lambda, 2\lambda, \mu'\lambda, l_2, l_4, l_7)}$ in total, modeling H_7 as a random oracle. □

The above theorem proves that CTAP 2.1 only accepts messages under the user's approval, which is captured by winning condition 1. Winning conditions 2 and 3 capture the uniqueness of each session identifiers: if two sessions are partnered with each other, then they are each other's unique partners. Condition 4 ensures the token only accepts the authorization from a client that it binds to if (1) the binding phase is trusted, (2) the binding state (on the client side) is not compromised if available, and (3) the user PIN that sets up the token is not corrupted.

As is to be expected, the above theorem only holds when the token's user PINs have large enough entropy. If a user PIN is predictable, the attacker can perform active attacks and authorize malicious commands towards the token.

Moving to the security guarantees against quantum attackers, we note that the asymmetric cryptographic primitives in $\mathsf{puvProtocol}_1$ and $\mathsf{puvProtocol}_2$ are simply ECDH, which is quantum-vulnerable. Therefore, $\mathcal{A}$ can trivially win $\mathsf{SUF\text{-}t'}$ experiment by selecting the Pin/Uv Auth Protocol in a test session to be $\mathsf{puvProtocol}_1$ or $\mathsf{puvProtocol}_2$. The theorem below suggests the security of the test session if $\mathsf{puvProtocol}_3$ is selected as instantiation.

Theorem 23 (QPT security of CTAP 2.1)**.** *Let* $\mathsf{ePACA} = (\mathsf{Reboot}, \mathsf{Setup}, \mathsf{Bind}, \mathsf{Auth}, \mathsf{Validate})$ *denote the CTAP 2.1 protocol described in Section 5.5.2. Assume that the underlying* H *is* $\epsilon^{\mathsf{coll\text{-}res}}_{\mathsf{H}}$*-collision resistant,* H_5 *is* $\epsilon^{\mathsf{swap}}_{\mathsf{H}_5}$-$\mathsf{swap}$ *secure,* H_i *is* $\epsilon^{\mathsf{prf}}_{\mathsf{H}_i}$-$\mathsf{prf}$ *secure for* $i \in \{6, 7\}$*,* SKE_3 *is* $\epsilon^{\mathsf{ind\text{-}1cpa}}_{\mathsf{SKE}_3}$-$\mathsf{IND\text{-}1CPA}$ *and* $\epsilon^{\mathsf{ind\text{-}1\$pa\text{-}lpc}}_{\mathsf{SKE}_3}$-$\mathsf{IND\text{-}1\$PA\text{-}LPC}$ *secure, and that the* KEM *in* $\mathsf{puvProtocol}_3$ *with public-key entropy* α_{pk} *and ciphertext entropy* α_c *is* $\epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}}$-$\mathsf{IND\text{-}CCA}$ *secure. If there exists a* QPT *attacker* $\mathcal{A}$ *that breaks the* $\mathsf{SUF\text{-}t'}$ *security of* ePACA *for a test session* π *that uses* $\mathsf{puvProtocol}_3$*, then we have that*

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{SUF\text{-}t'}}_{\mathsf{ePACA}}(\mathcal{A}) \leq &(q_{\textsc{Setup}} + q_{\textsc{Execute}})(\epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}} + \epsilon^{\mathsf{swap}}_{\mathsf{H}_5} + \epsilon^{\mathsf{prf}}_{\mathsf{H}_6}) \\
&+\binom{q_{\textsc{Setup}} + q_{\textsc{Execute}}}{2} 2^{1-l_6} + \epsilon^{\mathsf{coll\text{-}res}}_{\mathsf{H}} + q_{\textsc{NewU}} 2^{-\alpha_{\mathfrak{D}}} \\
&+\binom{q_{\textsc{Send-Bind-T}}}{2} 2^{-\mu'\lambda} + \binom{q_{\textsc{Execute}}}{2}(2^{-\alpha_{pk}} + 2^{-\alpha_c}) \\
&+q_{\textsc{Setup}}\epsilon^{\mathsf{ind\text{-}1cpa}}_{\mathsf{SKE}_3} + q_{\textsc{Execute}}\epsilon^{\mathsf{ind\text{-}1\$pa\text{-}lpc}}_{\mathsf{SKE}_3} \\
&+q_{\textsc{Setup}}\mathsf{pinRetriesMax}2^{-\alpha_{\mathfrak{D}}} + \binom{q_{\textsc{Execute}}}{2}(2^{-\alpha_{pk}} + 2^{-\alpha_c}) \\
&+q_{\textsc{Validate}}(2^{-\mu'\lambda} + \epsilon^{\mathsf{prf}}_{\mathsf{H}_7} + 2^{-l_7})
\end{aligned}$$

where $q_{\mathcal{O}}$ *denotes the number of queries to* $\mathcal{O} = \{\textsc{Setup}, \textsc{Execute}, \textsc{Validate}\}$*.*

Proof Sketch. The proof is similar to the one for Theorem 22 and consists of following steps: (1) By the $\mathsf{IND\text{-}CCA}$ security of KEM, the swap security of H_5, and the prf security of H_6, all keys K derived in from the encapsulation of the underlying $\mathsf{puvProtocol}$ in the $\mathsf{obtainSharedSecret}\text{-}C\text{-}\mathsf{end}$ algorithm, which is only invoked in the $\textsc{Setup}$ and $\textsc{Execute}$ oracles, are distinct except probability $(q_{\textsc{Setup}} + q_{\textsc{Execute}})\epsilon^{\mathsf{sCDH}}_{\mathsf{ECDH}} + (q_{\textsc{Setup}} + q_{\textsc{Execute}})(\epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}} +$

$\epsilon_{\mathsf{H}_5}^{\mathsf{swap}} + \epsilon_{\mathsf{H}_6}^{\mathsf{prf}}) + \binom{q_{\textsc{Setup}}+q_{\textsc{Execute}}}{2}2^{1-l_6}$. (2) By the entropy of the user PIN $\alpha_{\mathfrak{D}}$, none of the user PIN sampled in NewU oracle is predicable except with probability $q_{\textsc{NewU}}2^{-\alpha_{\mathfrak{D}}}$. (3) By the collision-resistance of H and the entropy $2^{-\mu'\lambda}$ of pt values sampled in the Send-Bind-T oracle, we have all $\mathsf{H}(\mathsf{pin})$ and pt values respectively distinct except probability in total $\epsilon_{\mathsf{H}}^{\mathsf{coll\text{-}res}} + \binom{q_{\textsc{Send-Bind-T}}}{2}2^{-\mu'\lambda}$. (4) By the IND-1CPA security of SKE_3, the pins encrypted by the underlying $\mathsf{puvProtocol}_3$ in the $\mathsf{setPIN}\text{-}C$ algorithm, which is only invoked in the Setup oracle, are indistinguishable from random except probability $q_{\textsc{Setup}}\epsilon_{\mathsf{SKE}_3}^{\mathsf{ind\text{-}1cpa}}$. (5) By the IND-1\$PA-LPC security of SKE_3, the $\mathsf{pinHash}$s encrypted by the underlying $\mathsf{puvProtocol}_3$ in the $\mathsf{obtainPinUvAuthToken}\text{-}C\text{-}\mathsf{start}$ algorithm, which is invoked only in the Execute oracle, are indistinguishable from random except probability $q_{\textsc{Execute}}\epsilon_{\mathsf{SKE}_3}^{\mathsf{ind\text{-}1\$pa\text{-}lpc}}$.

Finally, the attacker $\mathcal{A}$ cannot trigger the flip of the $\mathsf{win\text{-}SUF\text{-}t'}$ predicate in Figure 5.12 via condition (i) in Line 8, due to the design of CTAP 2.1, see $\mathsf{validate}\text{-}T$ algorithm in CTAP 2.1, (ii) in Line 9, since the collision of KEM public keys or ciphertexts with entropy α_{pk} or α_c happens at most $\binom{q_{\textsc{Execute}}}{2}(2^{-\alpha_{pk}} + 2^{-\alpha_c})$, (iii) in Line 10, due to the pairwise distinct KEM public keys and pts in the tokens' session identifiers, (iv) in Line 11-14, since $\mathcal{A}$ obtains no information about pins or pts and can only win by randomly guessing the pin in the Setup oracle maximal $\mathsf{pinRetriesMax}$ times for each token session, or the pt values or the tags t in the $\mathsf{Validate}$ algorithm in the Validate oracle, which happens with probability except $q_{\textsc{Setup}}\mathsf{pinRetriesMax}2^{-\alpha_{\mathfrak{D}}} + q_{\textsc{Validate}}2^{-\mu'\lambda} + \epsilon_{\mathsf{H}_7}^{\mathsf{prf}} + 2^{-l_7}$ in total, assuming the prf security of the underlying H_7. □

As such, we suggest to add our PQ instantiation $\mathsf{puvProtocol}_3$ of CTAP 2.1 to the specifications. Below, we give the suggestions on concrete candidates for primitives in $\mathsf{puvProtocol}_3$.

Instantiation: We suggest to instantiate the underlying KEM with any Round 3 Finalist nominated by NIST and the SKE_3 with AES-512-CBC with randomized initial vector. The underlying functions $\mathsf{H}_i : \{0,1\}^\star \to \{0,1\}^{l_i}$ for $i \in \{5,6,7\}$ can be instantiated with HMAC-SHA-512. Moreover, we suggest to increase the security parameter from 256 to 512 to against Grover's attack and to achieve 256-bits security against quantum attackers.

5.6 FIDO2 Composition

In this section, we analyze the security of the composition of WebAuthn 2 and CTAP 2.1. To provide a more generalized result, we first define the *user authentication* (ua) security model for the composition of any ePIA and ePACA protocols, which we refer to as ePIA+ePACA. Then, we formally reduce the ua security of ePIA+ePACA to the auth security of the underlying ePIA (see Section 5.4.4) and the SUF-t′ security of the underlying ePACA protocols (see Section 5.5.6). In this section, we respectively use $\bar{\pi}$ and π to denote the ePIA and ePACA session, respectively, to distinguish them clearly.

5.6.1 Security Model of ePIA+ePACA

As before, to define the ua security property, we start with describing the trust model, oracles, and winning conditions. **Trust Model** The trust model for ua covers both the ones for auth and for $\mathsf{SUF\text{-}t'}$. Additionally, we assume a server-to-client authenticated channel, which is in practice guaranteed by a TLS connection. As before, we assume "trust-on-first-use", which means, the Setup phase and the initialization of the Bind phase in ePACA and the $\mathsf{Register}$ phase in ePIA are authenticated.

Oracles During the execution of the ua experiment, the attacker $\mathcal{A}$ has access to all oracles defined in the $\mathsf{SUF\text{-}t'}$ experiment except $\textsc{Auth}$ and $\textsc{Validate}$. Furthermore, $\mathcal{A}$ is allowed to query $\textsc{NewS}$, $\textsc{NewTPla}$, and $\textsc{Corrupt}$ from the auth experiment, in addition to the following oracles:

Register$((S,i),(T,j,j'),(C,k),\mathsf{tb},UV,d)$**:** This oracle simulates the honest registration between server S and token T via client C. This oracle is the same as the one in the auth experiment except that after the invocation of $(m_{\mathsf{rcom}}, m_{\mathsf{rcl}}) \leftarrow \mathsf{rCom}(\mathsf{id}_S, m_{\mathsf{rch}}, \mathsf{tb})$, additionally $(m_{\mathsf{rcom}}, t) \leftarrow \textsc{Auth}(C,k,m_{\mathsf{rcom}})$ and $\mathsf{status} \leftarrow \textsc{Validate}(T,j',m_{\mathsf{rcom}},t,d)$ are queried. Moreover, the game aborts if $\mathsf{status} \neq \mathsf{accepted}$. Here, $\textsc{Auth}$ and $\textsc{Validate}$ are defined in the $\mathsf{SUF\text{-}t'}$ experiment.

Challenge$((S,i),(C,k),\mathsf{tb},UV)$**:** This oracle simulates the process of the server S generating a challenge nonce and sending it to the client C in an authenticated channel. This oracle is the same as in the auth experiment except that after the invocation of $(m_{\mathsf{rcom}}, m_{\mathsf{rcl}}) \leftarrow \mathsf{rCom}(\mathsf{id}_S, m_{\mathsf{rch}}, \mathsf{tb})$ we additionally query $(m_{\mathsf{rcom}}, t) \leftarrow \textsc{Auth}(C,k,m_{\mathsf{rcom}})$ and $\mathsf{status} \leftarrow \textsc{Validate}(T,j',m_{\mathsf{rcom}},t,d)$, and append tag t to the output.

Response$((T,j,j'),m_{\mathsf{acom}},t,d)$**:** This oracle simulates the token receiving messages from a client and producing its response. This oracle is the same as the one defined in the auth experiment except that we additionally query $\mathsf{status} \leftarrow \textsc{Validate}(T,j',m_{\mathsf{rcom}},t,d)$, and abort if $\mathsf{status} \neq \mathsf{accepted}$.

Complete$((S,i),m_{\mathsf{acl}},m_{\mathsf{arsp}})$**:** This oracle simulates the server verifying the response message and the client message. This oracle is the same as in the auth experiment except that the winning predicate is $\mathsf{Win\text{-}ua}$ defined in Figure 5.14.

It is important to note that $\textsc{Auth}$ and $\textsc{Validate}$ (from the $\mathsf{SUF\text{-}t'}$ experiment) are embedded in the $\textsc{Register}$, $\textsc{Challenge}$, and $\textsc{Response}$ oracles in Figure 5.13.

Winning Conditions We say *user authentication* (ua) holds, if all of the following conditions hold when an ePIA server session $\bar{\pi}_S^i$ accepts a client message m_{acl} and a response message m_{arsp}:

1. The non-$\perp$ session identifiers of the ePIA token (resp., server) sessions do not collide with each other, see Line 37 - 40 in Figure 5.14.

```
$\mathrm{Expr}^{\mathsf{ua}}_{\mathsf{ePIA+ePACA}}(\mathcal{A})$:
    $\mathcal{L}_{\mathsf{frsh}}, \mathcal{L}_{\textsc{Auth}} \leftarrow \emptyset$ //as in auth and SUF-t' experiments
    $\mathcal{L}_{\textsc{Register}}, \mathcal{L}_{\textsc{Challenge}}, \mathcal{L}_{\textsc{Response}} \leftarrow \emptyset$
    $\mathsf{win\text{-}ua} \leftarrow \mathsf{false}$
    $() \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}()$
    $\mathbf{return}\ \mathsf{win\text{-}ua}$

$\textsc{Register}((S,i),(T,j,j'),(C,k),\mathsf{tb},UV,d)$:
    $\mathbf{if}\ \mathsf{pkCP}_S = \perp\ \mathbf{or}\ \mathsf{suppUV}_T = \perp\ \mathbf{or}\ \bar{\pi}^i_S \neq \perp\ \mathbf{or}\ \bar{\pi}^j_T \neq \perp\ \mathbf{or}\ \mathsf{rc}_T[S] \neq \perp$: $\mathbf{return}\ \perp$
    $\bar{\pi}^i_S.\mathsf{pkCP} \leftarrow \mathsf{pkCP}_S$, $\bar{\pi}^j_T.\mathsf{suppUV} \leftarrow \mathsf{suppUV}_T$
    $m_{\mathsf{rch}} \xleftarrow{\$} \mathsf{rChall}(\bar{\pi}^i_S, \mathsf{tb}, UV)$
    $(m_{\mathsf{rcom}}, m_{\mathsf{rcl}}) \xleftarrow{\$} \mathsf{rCom}(\mathsf{id}_S, m_{\mathsf{rch}}, \mathsf{tb})$
    $(m_{\mathsf{rcom}}, t) \leftarrow \textsc{Auth}(C, k, m_{\mathsf{rcom}})$
    $\mathsf{status} \leftarrow \textsc{Validate}(T, j', m_{\mathsf{rcom}}, t, d)$
    $\mathbf{if}\ \mathsf{status} \neq \mathsf{accepted}$: $\mathbf{return}\ (m_{\mathsf{rch}}, m_{\mathsf{rcl}}, m_{\mathsf{rcom}}, t, \perp, \perp)$
    $(m_{\mathsf{rrsp}}, \mathsf{rc}_T) \xleftarrow{\$} \mathsf{rRsp}(\bar{\pi}^j_T, m_{\mathsf{rcom}})$
    $(\mathsf{rc}_S, d') \leftarrow \mathsf{rVrfy}(\bar{\pi}^i_S, m_{\mathsf{rcl}}, m_{\mathsf{rrsp}})$
    $\mathcal{L}_{\mathsf{frsh}} \leftarrow \mathcal{L}_{\mathsf{frsh}} \cup (S, T)$
    $\mathcal{L}_{\textsc{Register}} \leftarrow \mathcal{L}_{\textsc{Register}} \cup \{(S, i, T, j, j', C, k, m_{\mathsf{rch}}, m_{\mathsf{rcl}}, m_{\mathsf{rcom}}, t, m_{\mathsf{rrsp}})\}$
    $\mathbf{return}\ (m_{\mathsf{rch}}, m_{\mathsf{rcl}}, m_{\mathsf{rcom}}, t, m_{\mathsf{rrsp}}, d')$
$\textsc{Challenge}((S,i),(C,k),\mathsf{tb},UV)$:
    $\mathbf{if}\ \mathsf{pkCP}_S = \perp\ \mathbf{or}\ \bar{\pi}^i_S \neq \perp$: $\mathbf{return}\ \perp$
    $\bar{\pi}^i_S.\mathsf{pkCP} \leftarrow \mathsf{pkCP}_S$
    $m_{\mathsf{ach}} \xleftarrow{\$} \mathsf{aChall}(\bar{\pi}^i_S, \mathsf{tb}, UV)$
    $(m_{\mathsf{acom}}, m_{\mathsf{acl}}) \leftarrow \mathsf{aCom}(\mathsf{id}_S, m_{\mathsf{ach}}, \mathsf{tb})$
    $(m_{\mathsf{acom}}, t) \leftarrow \textsc{Auth}(C, k, m_{\mathsf{acom}})$
    $\mathcal{L}_{\textsc{Challenge}} \leftarrow \mathcal{L}_{\textsc{Challenge}} \cup \{(S, i, C, k, m_{\mathsf{ach}}, m_{\mathsf{acl}}, m_{\mathsf{acom}}, t)\}$
    $\mathbf{return}\ (m_{\mathsf{ach}}, m_{\mathsf{acl}}, m_{\mathsf{acom}}, t)$
$\textsc{Response}((T,j,j'), m_{\mathsf{acom}}, t, d)$:
    $\mathsf{status} \leftarrow \textsc{Validate}(T, j', m_{\mathsf{acom}}, t, d)$
    $\mathbf{if}\ \mathsf{status} \neq \mathsf{accepted}$: $\mathbf{return}\ \perp$
    $\mathbf{if}\ \mathsf{suppUV}_T = \perp\ \mathbf{or}\ \bar{\pi}^j_T \neq \perp$: $\mathbf{return}\ \perp$
    $\bar{\pi}^j_T.\mathsf{suppUV} \leftarrow \mathsf{suppUV}_T$
    $(m_{\mathsf{arsp}}, \mathsf{rc}_T) \xleftarrow{\$} \mathsf{aRsp}(\bar{\pi}^j_T, \mathsf{rc}_T, m_{\mathsf{acom}})$
    $\mathcal{L}_{\textsc{Response}} \leftarrow \mathcal{L}_{\textsc{Response}} \cup \{(T, j, j', m_{\mathsf{acom}}, t, d, m_{\mathsf{arsp}})\}$
    $\mathbf{return}\ m_{\mathsf{arsp}}$
$\textsc{Complete}((S,i), m_{\mathsf{acl}}, m_{\mathsf{arsp}})$:
    $\mathbf{if}\ \bar{\pi}^i_S = \perp\ \mathbf{or}\ \bar{\pi}^i_S.\mathsf{st}_{\mathsf{exe}} \neq \mathsf{running}$: $\mathbf{return}\ \perp$
    $(\mathsf{rc}_S, d) \xleftarrow{\$} \mathsf{aVrfy}(\bar{\pi}^i_S, \mathsf{rc}_S, m_{\mathsf{acl}}, m_{\mathsf{arsp}})$
    $\mathbf{if}\ d = 1$: $\mathsf{win\text{-}ua} \leftarrow \mathsf{Win\text{-}ua}(S, i)$
    $\mathbf{return}\ d$
```

Figure 5.13: ua security experiment for a ePIA+ePACA protocol. The winning condition Win-ua is defined in Figure 5.14. The Auth and Validate oracles are defined in $\mathrm{Expr}^{\mathsf{SUF\text{-}t'}}_{\mathsf{ePACA,Compl}}$ experiment in Figure 5.12.

```
Win-ua$(S, i)$:
36 //The non-$\perp$ session identifiers of ePIA token (resp. server) sessions do not collide with each other
37 if $\exists(T_1, j_1), (T_2, j_2)$ s.t. $(T_1, j_1) \neq (T_2, j_2)$ and $\pi_{T_1}^{j_1}.\mathsf{sid} = \pi_{T_2}^{j_2}.\mathsf{sid} \neq \perp$
38   return 1
39 if $\exists(S_1, i_1), (S_2, i_2)$ s.t. $(S_1, i_1) \neq (S_2, i_2)$ and $\pi_{S_1}^{i_1}.\mathsf{sid} = \pi_{S_2}^{i_2}.\mathsf{sid} \neq \perp$
40   return 1
41 //In ePIA, the partnered session have the identical agreed content unless the registration context on the token is corrupted
42 if $\exists(S', i'), (T', j')$ s.t. $\bar{\pi}_{S'}^{i'}.\mathsf{sid} = \bar{\pi}_{T'}^{j'}.\mathsf{sid} \neq \perp$ and $(S', T') \in \mathcal{L}_{\mathsf{frsh}}$
     and $\bar{\pi}_{S'}^{i'}.\mathsf{agCon} \neq \bar{\pi}_{T'}^{j'}.\mathsf{agCon}$: return 1
43 //The non-$\perp$ session identifiers of ePACA token (resp. client) sessions that completed Bind algorithm don't collide with each other
44 if $\exists(C_1, k_1), (C_2, k_2)$ s.t. $(C_1, k_1) \neq (C_2, k_2)$ and $\pi_{C_1}^{k_1}.\mathsf{st}_{\mathsf{exe}} =$
     $= \pi_{C_2}^{k_2}.\mathsf{st}_{\mathsf{exe}} = \mathsf{bindDone}$ and $\pi_{C_1}^{j_1}.\mathsf{sid} = \pi_{C_2}^{j_2}.\mathsf{sid}$: return 1
45 if $\exists(T_1, j_1'), (T_2, j_2')$ s.t. $(T_1, j_1') \neq (T_2, j_2')$ and $\pi_{T_1}^{j_1'}.\mathsf{st}_{\mathsf{exe}} =$
     $= \pi_{T_2}^{j_2'}.\mathsf{st}_{\mathsf{exe}} = \mathsf{bindDone}$ and $\pi_{T_1}^{j_1'}.\mathsf{sid} = \pi_{T_2}^{j_2'}.\mathsf{sid}$: return 1
46 //The ePIA or ePACA sessions used in the registration phase must partner with each other.
47 foreach $(S', x, T', y, y', C', z, m_{\mathsf{rch}}, m_{\mathsf{rcl}}, m_{\mathsf{rcom}}, t_{\mathsf{rcom}}, m_{\mathsf{rrsp}}) \in \mathcal{L}_{\text{REGISTER}}$
48   if $\bar{\pi}_{T'}^{y}.\mathsf{sid} \neq \bar{\pi}_{S'}^{x}.\mathsf{sid}$: return 1
49   $(C'', z') \leftarrow \mathsf{bindPartner}(T', y')$
50   if $(C'', z', m_{\mathsf{rcom}}, t_{\mathsf{rcom}}) \notin \mathcal{L}_{\text{AUTH}}$ and $\Big((C'', z') = (\perp, \perp)$
      or $\pi_{C''}^{z'}.\mathsf{compromised} = \mathsf{false}\Big)$ and $\pi_{T'}^{y'}.\mathsf{pinCorr} = \mathsf{false}$: return 1
51 //A response message $m_{\mathsf{arsp}}$ must be output by $T$ that registered with $S$, unless $T$'s registration context of $S$ is corrupted
52 $T \leftarrow \mathsf{regPartner}(S, i)$
53 if $\nexists j$ s.t. $\bar{\pi}_{S}^{i}.\mathsf{sid} = \bar{\pi}_{T}^{j}.\mathsf{sid}$
54   if $(S, T) \in \mathcal{L}_{\mathsf{frsh}}$: return 1
55 elseif $\nexists(j', m_{\mathsf{acom}}, t, d, m_{\mathsf{arsp}})$ s.t. $(T, j, j', m_{\mathsf{acom}}, t, d, m_{\mathsf{arsp}}) \in \mathcal{L}_{\text{RESPONSE}}$
56   return 1
57 //Above $m_{\mathsf{arsp}}$ must be output under user approval
58 elseif $d \neq \mathsf{accepted}$: return 1
59 else
60 //Above $m_{\mathsf{arsp}}$ must be output after above $T$ validates above message-tag pair $(m_{\mathsf{acom}}, t)$, which encodes $m_{\mathsf{ach}}$ output by session $\bar{\pi}_{S}^{i}$
61   $(C, k) \leftarrow \mathsf{bindPartner}(T, j')$
62   if $(C, k, m_{\mathsf{acom}}, t) \notin \mathcal{L}_{\text{AUTH}}$
63     if $(C, k) = (\perp, \perp)$ or $\pi_{C}^{k}.\mathsf{compromised} = \mathsf{false}$
64       if $\pi_{T}^{j'}.\mathsf{pinCorr} = \mathsf{false}$: return 1
65   elseif $\nexists(m_{\mathsf{ach}}, m_{\mathsf{acl}})$ s.t. $(S, i, C, k, m_{\mathsf{ach}}, m_{\mathsf{acl}}, m_{\mathsf{acom}}, t) \in \mathcal{L}_{\text{CHALLENGE}}$
66     return 1
67 return 0
```

Figure 5.14: The Win-ua in ua security experiment for ePIA+ePACA. The regPartner and bindPartner predicates are defined in Figure 5.5 and Figure 5.12, respectively.

2. The partnered token and server sessions must have the identical agreed content unless the registration context on the token is corrupted, see Line 42 in Figure 5.14.
3. The non-$\perp$ session identifiers of the ePACA token (resp., client) sessions that completed

Bind, do not collide with each other, see Line 44 - 45 in Figure 5.14.

4. During registration, the ePIA token and server sessions must partner with each other and the authorized command message and tag must have been output by one of the non-compromised partners of the ePACA token session without corrupting its setup user, see Line 47 - 50 in Figure 5.14.

5. The token T that has been registered with S, must own an ePIA session $\bar{\pi}_T^i$ that is partnered with $\bar{\pi}_S^i$ and produce a response message unless T's registration context of S is corrupted, see Line 52 - 56 in Figure 5.14.

6. The above response message must be produced after an ePACA session $\pi_T^{j'}$ validates some authorized command m_{acom} and tag t with the approval from the user, see Line 58 - 58 in Figure 5.14.

7. The above command m_{acom} and tag t must be authorized by a client ePACA session π_C^k that is partnered with $\pi_T^{j'}$ for some challenge message m_{rch} that has been produced by the ePIA session $\bar{\pi}_S^i$, unless π_C^k is compromised or the PIN that sets up token T has been corrupted, see Line 61 - 66 in Figure 5.14.

Definition 50 (ua security for ePIA+ePACA)**.** *Let* Compl $\in \{\mathsf{PPT}, \mathsf{QPT}\}$*,* ePACA *be an extended PIN-based access control for authenticators protocol, and* ePIA *be an extended passwordless authentication protocol. We say that the composition* ePIA+ePACA *has* user authentication*, or is* ua*-secure for short, if for all* Compl *attackers* $\mathcal{A}$ *the advantage*

$$\mathsf{Adv}^{\mathsf{ua}}_{\mathsf{ePIA+ePACA}}(\mathcal{A}) := \Pr[\mathrm{Expr}^{\mathsf{ua}}_{\mathsf{ePIA+ePACA}}(\mathcal{A}) = 1]$$

in winning the game $\mathrm{Expr}^{\mathsf{ua}}_{\mathsf{ePIA+ePACA}}$ *as described in Figure 5.13 is negligible in the implicit security parameter* λ*.*

We can reduce the security of the ePIA+ePACA protocol to the security of the ePIA and the ePACA protocol as stated in the next theorem. We give the full proof in Section 5.9.6.

Theorem 24 (PPT/QPT security of the composition)**.** *Let* Compl $\in \{\mathsf{PPT}, \mathsf{QPT}\}$*. Let* Σ *denote an* ePIA *protocol and* Π *denote an* ePACA *protocol. If there exists a* Compl *attacker* $\mathcal{A}$ *that breaks the* ua *security of the composition* $\Sigma + \Pi$*, then there must exist* Compl *attackers* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *that respectively break the* auth *security of* Σ *and the* SUF-t′ *security of* Π *such that*

$$\mathsf{Adv}^{\mathsf{ua}}_{\Sigma+\Pi,\mathsf{Compl}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{auth}}_{\Sigma,\mathsf{Compl}}(\mathcal{A}_1) + \mathsf{Adv}^{\mathsf{SUF\text{-}t'}}_{\Pi,\mathsf{Compl}}(\mathcal{A}_2).$$

In particular, the winning condition 1 and 3 capture the uniqueness of each WebAuthn 2 and CTAP 2.1 session identifiers. If two sessions are partnered with each other, then they are each other's unique partners. The winning condition 2 ensures that if the credential private key between the partnered token and server sessions is not corrupted, then both sessions must agree on the server identifier id_S, the $\mathsf{H}(\mathsf{ch}, \mathsf{tb})$ hash of the challenge nonce and the token binding states, the local counter n, and the user presence UP and verification UV conditions. Furthermore, if the underlying hash function H is collision resistant, the token and server sessions also implicitly agree on the token binding state tb. Our theorem proves partnership of WebAuthn 2 sessions in the authenticated registration phase and the resilience of man-in-the-middle attacks against WebAuthn 2 in the authentication phase unless the corruption of the registration context on the token, which are captured by winning conditions 4 and 5. The messages between every partnered token and server session in WebAuthn 2 must be authorized by the client, which is connected to the server over an authenticated channel in WebAuthn 2 and bound to the token in CTAP 2.1 unless the attacker make certain corruptions, which is captured by wining conditions 4, 6, and 7.

5.7 Related Work

The only published in-depth formal analysis of FIDO2 is Barbosa et al. [20], which we address in-depth. We note that a recently released manuscript [103] also analyzes aspects of FIDO2, but their work focuses on WebAuthn's privacy aspects, and introducing the possibility of revocation, notably in the context of cryptocurrency wallets. Our work is essentially orthogonal to [103] in terms of focus, and we consider the newer versions of both sub-protocols.

To provide context for our comparison to [20], we first revisit the largest changes in CTAP 2.1 compared to CTAP 2.0.

5.7.1 Comparison between CTAP 2.0 and CTAP 2.1

Compared to the expired proposed standard of CTAP 2.0 [60], the latest draft review of CTAP 2.1 [59] has a number of differences, mainly from the following four aspects:

1. The definition of CTAP 2.0 is directly based on the concrete primitives such as the Diffie-Hellman key exchange and hash functions, while CTAP 2.1 is based on a so-called "PIN/UV Auth Protocol" abstract scheme, denoted by $\mathsf{puvProtocol}$ for short, which leads CTAP 2.1 to be PQ ready. Up to date, two instantiations of $\mathsf{puvProtocol}$ are officially announced, where CTAP 2.1 instantiated by the $\mathsf{puvProtocol}_1$ is close to CTAP 2.0. In particular, CTAP 2.1 instantiated with our hybrid construction $\mathsf{puvProtocol}_3$ proposed in Section 5.5.5 is provably PQ secure, as proven in Theorem 23.
2. In CTAP 2.0, the binding state that is used for the client's authorization and the token's validation is defined as so-called *pinToken*, which has the length of multiple of 128 bits and can be of unlimited length. In CTAP 2.1, the binding state is defined as so-called *pinUvAuthToken*, the length of which is however fixed: either 128 or 256 bits.

3. In CTAP 2.0, the pinToken is sampled during the reboot phase and then repeatedly re-used until the next invocation of the reboot algorithm. In contrast, the pinUvAuthToken in CTAP 2.1 is one-time – it is re-sampled after every usage. This difference exerts a great influence on the security: While CTAP 2.0 only satisfies UF-t security as proven by Barbosa et al. [20], CTAP 2.1 provably satisfies SUF-t′ security, see Theorem 22.

4. CTAP 2.0 allows tokens and clients to share a pinUvAuthToken only when the users provide their correct pin, which is called *clientPIN method*. Instead, CTAP 2.1 additionally enables users to input their biometric information such as fingerprint if the built-in on-device user verification is physically supported by the token, which is called *built-in user verification method*. Notably, the built-in user verification method is always the preferred option when it is supported by the token. The biometric information is assumed to be unique and unpredictable for each user and is input to the token without any intermediary (therefore, the transmission can be considered to be authenticated). In our model, built-in user verification can be viewed as the simplified CTAP 2.1 using clientPIN method without the transmission of the encryption of pinHash.

5.7.2 Comparison with Barbosa et al. [20]

As mentioned before, our work builds on the first formal FIDO2 analysis in [20], and we compare several aspects.

WebAuthn comparison

1. **Different analysis target**: The analysis of [20] assumes attestation type `Basic` such that "the server is assumed to know the attestation public key that uniquely identifies the authenticator" [20]. However, the token's attestation key pair is generated in the factory and at least 100,000 tokens should share same attestation key pair to ensure privacy ([18, Section 14.4], [107, Section 14.4.1]). Thus, Barbosa et al.'s analysis indeed proves the security of a batch of tokens that share the same attestation key pair instead of a single token. In contrast, we investigate WebAuthn with the default attestation type `None`, and our Theorem 21 also applies to WebAuthn with attestation type `Basic`. In particular, our analysis focuses on the security of each single token rather than a batch of tokens.

2. **Fine-grained abstraction**: Our WebAuthn abstraction is more detailed than [20]. For example, we include the supported signature list pkCP of the server, the optional UV-support of the token, and the token binding state `tb`. Our theorem implies that the server and token ultimately agree on these values, which is crucial for the desired security. Furthermore, the supported schemes list enables us to to exhibit a downgrade attack against WebAuthn and specify a security notion "Algorithm Agreement" for the corresponding protection.

3. **Active interference**: The security model of WebAuthn in [20] seems to allow active interference during the registration. This is true in [20]'s model because it assumes that each token has a unique attestation key pair and the server knows in advance which public key to use for signature verification; yet this is not true in practice by design, as mentioned previously. The official specification [107, Section 13.4.4] clearly acknowledges the MitM attack on registration, contradicting the implication of [20].
4. **Stronger attacker capability**: Barbosa et al. assume the tokens to be tamper-proof, i.e., the attacker is prevented from corrupting the internal state of any token. Our model, instead, includes a corruption oracle that enables an attacker to reveal the private signing key, capturing the real world scenario in which some tokens might be stolen and the private keys compromised.

CTAP comparison

1. **Different analysis target**: Barbosa et al. analyzed CTAP 2.0 [60], while we investigate CTAP 2.1 [59]. As explained in Section 5.7.1, these two versions have numerous differences. Our paper carefully explores the abstraction gaps between CTAP 2.0 and CTAP 2.1.
2. **Improved security model**: We refine Barbosa et al.'s PACA security model. For example, the token binding states may be reset in REBOOT or SEND oracle. However, Barbosa et al. only mark the token sessions invalid in the REBOOT oracle but forgot the ones in the SEND oracle[11]. Furthermore, the PACA definition of invalidity is not suitable for CTAP 2.1, as the previous binding states of a token are reset after not only reboot but also the establishment of a new session. In this work, we define a code-based $\mathsf{SUF\text{-}t'}$ security, which refines and generalizes $\mathsf{SUF\text{-}t}$ security in [20].
3. **Proof gaps**: Although Barbosa et al. proved the security of CTAP 2.0, their proof has several technical gaps. To address this, we base the $\mathsf{SUF\text{-}t'}$ security of CTAP 2.1 on novel assumptions and provide a detailed proof.

The Composition of WebAuthn and CTAP

1. **Different security model**: The security of the composition of WebAuthn 2 and CTAP 2.1 relies on the respective security guarantees. The differences between the syntax and the security models of both WebAuthn and CTAP compared to [20] propagate into a different security model for the composition, and we provide a fully detailed proof.

[11]Recall that [20] defines the invalidity of a session such that "*if a token is rebooted, its binding states got reset and hence become invalid*" [20]

5.8 Limitation and Future Work

While our work covers many core aspects of CTAP and WebAuthn beyond the state-of-the-art, it remains an abstraction. Some of our main current limitations include that we do not yet model some of the new CTAP 2.1 features for enterprise customers, and do not make formal statements about the unlinkability of credentials or other detailed privacy statements. We leave the proof methodology for tighter upper bounds in all theorems in our paper as an open question.

5.9 Full Proofs

5.9.1 Proof of Theorem 19

Proof. The proof is given by a simple reduction. If there exists an attacker $\mathcal{A}$ that breaks $\mathsf{IND\text{-}1\$PA}$ security of SKE, then we can construct another attacker $\mathcal{B}$ that breaks $\mathsf{IND\text{-}CPA}$ security of SKE as follows:

1. $\mathcal{B}$ invokes $\mathcal{A}$.
2. When $\mathcal{A}$ outputs $(m_0^\star, m_1^\star)$, $\mathcal{B}$ returns 0 if $m_0^\star$ and $m_1^\star$ do not have the same length. Otherwise, $\mathcal{B}$ forwards $(m_0^\star, m_1^\star)$ to its $\mathcal{O}_{\mathsf{Enc}}$ oracles, and returns the response to $\mathcal{A}$.
3. Whenever $\mathcal{A}$ queries $\textsc{Rand}$ oracle with input l, $\mathcal{B}$ first samples $m_0', m_1' \xleftarrow{\$} \{0,1\}^l$. Then, $\mathcal{B}$ sends (m_0', m_1') to its $\mathcal{O}_{\mathsf{Enc}}$ oracle and receives response c'. Finally, $\mathcal{B}$ returns (m_0', m_1', c') to $\mathcal{A}$.
4. When $\mathcal{A}$ outputs b', $\mathcal{B}$ also outputs b'.

It is straightforward that $\mathcal{B}$ perfectly simulates $\mathsf{IND\text{-}1\$PA}$ experiment to $\mathcal{A}$ and $\mathcal{B}$ wins if and only if $\mathcal{A}$ wins. Thus, we have that

$$\epsilon_{\mathsf{SKE}}^{\mathsf{ind\text{-}1\$pa}} \leq \epsilon_{\mathsf{SKE}}^{\mathsf{ind\text{-}cpa}}$$

□

5.9.2 Proof of Theorem 20

Proof. The proof is given by a sequence of games. Let Adv_i denote the attacker $\mathcal{A}$'s advantage in winning Game i. It is straightforward that the attacker $\mathcal{A}$ can win only by random guessing if it outputs $m_0^\star = m_1^\star$, which yields the advantage 0. So, in the proof below, we assume $m_0^\star \neq m_1^\star$. Let $i^\star$ denote the smallest index such that the $i^\star$-th block of $m_0^\star$ does not equal the one of $m_1^\star$.

Game 0. This game is identical to the original $\mathsf{IND\text{-}1\$PA\text{-}LPC}$ experiment defined in Definition 47. Thus, we have that

$$\mathsf{Adv}_0 = \epsilon_{\mathsf{SKE}}^{\mathsf{ind\text{-}1\$pa\text{-}lpc}}$$

Game 1. This game is identical to **Game** 0 except the following modification:

1. The challenger $\mathcal{C}$ samples a random invertible permutation $f : \{0,1\}^{f_2(\lambda)} \rightarrow \{0,1\}^{f_2(\lambda)}$ in advance,
2. Whenever $\mathcal{C}$ needs to execute the encryption $\mathsf{SKE.Enc}(K, \cdot)$ on some messages, $\mathcal{C}$ replaces the underlying computation of $\mathsf{F}(K, \cdot)$ by $f(\cdot)$, and $\mathsf{F}^{-1}(K, \cdot)$ by $f^{-1}(\cdot)$.

It is straightforward that if $\mathcal{A}$ can distinguish **Game** 0 and **Game** 1, then there exists an attacker $\mathcal{B}_1$ that can break the prp security of F. Thus, we have that

$$\mathsf{Adv}_0 - \mathsf{Adv}_1 \leq \epsilon_{\mathsf{F}}^{\mathsf{prp}}$$

Game 2. In this game, the challenger aborts and let $\mathcal{A}$ immediately win if $\mathcal{A}$ can query LPC with input c such that $m_0^\star = \mathsf{SKE.Dec}(K, c)$ but $c \neq c^\star$. Let $n \cdot f_2(\lambda)$ denote the length of $m_0^\star$ for $n \geq 1$. We parse $m_0^\star$ into n blocks such that $m_0^\star = x_1^\star \parallel \cdots \parallel x_n^\star$. Similarly, we parse $c^\star = y_0^\star \parallel \cdots \parallel y_n^\star$ and $c = y_0 \parallel \cdots \parallel y_n$. Note that the condition $c \neq c^\star$. We use $j^\star$ to denote the smallest index such that $y_{j^\star} \neq y_{j^\star}^\star$.

We separate the analysis into the following two cases.

If $\mathsf{b} = 0$ In this case, $m_0^\star = \mathsf{SKE.Dec}(K, c) = \mathsf{SKE.Dec}(K, c^\star)$ but $c \neq c^\star$. We first claim that $j^\star = 0$. Suppose that $j^\star > 0$, which means $y_0 = y_0^\star$. Note that $m_0^\star = \mathsf{SKE.Dec}(K, c) = \mathsf{SKE.Dec}(K, c^\star)$, which implies that

$$x_i^\star = y_{i-1} \oplus f^{-1}(y_i) = y_{i-1}^\star \oplus f^{-1}(y_i^\star), \forall i \in \{1, \cdots, n\}$$

In particular,

$$x_1^\star = y_0 \oplus f^{-1}(y_1) = y_0^\star \oplus f^{-1}(y_1^\star)$$

By $y_0 = y_0^\star$, we can observe that $y_1 = f(x_1^\star \oplus y_0) = f(x_1^\star \oplus y_0^\star) = y_1^\star$. Repeating the steps above, we can further observe that $y_i = y_i^\star$ for $i = 1, 2, ..., n$ step by step. This contradicts to our condition $c \neq c^\star$.

Now, we focus on the first two blocks of the ciphertext c and $c^\star$. By the equation above, $m_0^\star = \mathsf{SKE.Dec}(K, c) = \mathsf{SKE.Dec}(K, c^\star)$ in particular implies that $y_1 = f(x_1^\star \oplus y_0) = f(f^{-1}(y_1^\star) \oplus y_0^\star \oplus y_0)$. Recall that f is a random permutation as defined in **Game** 1 and that $\mathcal{A}$ has no access to f or f^{-1}. Unless the permutation f has applied to $f^{-1}(y_1^\star) \oplus y_0^\star \oplus y_0$ in the RAND oracle, which happens with probability at most $q_{\text{RAND}} \lceil \frac{l_{\mathsf{max}}}{f_2(\lambda)} \rceil 2^{-f_2(\lambda)}$, where l_{max} denotes the maximal input of the queries to RAND oracle, the attacker has no information about y_1 and can guess y_1 only by random guessing, which happens at most $2^{-f_2(\lambda)}$ per query. Note that $\mathcal{A}$ can query LPC at most q_{LPC} times, by union bound theorem, we know that the attacker can win by query LPC oracle with probability at most $q_{\text{LPC}} 2^{-f_2(\lambda)} + q_{\text{RAND}} \lceil \frac{l_{\mathsf{max}}}{f_2(\lambda)} \rceil 2^{-f_2(\lambda)}$.

If $\mathsf{b} = 1$ In this case, the attacker needs to forge the ciphertext c that can be decrypted to $m_0^\star$ by himself. In particular, the attacker $\mathcal{A}$ needs to forge the ($i^\star$-1)-th and $i^\star$-th blocks of the ciphertext such that $y_{i^\star} = f(y_{i^\star\text{-}1} \oplus x^\star_{i^\star})$. Unless the permutation has applied to $y_{i^\star\text{-}1} \oplus x^\star_{i^\star}$, which happens with probability at most $q_{\text{RAND}}\lceil \frac{l_{\mathsf{max}}}{f_2(\lambda)} \rceil 2^{-f_2(\lambda)}$, the attacker receives no information about $y_{i^\star}$ and can only randomly guess, which happens with probability $2^{-f_2(\lambda)}$ per query. Note that $\mathcal{A}$ can query LPC at most q_{LPC} times, by union bound theorem, we know that the attacker can win by query LPC oracle with probability at most $q_{\text{LPC}} 2^{-f_2(\lambda)} + q_{\text{RAND}}\lceil \frac{l_{\mathsf{max}}}{f_2(\lambda)} \rceil 2^{-f_2(\lambda)}$.

To sum up, we have that

$$\mathsf{Adv}_1 - \mathsf{Adv}_2 \leq q_{\text{LPC}} 2^{-f_2(\lambda)} + q_{\text{RAND}}\lceil \frac{l_{\mathsf{max}}}{f_2(\lambda)} \rceil 2^{-f_2(\lambda)}$$

Game 3. This game is identical to **Game** 2 except the following modification:

1. Whenever the attacker $\mathcal{A}$ queries LPC with some input c, the challenger $\mathcal{C}$ simply returns 1 if $c = c^\star$, and 0 otherwise.

Recall that we ensure that the attacker $\mathcal{A}$ cannot query LPC with any input c such that $m_0^\star = \mathsf{SKE.Dec}(K, c)$ but $c \neq c^\star$. So, **Game** 2 and **Game** 3 look identical from the attacker's view and we have that

$$\mathsf{Adv}_2 = \mathsf{Adv}_3$$

Game 4. This game is identical to **Game** 3 except the following modification:

1. Whenever $\mathcal{C}$ needs to execute the encryption $\mathsf{SKE.Enc}(K, \cdot)$ on some messages, $\mathcal{C}$ replaces the underlying computation of $f(\cdot)$ by $\mathsf{F}(K, \cdot)$, and $f^{-1}(\cdot)$ by $\mathsf{F}^{-1}(K, \cdot)$.

It is straightforward that if $\mathcal{A}$ can distinguish **Game** 3 and **Game** 4, then there exists an attacker $\mathcal{B}_2$ that can break the prp security of F. Thus, we have that

$$\mathsf{Adv}_3 - \mathsf{Adv}_4 \leq \epsilon_{\mathsf{F}}^{\mathsf{prp}}$$

Final Analysis. In the end, we analyze the attacker $\mathcal{A}$'s advantage in winning **Game** 4 by reduction. Namely, if $\mathcal{A}$ can break **Game** 4, then we can construct an attacker $\mathcal{B}_3$ that breaks $\mathsf{IND\text{-}1\$PA}$ security of $\mathsf{SKE} = \mathsf{CBC}_0$ as follows:

1. $\mathcal{B}_3$ invokes $\mathcal{A}$.
2. When $\mathcal{A}$ outputs $(m_0^\star, m_1^\star)$, $\mathcal{B}_3$ forwards it to its challenger. Later, when $\mathcal{B}_3$ receives $c^\star$ from its challenger, $\mathcal{B}_3$ forwards $c^\star$ to $\mathcal{A}$.
3. When $\mathcal{A}$ queries RAND(l), $\mathcal{B}_3$ forwards this query to its challenger and the response back to $\mathcal{A}$.
4. When $\mathcal{A}$ queries LPC(c), $\mathcal{B}_3$ returns 1 if $c = c^\star$ and 0 otherwise.
5. When $\mathcal{A}$ outputs a bit b', $\mathcal{B}_3$ forwards b' to its challenger.

It is straightforward that $\mathcal{B}_3$ perfectly simulates **Game** 4 to $\mathcal{A}$ and $\mathcal{B}_3$ wins if and only if $\mathcal{A}$ wins. Thus, we have that

$$\mathsf{Adv}_4 \leq \epsilon_{\mathsf{SKE}}^{\mathsf{ind\text{-}1\$pa}}$$

Combing the statements above, the proof is concluded by

$$\epsilon_{\mathsf{SKE}}^{\mathsf{ind\text{-}1\$pa\text{-}lpc}} \leq 2\epsilon_{\mathsf{F}}^{\mathsf{prp}} + q_{\text{LPC}} 2^{-f_2(\lambda)} + q_{\text{RAND}} \lceil \frac{l_{\mathsf{max}}}{f_2(\lambda)} \rceil 2^{-f_2(\lambda)} + \epsilon_{\mathsf{SKE}}^{\mathsf{ind\text{-}1\$pa}}$$

□

5.9.3 Proof of Theorem 21

Proof. The proof is given by a sequence of games. Let Adv_i denotes the advantage of the Compl attacker $\mathcal{A}$ in winning Game i.

Game 0. This game is identical to the original experiment depicted in Figure 5.5. It holds that

$$\mathsf{Adv}_0 = \mathsf{Adv}_{\mathsf{PIA}}^{\mathsf{auth}}(\mathcal{A})$$

Game 1. This game is identical to **Game** 0 except that the game aborts and lets $\mathcal{A}$ immediately win if there exist two credential identifiers that collide with each other. By this, we ensure that all credential identifiers are distinct. Note that cids are only sampled in the token's registration response rRsp algorithm and that the rRsp algorithm is invoked only when the attacker $\mathcal{A}$ queries REGISTER oracle, which happens at most q_{REGISTER} times, there are maximal $\binom{q_{\text{REGISTER}}}{2}$ pairs of cids. Note also that each cid is independently sampled from the set $\{0,1\}^{\geq\lambda}$. The collision of cids happens with probability $\binom{q_{\text{REGISTER}}}{2}2^{-\lambda}$. Hence, it holds that

$$\mathsf{Adv}_0 - \mathsf{Adv}_1 \leq \binom{q_{\text{REGISTER}}}{2}2^{-\lambda}$$

Game 2. This game is identical to **Game** 1 except that the challenger aborts the game and let $\mathcal{A}$ immediately win if there are two challenge nonces ch during the authentication phases that collide. By this, we ensure that all challenges nonce ch sampled in the authentication phases are distinct. Note that chs in the authentication phase are only sampled in the server's authentication challenge aChall algorithm and that the aChall algorithm is invoked only when the attacker $\mathcal{A}$ queries CHALLENGE oracle, which happens at most $q_{\text{CHALLENGE}}$ times. There are maximal $\binom{q_{\text{CHALLENGE}}}{2} = \frac{q_{\text{CHALLENGE}}(q_{\text{CHALLENGE}}-1)}{2}$ pairs of chs in the authentication phases. Note also that each ch is independently sampled in the set $\{0,1\}^{\geq\lambda}$. The collision of such chs happens with probability at most $\binom{q_{\text{CHALLENGE}}}{2}2^{-\lambda}$. Hence, it holds that

$$\mathsf{Adv}_1 - \mathsf{Adv}_2 \leq \binom{q_{\text{CHALLENGE}}}{2}2^{-\lambda}$$

Game 3. This game is identical to **Game** 2 except that the game aborts and the attacker $\mathcal{A}$ immediately wins if there exist two hash values $\mathsf{H}(x_1) = \mathsf{H}(x_2)$ that collide on different inputs $x_1 \neq x_2$. Note that this is in fact captured by the collision resistance of the underlying H by definition. Thus, we have that

$$\mathsf{Adv}_2 - \mathsf{Adv}_3 \leq \epsilon_{\mathsf{H}}^{\mathsf{coll\text{-}res}}$$

Final Analysis. Now, we analyze the probability that $\mathcal{A}$ wins **Game** 3. Note that $\mathcal{A}$ can win only when if one of the following conditions holds when a server π_S^i accepts a response message in the COMPLETE oracle,

1. if $\exists (T_1, j_1), (T_2, j_2)$ such that $(T_1, j_1) \neq (T_2, j_2)$ and $\pi_{T_1}^{j_1}.\mathsf{sid} = \pi_{T_2}^{j_2}.\mathsf{sid} \neq \perp$
2. if $\exists (S_1, i_1), (S_2, i_2)$ s.t. $(S_1, i_1) \neq (S_2, i_2)$ and $\pi_{S_1}^{i_1}.\mathsf{sid} = \pi_{S_2}^{i_2}.\mathsf{sid} \neq \perp$
3. if $(S, T) \in \mathcal{L}_{\mathsf{frsh}}$ and $\nexists j$ such that $\pi_S^i.\mathsf{sid} = \pi_T^j.\mathsf{sid}$ for $T \leftarrow \mathsf{regPartner}(S)$
4. if $\exists (S', i'), (T', j')$ such that $\pi_{S'}^{i'}.\mathsf{sid} = \pi_{T'}^{j'}.\mathsf{sid} \neq \perp$ and $(S', T') \in \mathcal{L}_{\mathsf{frsh}}$ and $\pi_{S'}^{i'}.\mathsf{agCon} \neq \pi_{T'}^{j'}.\mathsf{agCon}$

Let $\mathsf{Adv}_{3.1}$, $\mathsf{Adv}_{3.2}$, $\mathsf{Adv}_{3.3}$, and $\mathsf{Adv}_{3.4}$ respectively denote the advantage of $\mathcal{A}$ in winning **Game** 3 via condition (1), (2), (3), or (4). Thus, we have that

$$\mathsf{Adv}_3 \leq \max(\mathsf{Adv}_{3.1}, \mathsf{Adv}_{3.2}, \mathsf{Adv}_{3.3}, \mathsf{Adv}_{3.4})$$

*Case (1) Note that the session identifier of the token sessions $\pi_T^j.\mathsf{sid}$ for any (T, j) includes the credential identifier cid, which is sampled by the token at the registration phase and then stored in the registration context. Note also that we have ensured that all cids sampled by tokens are distinct in **Game** 1. So, no session identifiers can be identical across the token sessions that uses different registration contexts.

Note that the identifier of a token sessions at the registration phase include the counter $n = 0$. Note also that the identifier of a token session at the authentication phase includes the counter n, which is stored in the registration context and incremented by 1 before the session identifiers are set. This means, no session identifiers of different token sessions that makes use of the same registration context collide due to the increment of counter n.

To sum up, we have that

$$\mathsf{Adv}_{3.1} = 0$$

*Case (2) First, we can observe that the session identifiers of each server session π_S^i includes $\mathsf{H}(\mathsf{id}_S)$. Note that we assume the identifier id_S of each server S is unique and that we have ensured no collision of the hash output on different inputs. So, no session identifiers can be identical across different servers.

Note that the session identifier of a server session at the registration phase does not include the $\mathsf{H}(m_{\mathsf{rcl}})$, while the one of a server session at the authentication includes $\mathsf{H}(m_{\mathsf{acl}})$.

The session identifiers of server sessions at the registration phase and the authentication phases can be easily distinguished. Note also that we have ensured that all chs are distinct in **Game** 2 and that no collision of the hash output on different inputs exists in **Game** 3. This means, no session identifiers of different sessions of the same server can be identical.

To sum up, we have that

$$\mathsf{Adv}_{3.2} = 0$$

*Case (3) Note that server session π_S^i accepts the response message only when $\mathsf{rc}_S[\mathsf{cid}] \neq \perp$ for some cid. Note also that $\mathsf{rc}_S[\mathsf{cid}] \neq \perp$ is set only in the registration phase and that all cids are sampled by the tokens followed by sending to the server session over an authenticated channel. There must be a token T that registers with the server S, which further implies that there must exist a token T such that $\mathsf{rc}_T[S] \neq \perp$. Thus, we have $T = \mathsf{regPartner}(S, i) \neq \perp$.

Then, we compute the probability of the occurrence of Case (3) by reduction. We construct an attacker $\mathcal{B}$ that breaks the $\mathsf{euf\text{-}cma}$ security of DS, which is invoked in the COMPLETE oracle, by invoking $\mathcal{A}$. Note that $\mathcal{A}$ can query REGISTER oracle at most q_{REGISTER} times. $\mathcal{B}$ first guesses an index y such that the y-th REGISTER query inputs $((S, \pi_S^i.\mathsf{regIndex}), (T, j), \mathsf{tb}, UV)$ and that the attacker $\mathcal{A}$ can finally wins condition (3) due to $\mathsf{Win\text{-}auth}(S, i)$. Note that each session can be constructed at most once. So, the existence of such y-th query is well-defined and unique. It's obvious that $\mathcal{B}$ guesses correctly with probability at least $\frac{1}{q_{\text{REGISTER}}}$.

Next, $\mathcal{B}$ receives a public verification key vk from his challenger and honestly simulates **Game** 3 to $\mathcal{A}$ except when answering the following queries:

- The y-th query REGISTER$((S, i), (T, j), \mathsf{tb}, UV)$: $\mathcal{B}_3$ honestly simulates this oracle except that he directly uses vk, the public verification key from his challenger, in the rRsp algorithm instead of sampling it by himself. Moreover, $\mathcal{B}$ records $\tilde{S} := S$ and $\tilde{T} := T$.
- RESPONSE$((T, j), m_{\mathsf{acom}})$, where $m_{\mathsf{acom}} = (\mathsf{id}, h, UV, UP)$ for $\mathsf{id} = \mathsf{id}_{\tilde{S}}$, $T = \tilde{T}$, and some h, UV, UP: $\mathcal{B}$ honestly simulates this oracle except that he queries his signing oracle on (ad, h) for the signature σ instead of computing it by himself.
- CORRUPT(S, T): If $(S, T) \neq (\tilde{S}, \tilde{T})$, $\mathcal{B}$ simply returns the $\mathsf{rc}_T[S].sk$. Otherwise, $\mathcal{B}$ aborts the simulation.

Recall that the server's session at the authentication phase is set to $\mathsf{accepted}$ only in the COMPLETE oracle. If $\mathcal{B}$ guesses the index y correctly, in order to trigger the winning condition, then $\mathcal{A}$ must query COMPLETE$((\tilde{S}, i), m_{\mathsf{acl}}, m_{\mathsf{arsp}})$ at some point for some $m_{\mathsf{acl}} = (\mathsf{ch}, \mathsf{tb})$ and $m_{\mathsf{arsp}} = (\mathsf{cid}, ad, \sigma, \mathsf{uid})$. Moreover $(\tilde{S}, \tilde{T}) \in \mathcal{L}_{\mathsf{frsh}}$ indicates that the attacker $\mathcal{A}$ has never queried CORRUPT$(\tilde{S}, \tilde{T})$, which further means that the abortion never happens.

Then, assume that $\mathcal{A}$ can win via the $\textsc{Complete}((\tilde{S}, i), m_{\mathsf{acl}}, m_{\mathsf{arsp}})$ query for some i, m_{acl}, and m_{arsp}. $\mathcal{B}$ outputs (m', σ') where $m' = (ad, \mathsf{H}(m_{\mathsf{acl}}))$ and $\sigma' = \sigma$. It is straightforward that $\mathcal{B}$ perfectly simulates **Game** 3 to $\mathcal{A}$ if $\mathcal{B}$ guesses the index y correctly.

Note that the session identifier of each token at the authentication phase is part of the m_{arsp} that it produced. The condition $\nexists j$ such that $\pi^i_{\tilde{S}}.\mathsf{sid} = \pi^j_{\tilde{T}}.\mathsf{sid}$ indicates that such m_{arsp} must not be produced by any token sessions. Further, this also implies that the message $m' = (ad, h)$ has never been sent to the signing oracle. Moreover, $\pi^i_{\tilde{S}}.\mathsf{st}_{\mathsf{exe}} = \mathsf{accepted}$ implies that $\mathsf{DS.Vfy}(vk, m', \sigma') = 1$. To sum up, $\mathcal{B}$ will always win the $\mathsf{euf\text{-}cma}$ experiment. Thus, we have that

$$\mathsf{Adv}_{3.3} \leq q_{\textsc{Register}} \epsilon^{\mathsf{euf\text{-}cma}}_{\mathsf{DS}}$$

*Case (4) First, by Case (1) and (2) we know that there are no two distinct token (resp. server) sessions that have the identical non-$\perp$ session identifiers. Thus, if there exist $(S', i'), (T', j')$ such that $\pi^{i'}_{S'}$ partner with $\pi^{j'}_{T'}$, then they are each other's unique partner.

Second, we first consider the registration phase. Note that the registration phase is over an authenticated channel. In each registration query $\textsc{Register}((S', i'), (T', j'), \mathsf{tb}, UV)$ for some tb and UV, $\pi^{i'}_{S'}$ will partner with $\pi^{j'}_{T'}$ if no abortion happens. Moreover, each message sent by the server session $\pi^{i'}_{S'}$ will then arrive at the token session $\pi^{j'}_{T'}$, which trivially indicates that $\pi^{i'}_{S'}.\mathsf{agCon} = \pi^{j'}_{T'}.\mathsf{agCon}$.

Finally, we consider the authentication phase. If $\pi^{i'}_{S'}.\mathsf{sid} \neq \perp$ is set during the authentication phase, then the session $\pi^{i'}_{S'}$ must accepts a response message via the $\textsc{Complete}$ oracle. By Case (3), we know that $\pi^{i'}_{S'}$ partner with π^j_T, where T is the registration partner of S', except probability at most $\mathsf{Adv}_{3.3}$. Recall that $\pi^{i'}_{S'}$ and $\pi^{j'}_{T'}$ are each other's unique partner. We have that $(T, j) = (T', j')$ except probability at most $\mathsf{Adv}_{3.3}$.

Note that $\pi^{i'}_{S'}.\mathsf{sid} = \pi^{j'}_{T'}.\mathsf{sid} \neq \perp$ indicates that $\pi^{i'}_{S'}$ and $\pi^{j'}_{T'}$ agree on the hash of the server identifier $\mathsf{H}(\mathsf{id}_S)$, the credential identifier cid, the hash of the client message $\mathsf{H}(m_{\mathsf{acl}})$, and the counter n. Recall that the we ensure that the hash values will not collie on different input. So, the agreement on the hash of the server identifier $\mathsf{H}(\mathsf{id}_S)$ indicates the agreement on the server identifier id_S. The attacker $\mathcal{A}$ can wins via Case (4) only when $\pi^{i'}_{S'}$ and $\pi^{j'}_{T'}$ do not have agreement on the UP and UV conditions. However, note that UP and UV are included in the associated data, which is an input of the digital signature verification algorithm in the aVrfy algorithm. By applying a reduction similar to the one in Case (3), we know that the attacker can win with probability at most $\mathsf{Adv}_{3.3} \leq q_{\textsc{Register}} \epsilon^{\mathsf{euf\text{-}cma}}_{\mathsf{DS}}$.

To sum up, the attacker $\mathcal{A}$ can wins Case (4) with advantage:

$$\mathsf{Adv}_{3.4} \leq \mathsf{Adv}_{3.3} + \mathsf{Adv}_{3.3} \leq 2 q_{\textsc{Register}} \epsilon^{\mathsf{euf\text{-}cma}}_{\mathsf{DS}}$$

Merging the statements above, we have that

$$\mathsf{Adv}_3 \leq \max(\mathsf{Adv}_{3.1}, \mathsf{Adv}_{3.2}, \mathsf{Adv}_{3.3}, \mathsf{Adv}_{3.4}) \leq 2 q_{\textsc{Register}} \epsilon^{\mathsf{euf\text{-}cma}}_{\mathsf{DS}}$$

The proof is concluded by

$$\mathsf{Adv}^{\mathsf{auth}}_{\mathsf{PIA}}(\mathcal{A}) \leq \binom{q_{\textsc{Register}}}{2} 2^{-\lambda} + \binom{q_{\textsc{Challenge}}}{2} 2^{-\lambda} + \epsilon^{\mathsf{coll\text{-}res}}_{\mathsf{H}} + 2q_{\textsc{Register}}\epsilon^{\mathsf{euf\text{-}cma}}_{\mathsf{DS}}$$

□

5.9.4 Proof of Theorem 22

Proof. We give the proof by a sequence of games. Each game is simulated between a challenger $\mathcal{C}$ and an attacker $\mathcal{A}$. Let Adv_i denote the attacker $\mathcal{A}$'s advantage in winning game i. Let $(pk_{T,i}, sk_{T,i})$ denote the ECDH public key pair owned and used by π^i_T. Let $(pk_{C,j}, sk_{C,j})$ denote the ECDH public key pair owned and used by π^j_C.

Game 0. This game is identical to the $\mathrm{Expr}^{\mathsf{SUF\text{-}t'}}_{\mathsf{PACA}}$ experiment. Hence, it holds that

$$\mathsf{Adv}_0 = \mathsf{Adv}^{\mathsf{SUF\text{-}t'}}_{\mathsf{ePACA}}(\mathcal{A})$$

Game 1. This game is identical to **Game** 0 except the following modifications:

1. At the beginning of this game, the challenger $\mathcal{C}$ sets up two lists $\mathcal{L}^1_{\mathsf{sCDH}}$ and $\mathcal{L}_{\mathsf{H}_1}$, which are initialized to $\emptyset$.
2. When the attacker $\mathcal{A}$ queries Setup and Execute oracles such that the challenger $\mathcal{C}$ needs to run $\mathsf{encapsulate}_1(pk')$ of a stateful Pin/Uv Auth Protocol $\mathsf{puvProtocol}_1$, $\mathcal{C}$ first looks up whether there exists a value $\tilde{K}$ such that $(pk', \mathsf{puvProtocol}_1.pk, \tilde{K}) \in \mathcal{L}^1_{\mathsf{sCDH}}$.

   If such value does not exist, $\mathcal{C}$ then checks for all $(u, v) \in \mathcal{L}_{\mathsf{H}_1}$ such that u is the x-coordinate of any ECDH point P whether $(pk')^{\mathsf{puvProtocol}_1.sk} = P$. If any such check succeeds, the challenger sets $\tilde{K} \leftarrow v$ and adds $(pk', \mathsf{puvProtocol}_1.pk, \tilde{K})$ into list $\mathcal{L}^1_{\mathsf{sCDH}}$.

   Otherwise, $\mathcal{C}$ simply samples $\tilde{K} \overset{\$}{\leftarrow} \{0,1\}^{l_1}$ uniformly at random and adds the tuple $(pk'$, $\mathsf{puvProtocol}_1.pk, \tilde{K})$ into list $\mathcal{L}^1_{\mathsf{sCDH}}$.

   Finally, the challenger replaces the computation of Line 101 and Line 102 in Figure 5.9 by

   $$K \leftarrow \tilde{K}$$

3. When the attacker $\mathcal{A}$ queries Setup and Send-Bind-T oracles such that the challenger $\mathcal{C}$ needs to run $\mathsf{decapsulate}_1(c)$ of a stateful Pin/Uv Auth Protocol $\mathsf{puvProtocol}_1$, $\mathcal{C}$ first looks up whether there exists a value $\tilde{K}$ such that $(\mathsf{puvProtocol}_1.pk, c, \tilde{K}) \in \mathcal{L}^1_{\mathsf{sCDH}}$.

   If such value does not exist, $\mathcal{C}$ then checks for all $(u, v) \in \mathcal{L}_{\mathsf{H}_1}$ such that u is the x-coordinate of any ECDH point P whether $c^{\mathsf{puvProtocol}_1.sk} = P$. If any such check succeeds, the challenger sets $\tilde{K} \leftarrow v$ and adds $(\mathsf{puvProtocol}_1.pk, c, \tilde{K})$ into list $\mathcal{L}^1_{\mathsf{sCDH}}$.

   Otherwise, $\mathcal{C}$ simply samples $\tilde{K} \overset{\$}{\leftarrow} \{0,1\}^{l_1}$ uniformly at random and adds $(\mathsf{puvProtocol}_1.pk$, $c, \tilde{K})$ into list $\mathcal{L}^1_{\mathsf{sCDH}}$.

Finally, the challenger replaces the computation of Line 105 and Line 106 in Figure 5.9 by

$$K \leftarrow \tilde{K}$$

4. Whenever the attacker $\mathcal{A}$ queries random oracle H_1 with input u and the random oracle outputs v, the challenger adds (u, v) into $\mathcal{L}_{\mathsf{H}_1}$.

We compute the probability that the attacker $\mathcal{A}$ can distinguish **Game** 0 and **Game** 1 by $n_{1,1}$ hybrid games, where $n_{1,1}$ denotes the number of ECDH public keys that underlie the stateful Pin/Uv Auth Protocol $\mathsf{puvProtocol}_1$ of any token and are sent to the attacker $\mathcal{A}$. Let $(pk^y_{\mathsf{token}}, sk^y_{\mathsf{token}})$ denote the y-th ECDH key pair that are sampled by the underlying stateful Pin/Uv Auth Protocol $\mathsf{puvProtocol}_1$ of any token and are sent to the attacker $\mathcal{A}$ for $y \in [n_{1,1}]$. Recall that the same ECDH key pairs of tokens might be repeatedly used by different sessions and the attacker knows the public keys of each token's or client's session only by querying Setup and Execute oracles. Each hybrid game $\mathsf{hy}.y$ for $y \in [n_{1,1}]$ is simulated as following:

Game $\mathsf{hy}.0$. This game is identical to **Game** 0 except that at the beginning of this game the challenger $\mathcal{C}$ sets up two lists $\mathcal{L}^1_{\mathsf{sCDH}}$ and $\mathcal{L}_{\mathsf{H}_1}$, which are initialized to $\emptyset$. Whenever the attacker $\mathcal{A}$ queries random oracle H_1 with input u and the random oracle outputs v, the challenger adds (u, v) into $\mathcal{L}_{\mathsf{H}_1}$. The list $\mathcal{L}^1_{\mathsf{sCDH}}$ is never used. This is indeed the modification 1 and 4 in **Game** 1. Obviously, **Game** 0 and **Game** $\mathsf{hy}.0$ are identical from the attacker's view, and we have:

$$\mathsf{Adv}_0 = \mathsf{Adv}_{\mathsf{hy}.0}$$

Game $\mathsf{hy}.y$. This game is identical to **Game** $\mathsf{hy}.(y\text{-}1)$ except the following modifications:

1. When $\mathcal{A}$ sends any query Setup or Execute on input (T, i, C, j, U) such that the underlying Pin/Uv Auth Protocol protocol of session π^i_T is a $\mathsf{puvProtocol}_1$ with $(pk_{T,i}, sk_{T,i}) = (pk^y_{\mathsf{token}}, sk^y_{\mathsf{token}})$, the challenger has to execute $\mathsf{encapsulate}_1(pk_{T,i})$. Instead of invoking $\mathsf{encapsulate}_1(pk_{T,i})$ directly, $\mathcal{C}$ first looks up whether there exists a value $\tilde{K}$ such that $(pk_{T,i}, pk_{C,j}, \tilde{K}) \in \mathcal{L}^1_{\mathsf{sCDH}}$.

   If such value does not exist, $\mathcal{C}$ then checks for all $(u, v) \in \mathcal{L}_{\mathsf{H}_1}$ such that u is the x-coordinate of any ECDH point P whether $pk_{C,j}^{sk^y_{\mathsf{token}}} = P$. If any such check succeeds, the challenger sets $\tilde{K} \leftarrow v$ and adds $(pk_{T,i}, pk_{C,j}, \tilde{K})$ into list $\mathcal{L}^1_{\mathsf{sCDH}}$.

   Otherwise, $\mathcal{C}$ simply samples $\tilde{K} \xleftarrow{\$} \{0,1\}^{l_1}$ uniformly at random and adds $(pk_{T,i}, pk_{C,j}, \tilde{K})$ into list $\mathcal{L}^1_{\mathsf{sCDH}}$.

   Finally, the challenger replaces the computation of Line 101 and Line 102 in Figure 5.9 by

$$K \leftarrow \tilde{K}$$

2. When $\mathcal{A}$ sends the query SEND-BIND-T on input (T, i, m) such that $(pk_{T,i}, sk_{T,i}) = (pk^y_{\mathsf{token}}, sk^y_{\mathsf{token}})$ and that $m = c \parallel c_{ph}$, the challenger first checks whether there exists a value $\tilde{K}$ such that $(pk_{T,i}, c, \tilde{K}) \in \mathcal{L}^1_{\mathsf{sCDH}}$.

   If such value does not exist, $\mathcal{C}$ then checks for all $(u, v) \in \mathcal{L}_{\mathsf{H}_1}$ such that u is the x-coordinate of any ECDH point P whether $c = P$. If any such check succeeds, the challenger sets $\tilde{K} \leftarrow v$ and adds $(pk_{T,i}, c, \tilde{K})$ into list $\mathcal{L}^1_{\mathsf{sCDH}}$.

   Otherwise, $\mathcal{C}$ simply samples $\tilde{K} \xleftarrow{\$} \{0,1\}^{l_1}$ uniformly at random and adds $(pk_{T,i}, c, \tilde{K})$ into list $\mathcal{L}^1_{\mathsf{sCDH}}$.

   Finally, the challenger $\mathcal{C}$ is supposed to execute $\mathsf{decapsulate}_1(c)$. Instead of invoking $\mathsf{decapsulate}_1(c)$ directly, $\mathcal{C}$ replaces the computation of Line 105 and Line 106 in Figure 5.9 by

$$K \leftarrow \tilde{K}$$

Let event E_1 denote the probability that the attacker $\mathcal{A}$ can distinguish **Game** $\mathsf{hy}.(y\text{-}1)$ and **Game** $\mathsf{hy}.y$. Note that the modifications between every two adjacent hybrid games are independent. It holds that

$$\mathsf{Adv}_{\mathsf{hy}.(y\text{-}1)} - \mathsf{Adv}_{\mathsf{hy}.y} \leq \Pr[E_1], \forall y \in [n_{1,1}]$$

Then, we analyze the probability of the occurrence of E_1 by reduction. Namely, if E_1 occurs, then we can construct an attacker $\mathcal{B}_1$ that breaks sCDH assumption over ECDH by invoking $\mathcal{A}$. On inputs $(\mathsf{ECDH}, A = g^a, B = g^b)$, $\mathcal{B}_1$ sets the y-th ECDH public key pk^y_{token} among all ECDH public key underlying any $\mathsf{puvProtocol}_1$ of all tokens to be $A = g^a$. Then, $\mathcal{B}_1$ simulates **Game** $\mathsf{hy}.(y\text{-}1)$ honestly, except the following modifications:

1. When $\mathcal{A}$ sends $\mathcal{B}_1$ the w-th query SETUP or EXECUTE on input (T, i, C, j, U) for $w \geq 1$ such that $pk_{T,i}$ is supposed to be pk^y_{token}, $\mathcal{B}_1$ first samples $r_w \leftarrow \mathbb{Z}_q$, where q is the prime order of the the underlying cyclic group of ECDH and sets $pk_{C,j} \leftarrow B \cdot g^{r_w} = g^{b+r_w}$ in the $\mathsf{obtainSharedSecret}\text{-}C\text{-}\mathsf{end}$ algorithms. Next, when $\mathcal{B}_1$ needs to run $\mathsf{encapsulate}_1(pk_{T,i})$ algorithm in $\mathsf{obtainSharedSecret}\text{-}C\text{-}\mathsf{end}$ algorithm, $\mathcal{B}_1$ first looks up whether there exists a value $\tilde{K}$ such that $(pk_{T,i}, pk_{C,j}, \tilde{K}) \in \mathcal{L}^1_{\mathsf{sCDH}}$.

   If such value does not exist, for all $(u, v) \in \mathcal{L}_{\mathsf{H}_1}$ such that u is the x-coordinate of any ECDH point P, $\mathcal{B}_1$ queries its $\mathcal{O}_a$ oracle on $(pk_{C,j}, P)$. If any response is true, the challenger sets $\tilde{K} \leftarrow v$ and adds $(pk_{T,i}, pk_{C,j}, \tilde{K})$ into list $\mathcal{L}^1_{\mathsf{sCDH}}$.

   Otherwise, $\mathcal{B}_1$ simply samples $\tilde{K} \xleftarrow{\$} \{0,1\}^{l_1}$ uniformly at random and adds $(pk_{T,i}, pk_{C,j}, \tilde{K})$ into list $\mathcal{L}^1_{\mathsf{sCDH}}$.

   Finally, $\mathcal{B}_1$ honestly performs the remaining execution except replacing the computation of Line 101 and Line 102 in Figure 5.9 by

$$K \leftarrow \tilde{K}$$

2. When $\mathcal{A}$ sends the query SEND-BIND-T on input (T, i, m) such that $pk_{T,i} = A = g^a$ and $m = c \parallel c_{ph}$, $\mathcal{B}_1$ first checks whether there exists a value $\tilde{K}$ such that $(pk_{T',i'}, c, \tilde{K}) \in \mathcal{L}^1_{\mathsf{sCDH}}$.

   If such value does not exist, for each tuple $(u, v) \in \mathcal{L}_{\mathsf{H}_1}$ such that u is the x-coordinate of any point P on ECDH, $\mathcal{B}_1$ queries $\mathcal{O}_a$ to its challenger on input (c, P). If any response from the challenger is true, $\mathcal{B}_1$ uses the corresponding value v as the hash H_1 of x-coordinate of the Diffie-Hellman exchange of $pk_{T,i}$ and c for the subsequent computation.

   If no response from the challenger is true, this means, c_{ph} is a random ciphertext that is produced by $\mathcal{A}$ without knowing the correct symmetric key K. $\mathcal{B}_1$ simply samples $\tilde{K} \xleftarrow{\$} \{0,1\}^{l_1}$ uniformly at random and adds $(pk_{T,i}, c, \tilde{K})$ into list $\mathcal{L}^1_{\mathsf{sCDH}}$. Finally, $\mathcal{B}_1$ simply uses $\tilde{K}$ as the hash H_1 of x-coordinate of the Diffie-Hellman exchange of $pk_{T,i}$ and c for the subsequent computation.
3. Finally, $\mathcal{A}$ terminates at some point and is expected to distinguish **Game** $\mathsf{hy}.(y\text{-}1)$ from **Game** $\mathsf{hy}.y$. For all $(u, v) \in \mathcal{L}_{\mathsf{H}_1}$ such that u is the x-coordinate of some ECDH point P and all r_w sampled above, $\mathcal{B}_1$ queries $\mathcal{O}_a(B \cdot g^{r_w}, P)$ to its challenger. If any response is true, $\mathcal{B}_1$ returns $P \cdot A^{-r_w}$ to its challenger. Otherwise, $\mathcal{B}_1$ outputs a random cyclic group element on ECDH.

Obviously, $\mathcal{B}_1$ simulates **Game** $\mathsf{hy}.(y\text{-}1)$ and **Game** $\mathsf{hy}.y$ to $\mathcal{A}$ perfectly. From the attacker $\mathcal{A}$'s view, the only difference between **Game** $\mathsf{hy}.(y\text{-}1)$ and **Game** $\mathsf{hy}.y$ is whether the key $\tilde{K}$ is computed from the hash H_1 of the x-coordinate of the real Diffie-Hellman exchange or sampled uniformly at random. If $\mathcal{A}$ can distinguish **Game** $\mathsf{hy}.(y\text{-}1)$ from **Game** $\mathsf{hy}.y$ effectively, $\mathcal{A}$ must have queried the x-coordinate of the real Diffie-Hellman exchange of A and $B \cdot g^{r_w}$ to the random oracle H_1 for some w. This means, $\mathcal{B}_1$ can always return $P \cdot A^{-r_w} = (B \cdot g^{r_w})^a \cdot A^{-r_w} = g^{ab+ar_w-ar_w} = g^{ab}$ to its challenger. Thus, it holds that

$$\Pr[E_1] \leq \epsilon^{\mathsf{sCDH}}_{\mathsf{ECDH}}$$

Furthermore, **Game** $\mathsf{hy}.n_{1,1}$ have replaced all shared symmetric key K produced by honest clients in $\mathsf{encapsulate}_1$ algorithm with a random key $\tilde{K}$. Thus, **Game** $\mathsf{hy}.n_{1,1}$ is identical to **Game** 1 and we have:

$$\mathsf{Adv}_1 = \mathsf{Adv}_{\mathsf{hy}.n_{1,1}}$$

Combing the statements above, we have that

$$\mathsf{Adv}_0 - \mathsf{Adv}_1 \leq n_{1,1}\epsilon^{\mathsf{sCDH}}_{\mathsf{ECDH}}$$

For now, we continue to use the term $n_{1,1}$ and will reduce it in the subsequent games.

Game 2. This game is identical **Game** 1 except that the challenger $\mathcal{C}$ aborts the simulation and let $\mathcal{A}$ immediately win if there exists collision between $\tilde{K}$ sampled in **Game** 1. Recall that $\tilde{K}$s are sampled at most $n_{1,1}$ times. Note that the collision happens between every two keys with probability at most 2^{-l_1} and that there exists at most $\binom{n_{1,1}}{2}$ pairs. We have that

$$\mathsf{Adv}_1 - \mathsf{Adv}_2 \leq \binom{n_{1,1}}{2} 2^{-l_1}$$

Game 3. This game is identical to **Game** 2 except the following modifications:

1. At the beginning of this game, the challenger $\mathcal{C}$ sets up two lists $\mathcal{L}^2_{\mathsf{sCDH}}$ and $\mathcal{L}_{\mathsf{H}_3}$, which are initialized to $\emptyset$.
2. When the attacker $\mathcal{A}$ queries Setup and Execute oracles such that the challenger $\mathcal{C}$ needs to run $\mathsf{encapsulate}_2(pk')$ of a stateful Pin/Uv Auth Protocol $\mathsf{puvProtocol}_2$, $\mathcal{C}$ first looks up whether there exists values $\tilde{K}_1$ and $\tilde{K}_2$ such that $(pk', \mathsf{puvProtocol}_2.pk, \tilde{K}_1, \tilde{K}_2) \in \mathcal{L}^2_{\mathsf{sCDH}}$.

   If such value does not exist, $\mathcal{C}$ then checks for all $((u, u'), v) \in \mathcal{L}_{\mathsf{H}_3}$ such that u is the x-coordinate of any ECDH point P and $u' \in \{$"CTAP2 HMAC key", "CTAP2 AES key"$\}$ whether $(pk')^{\mathsf{puvProtocol}_2.sk} = P$. If any such check succeeds, the challenger queries random oracle H_3 and sets $\tilde{K}_1 \leftarrow \mathsf{H}_3(u,$ "CTAP2 HMAC key"$)$ and $\tilde{K}_2 \leftarrow \mathsf{H}_3(u,$ "CTAP2 AES key"$)$ and adds $(pk', \mathsf{puvProtocol}_2.pk, \tilde{K}_1, \tilde{K}_2)$ into list $\mathcal{L}^2_{\mathsf{sCDH}}$.

   Otherwise, $\mathcal{C}$ simply samples $\tilde{K}_1, \tilde{K}_2 \xleftarrow{\$} \{0,1\}^{l_3}$ uniformly at random and adds $(pk', \mathsf{puvProtocol}_2.pk, \tilde{K}_1, \tilde{K}_2)$ into list $\mathcal{L}^2_{\mathsf{sCDH}}$.

   Finally, the challenger replaces the computation of Line 121 - 123 in Figure 5.10 by

   $$K_1 \leftarrow \tilde{K}_1, K_2 \leftarrow \tilde{K}_2$$

3. When the attacker $\mathcal{A}$ queries Setup and Send-Bind-T oracles such that the challenger $\mathcal{C}$ needs to run $\mathsf{decapsulate}_2(c)$ of a stateful Pin/Uv Auth Protocol $\mathsf{puvProtocol}_2$, $\mathcal{C}$ first looks up whether there exists values $\tilde{K}_1$ and $\tilde{K}_2$ such that $(\mathsf{puvProtocol}_2.pk, c, \tilde{K}_1, \tilde{K}_2) \in \mathcal{L}^2_{\mathsf{sCDH}}$.

   If such value does not exist, $\mathcal{C}$ then checks for all $((u, u'), v) \in \mathcal{L}_{\mathsf{H}_3}$ such that u is the x-coordinate of any ECDH point P and $u' \in \{$"CTAP2 HMAC key","CTAP2 AES key"$\}$ whether $c^{\mathsf{puvProtocol}_2.sk} = P$. If any such check succeeds, the challenger queries random oracle H_3 and sets $\tilde{K}_1 \leftarrow \mathsf{H}_3(u,$ "CTAP2 HMAC key"$)$ and $\tilde{K}_2 \leftarrow \mathsf{H}_3(u,$"CTAP2 AES key"$)$ and adds $(pk', \mathsf{puvProtocol}_2.pk, \tilde{K}_1, \tilde{K}_2)$ into list $\mathcal{L}^2_{\mathsf{sCDH}}$.

   Otherwise, $\mathcal{C}$ simply samples $\tilde{K}_1, \tilde{K}_2 \xleftarrow{\$} \{0,1\}^{l_3}$ uniformly at random and adds the tuple $(\mathsf{puvProtocol}_2.pk, c, \tilde{K}_1, \tilde{K}_2) \in \mathcal{L}^2_{\mathsf{sCDH}}$ into list $\mathcal{L}^2_{\mathsf{sCDH}}$.

   Finally, the challenger replaces the computation of Line 127- 129 in Figure 5.10 by

   $$K_1 \leftarrow \tilde{K}_1, K_2 \leftarrow \tilde{K}_2$$

4. Whenever the attacker $\mathcal{A}$ queries random oracle H_3 with input u and the random oracle outputs v, the challenger adds (u, v) into $\mathcal{L}_{\mathsf{H}_3}$.

Similar to **Game** 1, we can compute the probability that the attacker $\mathcal{A}$ can distinguish **Game** 2 and **Game** 3 by $n_{1,2}$ hybrid games, where $n_{1,2}$ denotes the number of ECDH public keys that underlie the stateful Pin/Uv Auth Protocol $\mathsf{puvProtocol}_2$ of any token and are sent to the attacker $\mathcal{A}$. Thus, we can easily have that

$$\mathsf{Adv}_2 - \mathsf{Adv}_3 \leq n_{1,2}\epsilon^{\mathsf{sCDH}}_{\mathsf{ECDH}}$$

Game 4. This game is identical **Game** 3 except that the challenger $\mathcal{C}$ aborts the simulation and lets $\mathcal{A}$ immediately win if there exists collision between $\tilde{K}_1$ or collision between $\tilde{K}_2$ sampled in **Game** 3. Recall that $\tilde{K}_1$ and $\tilde{K}_2$ both are sampled at most $n_{1,2}$ times. Note that the collision happens between every two keys with probability at most 2^{-l_3} and that there exists at most $\binom{n_{1,2}}{2}$ pairs. We have that

$$\mathsf{Adv}_3 - \mathsf{Adv}_4 \leq 2\binom{n_{1,2}}{2}2^{-l_3} = \binom{n_{1,2}}{2}2^{-l_3+1}$$

Game 5. This game is identical to **Game** 4 except the following modifications:

1. At the beginning of this game, the challenger $\mathcal{C}$ sets up two lists $\mathcal{L}^3_{\mathsf{sCDH}}$ and $\mathcal{L}_{\mathsf{H}_5}$, which are initialized to $\emptyset$.
2. When the attacker $\mathcal{A}$ queries Setup and Execute oracles such that the challenger $\mathcal{C}$ needs to run $\mathsf{encapsulate}_3(pk')$ of a stateful Pin/Uv Auth Protocol $\mathsf{puvProtocol}_3$, where $pk' = (pk'_1, pk'_2)$, $\mathcal{C}$ first executes $(c_2, Z_2) \leftarrow \mathsf{KEM.KEM.Encaps}(pk'_2)$ and looks up whether there exists a value $\tilde{Z}$ such that $(pk'_1, \mathsf{puvProtocol}_3.pk_1, Z_2, \tilde{Z}) \in \mathcal{L}^3_{\mathsf{sCDH}}$.

   If such value does not exist, $\mathcal{C}$ then checks for all $((u, u'), v) \in \mathcal{L}_{\mathsf{H}_5}$ such that u is the x-coordinate of any ECDH point P and $u' = Z_2$ whether $(pk'_1)^{\mathsf{puvProtocol}_3.sk_1} = P$. If any such check succeeds, the challenger sets $\tilde{Z} \leftarrow v$ and adds $(pk'_1, \mathsf{puvProtocol}_3.pk_1, Z_2, \tilde{Z})$ into list $\mathcal{L}^3_{\mathsf{sCDH}}$.

   Otherwise, $\mathcal{C}$ simply samples $\tilde{Z} \xleftarrow{\$} \{0,1\}^{l_5}$ uniformly at random and adds the tuple $(pk'_1, \mathsf{puvProtocol}_3.pk_1, Z_2, \tilde{Z})$ into list $\mathcal{L}^3_{\mathsf{sCDH}}$.

   Finally, the challenger replaces the computation of Line 167 - 169 in Figure 5.11 by

$$Z \leftarrow \tilde{Z}$$

3. When the attacker $\mathcal{A}$ queries Setup and Send-Bind-T oracles such that the challenger $\mathcal{C}$ needs to run $\mathsf{decapsulate}_3(c)$ of a stateful Pin/Uv Auth Protocol $\mathsf{puvProtocol}_3$, where $c = (c_1, c_2)$, $\mathcal{C}$ first executes $Z_2 \leftarrow \mathsf{KEM.KEM.Decaps}(\mathsf{puvProtocol}_3.sk_2, c_2)$ and looks up whether there exists a value $\tilde{Z}$ such that $(\mathsf{puvProtocol}_3.pk_1, c_1, Z_2, \tilde{Z}) \in \mathcal{L}^3_{\mathsf{sCDH}}$.

If such value does not exist, $\mathcal{C}$ then checks for all $((u,u'),v) \in \mathcal{L}_{\mathsf{H}_5}$ such that u is the x-coordinate of any ECDH point P and $u' = Z_2$ whether $(c_1)^{\mathsf{puvProtocol}_3.sk_1} = P$. If any such check succeeds, the challenger sets $\tilde{Z} \leftarrow v$ and adds $(\mathsf{puvProtocol}_3.pk_1, c_1, Z_2, \tilde{Z})$ into list $\mathcal{L}^3_{\mathsf{sCDH}}$.

Otherwise, $\mathcal{C}$ simply samples $\tilde{Z} \xleftarrow{\$} \{0,1\}^{l_5}$ uniformly at random and adds $(\mathsf{puvProtocol}_3.pk_1, c_1, Z_2, \tilde{Z})$ into list $\mathcal{L}^3_{\mathsf{sCDH}}$.

Finally, the challenger replaces the computation of Line 177 - 179 in Figure 5.11 by

$$Z \leftarrow \tilde{Z}$$

4. Whenever the attacker $\mathcal{A}$ queries random oracle H_5 with input u and the random oracle outputs v, the challenger adds (u,v) into $\mathcal{L}_{\mathsf{H}_5}$.

Similar to **Game** 1, we can compute the probability that the attacker $\mathcal{A}$ can distinguish **Game** 4 and **Game** 5 by $n_{1,3}$ hybrid games, where $n_{1,3}$ denotes the number of ECDH public keys that underlie the stateful Pin/Uv Auth Protocol $\mathsf{puvProtocol}_3$ of any token and are sent to the attacker $\mathcal{A}$. Thus, we can easily have that

$$\mathsf{Adv}_4 - \mathsf{Adv}_5 \leq n_{1,3}\epsilon^{\mathsf{sCDH}}_{\mathsf{ECDH}}$$

Game 6. This game is identical **Game** 5 except that the challenger $\mathcal{C}$ aborts the simulation and let $\mathcal{A}$ immediately win if there exists collision between $\tilde{Z}$ sampled in **Game** 5. Recall that $\tilde{Z}$s are sampled (either uniformly at random or from the random oracle) at most $n_{1,3}$ times. Note that the collision happens between every two keys with probability at most 2^{-l_5} and that there exist at most $\binom{n_{1,3}}{2}$ pairs. We have that

$$\mathsf{Adv}_5 - \mathsf{Adv}_6 \leq \binom{n_{1,3}}{2} 2^{-l_5}$$

Game 7. This game is identical **Game** 6 except that the challenger $\mathcal{C}$ aborts the simulation and let $\mathcal{A}$ immediately win if there exists collision between K_1s or collision between K_2s derived in $\mathsf{encapsulate}_3$. Recall that K_1s and K_2s both are produced by $\mathsf{H}_6(\tilde{Z}, \text{"CTAP2 HMAC key"})$ in $\mathsf{encapsulate}_3$ at most $n_{1,3}$ times and that $\tilde{Z}$s are assumed to be distinct from each other in **Game** 6. Note that the collision happens between every two keys with probability at most 2^{-l_6} and that there exists at most $\binom{n_{1,3}}{2}$ pairs. We have that

$$\mathsf{Adv}_6 - \mathsf{Adv}_7 \leq 2\binom{n_{1,3}}{2} 2^{-l_6} = \binom{n_{1,3}}{2} 2^{-l_6+1}$$

Recall that the honest public keys of tokens (resp. the ones of clients) used in the $\mathsf{encapsulate}_i$ and $\mathsf{decapsulate}_i$ for $i \in \{1,2,3\}$ are sent to attacker $\mathcal{A}$ in $\mathsf{obtainSharedSecret}\text{-}T$ (resp. $\mathsf{obtainSharedSecret}\text{-}C\text{-}\mathsf{end}$) algorithm only when answering the SETUP and EXECUTE oracles. So, we have that $n_{1,1} + n_{1,2} + n_{1,3} \leq q_{\textsc{Setup}} + q_{\textsc{Execute}}$.

Merging the statements above, we can state the upper bound as:

$$\begin{aligned}\mathsf{Adv}_0 - \mathsf{Adv}_7 \leq &(n_{1,1} + n_{1,2} + n_{1,3})\epsilon^{\mathsf{sCDH}}_{\mathsf{ECDH}} + \binom{n_{1,1}}{2}2^{-l_1} + \binom{n_{1,2}}{2}2^{-l_3+1} \\ &+ \binom{n_{1,3}}{2}(2^{-l_5} + 2^{-l_6+1}) \\ \leq &(q_{\textsc{Setup}} + q_{\textsc{Execute}})\epsilon^{\mathsf{sCDH}}_{\mathsf{ECDH}} + \binom{q_{\textsc{Setup}} + q_{\textsc{Execute}}}{2}2^{2-\min(l_1,l_3,l_5,l_6)}\end{aligned}$$

Game 8. In this game, the challenger $\mathcal{C}$ aborts the game and lets the attacker $\mathcal{A}$ immediately win if there exist two inputs pin and pin' during $\mathcal{C}$'s execution such that $\mathsf{H}(\mathsf{pin}) = \mathsf{H}(\mathsf{pin}')$. This violates the collision resistance of H by definition. Since H is assumed to be $\epsilon^{\mathsf{coll\text{-}res}}_{\mathsf{H}}$-collision resistant, we have that

$$\mathsf{Adv}_7 - \mathsf{Adv}_8 \leq \epsilon^{\mathsf{coll\text{-}res}}_{\mathsf{H}}$$

Game 9. In this game, the challenger $\mathcal{C}$ aborts the game and lets the attacker $\mathcal{A}$ immediately win if the challenger honestly samples two identical ECDH public keys of tokens or of clients and sends them to the attacker. Recall that each sampled ECDH public keys are sent to the attacker only in Setup or Execute oracles. And one newly sampled ECDH keys of tokens and one of clients are sent to the attacker in both Setup and Execute queries. In total, there are at most $(q_{\textsc{Setup}} + q_{\textsc{Execute}})$ ECDH public keys of tokens and $(q_{\textsc{Setup}} + q_{\textsc{Execute}})$ ECDH public keys of clients. Note that the collision happens between every two public keys with probability at most 2^{-q}, where q denote the prime order of the underlying ECDH group, and that there exist at most $\binom{(q_{\textsc{Setup}}+q_{\textsc{Execute}})}{2}$ pairs of tokens (resp. of clients). We have that

$$\mathsf{Adv}_8 - \mathsf{Adv}_9 \leq 2\binom{(q_{\textsc{Setup}} + q_{\textsc{Execute}})}{2}2^{-q} \leq \binom{(q_{\textsc{Setup}} + q_{\textsc{Execute}})}{2}2^{1-q}$$

Game 10. This game is identical to **Game** 9 except that the following modifications:

1. The challenger $\mathcal{C}$ samples a random $\tilde{\mathsf{pin}} \xleftarrow{\$} \mathfrak{D}$ at the beginning of the game but never uses it. The challenger aborts and lets $\mathcal{A}$ immediately win if $\tilde{\mathsf{pin}}$ collides with any user pin pin_U for any user U.
2. Whenever the attacker $\mathcal{A}$ queries oracle Setup inputting any (T, i, C, j, U), the challenger replaces $\mathsf{pin} \leftarrow \mathsf{st}_T.\mathsf{puvProtocol.decrypt}(K, c_p)$ in the $\mathsf{setPIN}\text{-}T$ algorithm by

$$\mathsf{pin} \leftarrow \mathsf{pin}_U$$

3. The challenger aborts the game and lets $\mathcal{A}$ immediately win if there exits a collision between pts used in Send-Bind-T oracles.

Note that the Setup phase is assumed to be authenticated. The user U's pin pin_U, which was encrypted by the client, can always be decrypted by the token.

Moreover, note that the attacker can query NewU at most q_{NewU} times and each user pin is sampled from distribution $\mathfrak{D}$ with min-entropy $\alpha_{\mathfrak{D}}$. The probability that $\tilde{\mathsf{pin}}$ collides with any other user pin pin_U is bounded by $q_{\text{NewU}} 2^{-\alpha_{\mathfrak{D}}}$.

Furthermore, note that the pts are random strings of length $\mu\lambda, 2\lambda, \mu'\lambda$ respectively in $\mathsf{puvProtocol}_1, \mathsf{puvProtocol}_2$, and $\mathsf{puvProtocol}_3$. Note also that pts are used only in Send-Bind-T oracles. The collision between pts happens with probability at most $\binom{q_{\text{Send-Bind-T}}}{2} 2^{-\min(\mu,2,\mu')\lambda}$.

So, we have that

$$\mathsf{Adv}_9 - \mathsf{Adv}_{10} \leq q_{\text{NewU}} 2^{-\alpha_{\mathfrak{D}}} + \binom{q_{\text{Send-Bind-T}}}{2} 2^{-\min(\mu,2,\mu')\lambda}$$

Game 11. This games is identical to **Game** 10 except the following modification:

1. Whenever $\mathcal{A}$ queries $\text{Setup}(T, i, C, j, U)$ and the challenger sets the selected Pin/Uv Auth Protocol of the client session π_C^j to be $\pi_C^j.\mathsf{selectedpuvProtocol} = \mathsf{puvProtocol}_1$ during the execution, the challenger replaces $c_p = \mathsf{SKE}_1.\mathsf{Enc}(\tilde{K}, \mathsf{pin}_U)$ in the $\mathsf{setPIN}\text{-}C(\pi_C^j, \mathsf{pin}_U)$ by $\tilde{c}_p \leftarrow \mathsf{SKE}_1.\mathsf{Enc}(\tilde{K}, \tilde{\mathsf{pin}})$, where $\tilde{\mathsf{pin}}$ was sampled in **Game** 10.

We prove that **Game** 10 and **Game** 11 are indistinguishable from $\mathcal{A}$'s view by $n_{2,1}$ hybrid games, where $n_{2,1}$ denotes the number of $\tilde{K}$ sampled in Setup oracle when the underlying Pin/Uv Auth Protocol is $\mathsf{puvProtocol}_1$. Let $\tilde{K}^y$ denotes the y-th $\tilde{K}$ sampled by the challenger $\mathcal{C}$ in Setup oracle when the underlying Pin/Uv Auth Protocol is $\mathsf{puvProtocol}_1$. The hybrid game $\mathsf{hy}.y$ for $y \in [n_{2,1}]$ is defined below.

Game $\mathsf{hy}.0$. This game is identical to **Game** 10 and we have that:

$$\mathsf{Adv}_{10} = \mathsf{Adv}_{\mathsf{hy}.0}$$

Game $\mathsf{hy}.y$. This game is identical to **Game** $\mathsf{hy}.(y\text{-}1)$ except the following modifications:

1. When $\mathcal{A}$ queries $\text{Setup}(T, i, C, j, U)$ and that will produce the y-th $\tilde{K}^y$ during the game, the challenger replaces $c_p = \mathsf{SKE}_1.\mathsf{Enc}(\tilde{K}^y, \mathsf{pin}_U)$ in the $\mathsf{setPIN}\text{-}C(\pi_C^j, \mathsf{pin}_U)$ by $\tilde{c}_p \leftarrow \mathsf{SKE}_1.\mathsf{Enc}(\tilde{K}^y, \tilde{\mathsf{pin}})$.

Let event E_2 denote the probability that the attacker $\mathcal{A}$ can distinguish **Game** $\mathsf{hy}.(y\text{-}1)$ and **Game** $\mathsf{hy}.y$. Note that the modifications between every two adjacent hybrid games are independent. It holds that

$$\mathsf{Adv}_{\mathsf{hy}.(y\text{-}1)} - \mathsf{Adv}_{\mathsf{hy}.y} \leq \Pr[E_2], \forall y \in [n_{2,1}]$$

Then, we analyze the probability of the occurrence of E_2 by reduction. Namely, if E_2 occurs, then we can construct an attacker $\mathcal{B}_2$ that breaks $\mathsf{IND\text{-}1CPA\text{-}H}_2$ security of

SKE_1 by invoking $\mathcal{A}$. $\mathcal{B}_2$ simulates **Game** $\mathsf{hy}.(y\text{-}1)$ honestly, except for the query to the $\textsc{Setup}(T, i, C, j, U)$ that will produce the y-th $\tilde{K}^y$ during the game. To handle this query, $\mathcal{B}_2$ executes following step:

1. $\mathcal{B}_2$ sends $(\mathsf{pin}_U, \tilde{\mathsf{pin}})$ to its challenger and obtains (c, t). Then, $\mathcal{B}_2$ sets (c, t) as the output of $\mathsf{setPIN}\text{-}C(\pi_C^j, \mathsf{pin}_U)$ algorithm.

Note that we have ensured that all sampled ECDH public keys are distinct in **Game** 9 and that all sampled $\tilde{K}$s in $\mathcal{L}^1_{\mathsf{sCDH}}$ are distinct in **Game** 2. It's easy to observe that $\mathcal{B}_2$ perfectly simulates **Game** $\mathsf{hy}.(y\text{-}1)$ or **Game** $\mathsf{hy}.y$. Moreover, $\mathcal{B}_2$ simulates **Game** $\mathsf{hy}.(y\text{-}1)$ if $(c, t) = (\mathsf{SKE}_1.\mathsf{Enc}(\tilde{K}^y, \mathsf{pin}_U), \mathsf{H}_2(\tilde{K}^y, c))$ and **Game** $\mathsf{hy}.y$ if $(c, t) = (\mathsf{SKE}_1.\mathsf{Enc}(\tilde{K}^y, \tilde{\mathsf{pin}}), \mathsf{H}_2(\tilde{K}^y, c))$. Thus, $\mathcal{B}_2$ can win $\mathsf{IND\text{-}1CPA\text{-}H_2}$ experiment whenever $\mathcal{A}$ can distinguish **Game** $\mathsf{hy}.(y\text{-}1)$ and **Game** $\mathsf{hy}.y$. Thus, we have that

$$\Pr[E_2] \le \epsilon_{\mathsf{SKE}_1}^{\mathsf{ind\text{-}1cpa\text{-}H_2}}$$

Moreover, **Game** $\mathsf{hy}.n_{2,1}$ have replaced all c_p in the $\textsc{Setup}$ oracles whenever the client chooses $\mathsf{puvProtocol}_1$. Thus, **Game** $\mathsf{hy}.n_{2,1}$ is identical to **Game** 11 and we have

$$\mathsf{Adv}_{11} = \mathsf{Adv}_{\mathsf{hy}.n_{2,1}}$$

Note that all hybrid games are independent. Combing the statements above, we have that

$$\mathsf{Adv}_{10} - \mathsf{Adv}_{11} \le n_{2,1}\epsilon_{\mathsf{SKE}_1}^{\mathsf{ind\text{-}1cpa\text{-}H_2}}$$

Here, we simply keep using the number $n_{2,1}$, which would be helpful for us to tighten our security upper bound in the following games.

Game 12. This games is identical to **Game** 11 except the following modification:

1. When $\mathcal{A}$ queries $\textsc{Setup}(T, i, C, j, U)$, where pin_U is not corrupted, and the challenger sets the selected Pin/Uv Auth Protocol of the client session π_C^j to be $\pi_C^j.\mathsf{selectedpuvProtocol} = \mathsf{puvProtocol}_2$ during the execution, the challenger replaces $c_p = \mathsf{SKE}_2.\mathsf{Enc}(\tilde{K}_2, \mathsf{pin}_U)$ in the $\mathsf{setPIN}\text{-}C(\pi_C^j, \mathsf{pin}_U)$ by $\tilde{c}_p \leftarrow \mathsf{SKE}_2.\mathsf{Enc}(\tilde{K}_2, \tilde{\mathsf{pin}})$, where $\tilde{\mathsf{pin}}$ was sampled in **Game** 10.

We prove that **Game** 11 and **Game** 12 are indistinguishable from $\mathcal{A}$'s view by $n_{2,2}$ hybrid games, where $n_{2,2}$ denotes the number of $\tilde{K}_2$ sampled in $\textsc{Setup}$ oracle when the underlying Pin/Uv Auth Protocol is $\mathsf{puvProtocol}_2$. Let $\tilde{K}_2^y$ denote the y-th $\tilde{K}_2$ sampled in $\textsc{Setup}$ oracle when the underlying Pin/Uv Auth Protocol is $\mathsf{puvProtocol}_2$. The hybrid game $\mathsf{hy}.y$ for $y \in [n_{2,2}]$ is defined below.

Game $\mathsf{hy}.0$. This game is identical to **Game** 11 and we have:

$$\mathsf{Adv}_{11} = \mathsf{Adv}_{\mathsf{hy}.0}$$

Game $\mathsf{hy}.y$**.** This game is identical to **Game** $\mathsf{hy}.(y\text{-}1)$ except the following modifications:

1. When $\mathcal{A}$ queries $\textsc{Setup}(T, i, C, j, U)$ that will make use of the y-th $\tilde{K}_2^y$ for $\mathsf{puvProtocol}_2$ during the game, the challenger replaces $c_p = \mathsf{SKE}_2.\mathsf{Enc}(\tilde{K}_2^y, \mathsf{pin}_U)$ in the execution $\mathsf{setPIN}\text{-}C(\pi_C^j, \mathsf{pin}_U)$ by $\tilde{c}_p \leftarrow \mathsf{SKE}_2.\mathsf{Enc}(\tilde{K}_2^y, \tilde{\mathsf{pin}})$.

Let event E_3 denote the probability that the attacker $\mathcal{A}$ can distinguish **Game** $\mathsf{hy}.(y\text{-}1)$ and **Game** $\mathsf{hy}.y$. Note that the modifications between every two adjacent games are independent. It holds that

$$\mathsf{Adv}_{\mathsf{hy}.(y\text{-}1)} - \mathsf{Adv}_{\mathsf{hy}.y} \leq \Pr[E_3], \forall y \in [n_{2,2}]$$

Then, we analyze the probability of the occurrence of E_3 by reduction. Namely, if E_3 occurs, then we can construct an attacker $\mathcal{B}_3$ that breaks $\mathsf{IND\text{-}1CPA}$ security of SKE_2 by invoking $\mathcal{A}$. $\mathcal{B}_3$ simulates **Game** $\mathsf{hy}.(y\text{-}1)$ honestly, except for the query to the $\textsc{Setup}(T, i, C, j, U)$ and that will produce the y-th $\tilde{K}_2^y$ for $\mathsf{puvProtocol}_2$ during the game. To handle this query, $\mathcal{B}_3$ executes the following steps:

1. $\mathcal{B}_3$ sends query$(\mathsf{pin}_U, \tilde{\mathsf{pin}})$ to its challenger and obtains c. Then, $\mathcal{B}_3$ sets c as the first output of $\mathsf{setPIN}\text{-}C(\pi_C^j, \mathsf{pin}_U)$ algorithm.

Note that we have already ensured that all ECDH public keys are distinct in **Game** 9 and that all used $\tilde{K}_2$ are distinct in **Game** 4. It's easy to observe that $\mathcal{B}_3$ perfectly simulates **Game** $\mathsf{hy}.(y\text{-}1)$ or **Game** $\mathsf{hy}.y$. Moreover, $\mathcal{B}_3$ simulates **Game** $\mathsf{hy}.(y\text{-}1)$ if $c = \mathsf{SKE}_2.\mathsf{Enc}(\tilde{K}_2^y, \mathsf{pin}_U)$ and **Game** $\mathsf{hy}.y$ if $c = \mathsf{SKE}_2.\mathsf{Enc}(\tilde{K}^y, \tilde{\mathsf{pin}})$. Thus, $\mathcal{B}_3$ can win $\mathsf{IND\text{-}1CPA}$ experiment whenever $\mathcal{A}$ can distinguish **Game** $\mathsf{hy}.(y\text{-}1)$ and **Game** $\mathsf{hy}.y$. Thus, we have that

$$\Pr[E_3] \leq \epsilon_{\mathsf{SKE}_2}^{\mathsf{ind\text{-}1cpa}}$$

Moreover, **Game** $\mathsf{hy}.n_{2,2}$ have replaced all c_p in the $\textsc{Setup}$ oracles whenever the client chooses $\mathsf{puvProtocol}_2$. Thus, **Game** $\mathsf{hy}.n_{2,2}$ is identical to **Game** 12 and we have

$$\mathsf{Adv}_{12} = \mathsf{Adv}_{\mathsf{hy}.n_{2,2}}$$

Combing the statements above, we have that

$$\mathsf{Adv}_{11} - \mathsf{Adv}_{12} \leq n_{2,2}\epsilon_{\mathsf{SKE}_2}^{\mathsf{ind\text{-}1cpa}}$$

Here, we simply keep using the number $n_{2,2}$, which would be helpful for us to tighten our security upper bound in the following games.

Game 13. This games is identical to **Game** 12 except the following modification:

1. When $\mathcal{A}$ queries $\textsc{Setup}(T, i, C, j, U)$ and the challenger sets the selected Pin/Uv Auth Protocol of the client session π_C^j to be $\pi_C^j.\mathsf{selectedpuvProtocol} = \mathsf{puvProtocol}_3$ during the execution, the challenger replaces $c_p = \mathsf{SKE}_3.\mathsf{Enc}(K_2, \mathsf{pin}_U)$ in $\mathsf{setPIN}\text{-}C(\pi_C^j, \mathsf{pin}_U)$ by $\tilde{c}_p \leftarrow \mathsf{SKE}_3.\mathsf{Enc}(K_2, \tilde{\mathsf{pin}})$, where $\tilde{\mathsf{pin}}$ was sampled in **Game** 10.

Let $n_{2,3}$ denote the number of K_2 sampled in $\textsc{Setup}$ oracle when the underlying Pin/Uv Auth Protocol is $\mathsf{puvProtocol}_3$. Similar to the analysis in **Game** 12, we can easily have that

$$\mathsf{Adv}_{12} - \mathsf{Adv}_{13} \leq n_{2,3}\epsilon_{\mathsf{SKE}_3}^{\mathsf{ind\text{-}1cpa}}$$

Note that $n_{2,1}$, $n_{2,2}$, and $n_{2,3}$ respectively denote the number of symmetric encryption keys produced by $\mathsf{puvProtocol}_1$, $\mathsf{puvProtocol}_2$, and $\mathsf{puvProtocol}_3$ in the $\textsc{Setup}$ oracle. Moreover, our CTAP 2.1 only supports these three versions. This implies that $n_{2,1} + n_{2,2} + n_{2,3} \leq q_{\textsc{Setup}}$. Further, we have that

$$\begin{aligned}\mathsf{Adv}_{10} - \mathsf{Adv}_{13} &\leq n_{2,1}\epsilon_{\mathsf{SKE}_1}^{\mathsf{ind\text{-}1cpa\text{-}H_2}} + n_{2,2}\epsilon_{\mathsf{SKE}_2}^{\mathsf{ind\text{-}1cpa}} + n_{2,3}\epsilon_{\mathsf{SKE}_3}^{\mathsf{ind\text{-}1cpa}} \\ &\leq q_{\textsc{Setup}} \max(\epsilon_{\mathsf{SKE}_1}^{\mathsf{ind\text{-}1cpa\text{-}H_2}}, \epsilon_{\mathsf{SKE}_2}^{\mathsf{ind\text{-}1cpa}}, \epsilon_{\mathsf{SKE}_3}^{\mathsf{ind\text{-}1cpa}})\end{aligned}$$

Game 14. This game is identical to **Game** 13 except the following modification:

1. Whenever the attacker $\mathcal{A}$ queries $\textsc{Send-Bind-T}(T, i, m)$ oracle, instead of checking the decrypted $\mathsf{pinHash} \neq \mathsf{st}_T.\mathsf{pinHash}$ in the $\mathsf{obtainPinUvAuthToken}\text{-}T$ algorithm, the challenger checks whether $\mathsf{pinHash} \neq \mathsf{H}(\mathsf{pin}_{\mathsf{st}_T.\mathsf{user}})$

Note that $\mathsf{st}_T.\mathsf{pinHash} = \mathsf{H}(\mathsf{pin}_{\mathsf{st}_T.\mathsf{user}})$. **Game** 13 and **Game** 14 are indeed identical and we have that:

$$\mathsf{Adv}_{13} = \mathsf{Adv}_{14}$$

Game 15. This games is identical to **Game** 14 except the following modification:

1. When $\mathcal{A}$ queries $\textsc{Execute}(T, i, C, j, U)$ and the challenger sets the selected Pin/Uv Auth Protocol of the client sessions π_C^j to be $\pi_C^j.\mathsf{selectedpuvProtocol} = \mathsf{puvProtocol}_1$, the challenger replaces $c_{ph} = \mathsf{SKE}_1.\mathsf{Enc}(\tilde{K}, \mathsf{H}(\mathsf{pin}_U))$ in the $\mathsf{obtainPinUvAuthToken}\text{-}C\text{-}\mathsf{start}(\pi_C^j, \mathsf{pin}_U)$ by $\tilde{c}_{ph} \leftarrow \mathsf{SKE}_1.\mathsf{Enc}(\tilde{K}, \mathsf{H}(\tilde{\mathsf{pin}}))$, where $\tilde{K}$ is the underlying symmetric key produced by $\mathsf{puvProtocol}_1$ and that $\tilde{\mathsf{pin}}$ was sampled in **Game** 10.

We prove that **Game** 14 and **Game** 15 are indistinguishable by $n_{3,1}$ hybrid games, where $n_{3,1}$ denotes the number of $\tilde{K}$ sampled in $\textsc{Execute}$ oracle when the underlying Pin/Uv Auth Protocol is $\mathsf{puvProtocol}_1$. Let $\tilde{K}^y$ denotes the y-th $\tilde{K}$ sampled in $\textsc{Execute}$ oracle when the underlying Pin/Uv Auth Protocol is $\mathsf{puvProtocol}_1$. The hybrid game $\mathsf{hy}.y$ for $y \in [n_{3,1}]$ is defined below.

Game $\mathsf{hy}.0$**.** This game is identical to **Game** 14 and we have:

$$\mathsf{Adv}_{14} = \mathsf{Adv}_{\mathsf{hy}.0}$$

Game $\mathsf{hy}.y$**.** This game is identical to **Game** $\mathsf{hy}.(y\text{-}1)$ except the following modifications:

1. When $\mathcal{A}$ queries $\textsc{Execute}(T, i, C, j, U)$ that will produce the y-th $\tilde{K}^y$ for $\mathsf{puvProtocol}_1$, the challenger replaces $c_{ph} = \mathsf{SKE}_1.\mathsf{Enc}(\tilde{K}^y, \mathsf{H}(\mathsf{pin}_U))$ in $\mathsf{obtainPinUvAuthToken}\text{-}C\text{-}\mathsf{start}(\pi_C^j, \mathsf{pin}_U)$ by $\tilde{c}_{ph} \leftarrow \mathsf{SKE}_1.\mathsf{Enc}(\tilde{K}^y, \tilde{\mathsf{pin}})$.

Let event E_4 denote the probability that the attacker $\mathcal{A}$ can distinguish **Game** $\mathsf{hy}.(y\text{-}1)$ and **Game** $\mathsf{hy}.y$. It holds that

$$\mathsf{Adv}_{\mathsf{hy}.(y\text{-}1)} - \mathsf{Adv}_{\mathsf{hy}.y} \leq \Pr[E_4]$$

Then, we analyze the probability of the occurrence of E_4 by reduction. Namely, if E_4 occurs, then we can construct an attacker $\mathcal{B}_4$ that breaks $\mathsf{IND\text{-}1\$PA\text{-}LPC}$ security of SKE_1 by invoking $\mathcal{A}$. $\mathcal{B}_4$ simulates **Game** $\mathsf{hy}.(y\text{-}1)$ honestly, except for the following queries:

1. When $\mathcal{A}$ sends $\textsc{Execute}(T, i, C, j, U)$ that will produce the y-th $\tilde{K}^y$ for $\mathsf{puvProtocol}_1$ in this phase. To handle this query, $\mathcal{B}_4$ sends $(\mathsf{H}(\mathsf{pin}_{\tilde{U}}), \mathsf{H}(\tilde{\mathsf{pin}}))$ to its challenger and obtains $\tilde{c}$. Then, $\mathcal{B}_4$ sets $\tilde{c}$ as the output of $\mathsf{obtainPinUvAuthToken}\text{-}C\text{-}\mathsf{start}(\pi_C^j, \mathsf{pin}_{\tilde{U}})$ algorithm. The reaming of this query is answered honestly.
2. When $\mathcal{A}$ queries $\textsc{Send-Bind-T}(T, i, m)$ following the above $\textsc{Execute}(T, i, C, j, U)$ query, $\mathcal{B}_4$ separate the cases depending on whether $m = pk_{C,j} \parallel \tilde{c}_{ph}$.
   (a) If $\mathsf{st}_T.\mathsf{user} \neq U$, then $\mathcal{B}_4$ simply performs as if the decrypted $\mathsf{pinHash}$ is unequal to $\mathsf{H}(\mathsf{pin}_{\mathsf{st}_T.\mathsf{user}})$.
   (b) If $\mathsf{st}_T.\mathsf{user} = U$ and $m = pk_{C,j} \parallel \tilde{c}_{ph}$, then $\mathcal{B}_4$ queries $\textsc{Rand}$ with input $\mu\lambda$ to its challenger and obtains $(pt_0, pt_1, \tilde{c}')$. Then, $\mathcal{B}_4$ sets $(\tilde{c}', \mathsf{false})$ as the output of $\mathsf{obtainPinUvAuthToken}\text{-}T(\pi_T^i, \mathsf{puvProtocol}_1, c, c_{ph})$. Meanwhile, $\mathcal{B}_4$ sets $\pi_T^i.\mathsf{bs} = pt_0$.
   (c) If $\mathsf{st}_T.\mathsf{user} = U$ but $m = pk_{C,j} \parallel c_{ph}$ for $c_{ph} \neq \tilde{c}_{ph}$, then $\mathcal{B}_4$ queries $\textsc{Lpc}(c_{ph})$ to its challenger. If the response is false, then $\mathcal{B}_4$ performs as if the decrypted $\mathsf{pinHash}$ does not match $\mathsf{H}(\mathsf{pin}_{\mathsf{st}_T.\mathsf{user}})$. Otherwise, $\mathcal{B}_4$ queries $\textsc{Rand}$ with input $\mu\lambda$ to its challenger and obtains $(pt_0, pt_1, \tilde{c}')$. Then, $\mathcal{B}_4$ sets $(\tilde{c}', \mathsf{false})$ as the output of $\mathsf{obtainPinUvAuthToken}\text{-}T(\pi_T^i, \mathsf{puvProtocol}_1, c, c_{ph})$. Meanwhile, $\mathcal{B}_4$ sets $\pi_T^i.\mathsf{bs} = pt_0$.
3. When $\mathcal{A}$ afterwards sends $\textsc{Send-Bind-C}(C, j, m)$ following the above $\textsc{Execute}(T, i, C, j, \tilde{U})$ and $\textsc{Send-Bind-T}(T, i, m)$ queries without abortion, $\mathcal{B}$ sets $\pi_C^j.\mathsf{bs} = pt_0$ if $m = \tilde{c}'$, and $\pi_C^j.\mathsf{bs} = pt_1$ otherwise.

It's easy to observe that $\mathcal{B}_4$ perfectly simulates **Game** $\mathsf{hy}.(y\text{-}1)$ if $c_{ph} = \mathsf{SKE}_1.\mathsf{Enc}(\tilde{K}^y, \mathsf{H}(\mathsf{pin}_U))$ and **Game** $\mathsf{hy}.y$ if $c_{ph} = \mathsf{SKE}_1.\mathsf{Enc}(\tilde{K}^y, \mathsf{H}(\tilde{\mathsf{pin}}))$. Thus, $\mathcal{B}_4$ can win $\mathsf{IND\text{-}1\$PA}$ experiment whenever $\mathcal{A}$ can distinguish **Game** $\mathsf{hy}.(y\text{-}1)$ and **Game** $\mathsf{hy}.y$. Thus, we have

$$\Pr[E_4] \leq \epsilon^{\mathsf{ind\text{-}1\$pa\text{-}lpc}}_{\mathsf{SKE}_1}$$

Moreover, **Game** $\mathsf{hy}.n_{3,1}$ have replaced all c_{ph} in the EXECUTE oracles whenever the client chooses $\mathsf{puvProtocol}_1$. Thus, **Game** $\mathsf{hy}.n_{3,1}$ is identical to **Game** 15 and we have

$$\mathsf{Adv}_{15} = \mathsf{Adv}_{\mathsf{hy}.n_{3,1}}$$

Note that we have assumed all $\tilde{K}$ of $\mathsf{puvProtocol}_1$ are distinct in **Game** 2 and all above hybrid games are independent. Combing the statements above, we have that

$$\mathsf{Adv}_{14} - \mathsf{Adv}_{15} \leq n_{3,1}\epsilon^{\mathsf{ind\text{-}1\$pa\text{-}lpc}}_{\mathsf{SKE}_1}$$

Similar as before, we simply keep using the number $n_{3,1}$, which would be helpful for us to tighten our security upper bound in the following games.

Game 16. This game is identical to **Game** 15 except the following modification:

1. When $\mathcal{A}$ queries $\textsc{Execute}(T, i, C, j, U)$ and the challenger sets the Pin/Uv Auth Protocol of the client session π^j_C to be $\pi^j_C.\mathsf{selectedpuvProtocol} = \mathsf{puvProtocol}_2$, the challenger replaces $c_{ph} \xleftarrow{\$} \mathsf{SKE}_2.\mathsf{Enc}(\tilde{K}_2, \mathsf{H}(\mathsf{pin}_U))$ in the $\mathsf{obtainPinUvAuthToken}\text{-}C\text{-}\mathsf{start}(\pi^j_C, \mathsf{pin}_U)$ by $\tilde{c}_{ph} \xleftarrow{\$} \mathsf{SKE}_2.\mathsf{Enc}(\tilde{K}_2, \mathsf{H}(\tilde{\mathsf{pin}}))$, where $\tilde{K}_2$ is the underlying symmetric key produced by $\mathsf{puvProtocol}_2$ and that $\tilde{\mathsf{pin}}$ was sampled in **Game** 10.

Similar to the analysis in **Game** 15, let $n_{3,2}$ denotes the number of $\tilde{K}_2$ of $\mathsf{authenticate}_2$ that are generated in $\textsc{Execute}(T, i, C, j, U)$ oracles. We can easily have the equation below by a sequence of hybrid games.

$$\mathsf{Adv}_{15} - \mathsf{Adv}_{16} \leq n_{3,2}\epsilon^{\mathsf{ind\text{-}1\$pa\text{-}lpc}}_{\mathsf{SKE}_2}$$

Game 17. This games is identical to **Game** 16 except the following modification:

1. When $\mathcal{A}$ queries $\textsc{Execute}(T, i, C, j, U)$ and the challenger sets the Pin/Uv Auth Protocol of the client session π^j_C to be $\pi^j_C.\mathsf{selectedpuvProtocol} = \mathsf{puvProtocol}_3$, the challenger replaces $c_{ph} \xleftarrow{\$} \mathsf{SKE}_3.\mathsf{Enc}(K_2, \mathsf{H}(\mathsf{pin}_U))$ in the $\mathsf{obtainPinUvAuthToken}\text{-}C\text{-}\mathsf{start}(\pi^j_C, \mathsf{pin}_U)$ by $\tilde{c}_{ph} \xleftarrow{\$} \mathsf{SKE}_3.\mathsf{Enc}(K_2, \mathsf{H}(\tilde{\mathsf{pin}}))$, where K_2 is the underlying symmetric key produced by $\mathsf{puvProtocol}_3$ and that $\tilde{\mathsf{pin}}$ was sampled in **Game** 10.

Similar to the analysis in **Game** 16, let $n_{3,3}$ denotes the number of K_2 of $\mathsf{authenticate}_3$ that are generated in $\textsc{Execute}(T, i, C, j, U)$ oracles. We can easily have the equation below by a sequence of hybrid games.

$$\mathsf{Adv}_{16} - \mathsf{Adv}_{17} \leq n_{3,3}\epsilon^{\mathsf{ind\text{-}1\$pa\text{-}lpc}}_{\mathsf{SKE}_3}$$

Note that there are only 3 kinds of $\mathsf{puvProtocols}$. We have $n_{3,1} + n_{3,2} + n_{3,3} \leq q_{\textsc{Execute}}$. It holds that

$$\begin{aligned}\mathsf{Adv}_{14} - \mathsf{Adv}_{17} \leq & n_{3,1}\epsilon^{\mathsf{ind\text{-}1\$pa\text{-}lpc}}_{\mathsf{SKE}_1} + n_{3,2}\epsilon^{\mathsf{ind\text{-}1\$pa\text{-}lpc}}_{\mathsf{SKE}_2} + n_{3,3}\epsilon^{\mathsf{ind\text{-}1\$pa\text{-}lpc}}_{\mathsf{SKE}_3} \\ \leq & q_{\textsc{Execute}} \max(\epsilon^{\mathsf{ind\text{-}1\$pa\text{-}lpc}}_{\mathsf{SKE}_1}, \epsilon^{\mathsf{ind\text{-}1\$pa\text{-}lpc}}_{\mathsf{SKE}_2}, \epsilon^{\mathsf{ind\text{-}1\$pa\text{-}lpc}}_{\mathsf{SKE}_3})\end{aligned}$$

Game 18. In this game, the challenger $\mathcal{C}$ aborts the game and let $\mathcal{A}$ immediately win if there exists a token session π^i_T that accepts a malicious c_{ph} sent by $\mathcal{A}$ via $\textsc{Send-Bind-T}$ query without corrupting the pin of the user $\mathsf{st}_T.\mathsf{user}$. More formally and concretely, the challenger $\mathcal{C}$ aborts the game and let $\mathcal{A}$ immediately win if there exists a token session π^i_T such that

1. the attacker has queried $\textsc{Send-Bind-T}(T, i, m)$ such that $m = pk \parallel c_{ph}$ is not included in the output of any query $\textsc{Execute}(T, i, C, j, U)$ for any C, j, U.
2. $\pi^i_T.\mathsf{pinCorr} = \mathsf{false}$
3. $\pi^i_T.\mathsf{st}_{\mathsf{exe}} = \mathsf{bindDone}$ and $\pi^i_T.\mathsf{bs} \neq \perp$

In this case, the input message m of $\textsc{Send-Bind-T}$ is forged by $\mathcal{A}$. Note that all the transcripts of a token T that $\mathcal{A}$ eavesdrops are independent of pin_U with $\mathsf{pin}_U \neq \tilde{\mathsf{pin}}$ and that $\mathcal{A}$ is not allowed to corrupt the user pin $\mathsf{pin}_{\mathsf{st}_T.\mathsf{user}}$ that setups token T. The condition "$\pi^i_T.\mathsf{st}_{\mathsf{exe}} = \mathsf{bindDone}$ and $\pi^i_T.\mathsf{bs} \neq \perp$" indicates that the attacker $\mathcal{A}$ must encrypt $\mathsf{H}(\mathsf{pin}_{\mathsf{st}_T.\mathsf{user}})$. Recall that the pin_U of any honest users U are sampled randomly following distribution $\mathfrak{D}$ with min-entropy $\alpha_{\mathfrak{D}}$ and that $\mathcal{A}$ can try at most $\mathsf{pinRetriesMax}$ times for each token session π^i_T. $\mathcal{A}$ can guess the $\mathsf{pin}_{\mathsf{st}_T.\mathsf{user}}$ for each token session π^i_T correctly with probability at most $\mathsf{pinRetriesMax}2^{-\alpha_{\mathfrak{D}}}$. Note also that tokens can be set pin only in $\textsc{Setup}$ oracles, which happens at most $q_{\textsc{Setup}}$ times. By union bound, we have that

$$\mathsf{Adv}_{17} - \mathsf{Adv}_{18} \leq q_{\textsc{Setup}}\mathsf{pinRetriesMax}2^{-\alpha_{\mathfrak{D}}}$$

Final Analysis. Now, let's finally check $\mathcal{A}$ can satisfy the winning conditions.

Note that the $\mathsf{win\text{-}SUF\text{-}t'}$ is set to true in the $\textsc{Validate}(T, i, M, t, d)$ query only when at least one of the following four winning conditions

1. the user decision $d \neq \mathsf{accepted}$, or
2. two distinct client sessions that completed Bind have the same session identifiers, or
3. two distinct token sessions that completed Bind have the same session identifiers, or

4. (M, t) was not output by any of π_T^i's uncompromised valid partners π_C^j before the setup user of token T is corrupted

However, we can observe that

1. $d \neq \mathsf{accepted}$: This is always false by definition, see $\mathsf{validate}\text{-}T$ algorithm in Figure 5.8.
2. $\exists(C_1, j_1), (C_2, j_2)$ such that $(C_1, j_1) \neq (C_2, j_2)$ **and** $\pi_{C_1}^{j_1}.\mathsf{st}_{\mathsf{exe}} = \pi_{C_2}^{j_2}.\mathsf{st}_{\mathsf{exe}} = \mathsf{bindDone}$ **and** $\pi_{C_1}^{j_1}.\mathsf{sid} = \pi_{C_2}^{j_2}.\mathsf{sid}$: Recall that the client sends the randomly sampled $\mathsf{puvProtocol}_i.pk$, which includes ECDH public key, to the tokens in the EXECUTE oracles. Recall also that we have ensured that all honestly sampled ECDH public keys are distinct and that the session identifier is defined as the full transcript during the execution of Bind algorithm. This means, two client sessions can never have the same identifier (not matter whether they are valid or not). Thus, this condition is always false.
3. $\exists(T_1, i_1), (T_2, i_2)$ such that $(T_1, i_1) \neq (T_2, i_2)$ **and** $\pi_{T_1}^{i_1}.\mathsf{st}_{\mathsf{exe}} = \pi_{T_2}^{i_2}.\mathsf{st}_{\mathsf{exe}} = \mathsf{bindDone}$ **and** $\pi_{T_1}^{i_1}.\mathsf{sid} = \pi_{T_2}^{i_2}.\mathsf{sid}$: Note that session identifiers of token sessions includes the token's public key pk, the client's encapsulation c, the encryption of pinHash c_{ph}, and the encryption of pt c_{pt}. Then, $\pi_{T_1}^{i_1}.\mathsf{sid} = \pi_{T_2}^{i_2}.\mathsf{sid}$ holds only when $(pk^1, c^1, c_{ph}^1, c_{pt}^1) = (pk^2, c^2, c_{ph}^2, c_{pt}^2)$, where $(pk^1, c^1, c_{ph}^1, c_{pt}^1)$ is included in the $\pi_{T_1}^{i_1}.\mathsf{sid}$ and $(pk^2, c^2, c_{ph}^2, c_{pt}^2)$ is included in the $\pi_{T_2}^{i_2}.\mathsf{sid}$. In particular, $(pk^1, c^1) = (pk^2, c^2)$ indicates that c_{pt}^1 and c_{ph}^2 are encrypted under the same symmetric key. Moreover, in **Game** 10 we ensured that there exists no collision between pts, which further implies that $c_{pt}^1 \neq c_{pt}^2$ if the underlying symmetric encryption is correct. Thus, this condition is always false.
4. for $(C', j') \leftarrow \mathsf{bindPartner}(T, i)$, all of the following conditions must hold: (a) $(C', j', M, t) \notin \mathcal{L}_{\text{AUTH}}$, (b) $\pi_{C'}^{j'} = (\perp, \perp)$ **or** $\pi_{C'}^{j'}.\mathsf{compromised} = \mathsf{false}$, (c) $\pi_T^i.\mathsf{pinCorr} = \mathsf{false}$: According to the condition (2), we know that the attacker $\mathcal{A}$ is not allowed to compromise the binding state of any client (C', j') such that $\pi_{C'}^{j'}.\mathsf{bs} = \pi_T^i.\mathsf{bs}$ According to the condition (3)$\pi_T^i.\mathsf{pinCorr} = \mathsf{false}$ and **Game** 18, we know that the attacker $\mathcal{A}$ cannot execute active attack against token to obtain the token biding state $\pi_T^i.\mathsf{bs}$. Thus, the attacker $\mathcal{A}$ has no idea about the $\pi_T^i.\mathsf{bs}$.

   According to the condition (1) $(C', j', M, t) \notin \mathcal{L}_{\text{AUTH}}$, we know that (M, t) was never output by any of the session π_T^i's partner. The attacker therefore has to forge the message-tag pair (M, t). Recall that the tag t is computed by applying random oracles H_2, H_4, and H_7 to the corresponding binding state $\pi_T^i.\mathsf{bs}$ and message M, respectively in $\mathsf{puvProtocol}_1$, $\mathsf{puvProtocol}_2$, and $\mathsf{puvProtocol}_3$. The attacker can only guess the either the tag directly or the toke binding state $\pi_T^i.\mathsf{bs}$. Moreover, recall that:

   (a) If the underlying authProtocol is $\mathsf{puvProtocol}_1$, then the attacker $\mathcal{A}$ can guess $\pi_T^i.\mathsf{bs}$ with probability $2^{-\mu\lambda}$ and tag t with probability 2^{-l_2}

(b) If the underlying authProtocol is $\mathsf{puvProtocol}_2$, then the attacker $\mathcal{A}$ can guess $\pi_T^i.\mathsf{bs}$ with probability $2^{-2\lambda}$ and tag t with probability 2^{-l_4}.

(c) If the underlying authProtocol is $\mathsf{puvProtocol}_3$, then the attacker $\mathcal{A}$ can guess $\pi_T^i.\mathsf{bs}$ with probability $2^{-\mu'\lambda}$ and the tag with probability 2^{-l_7}.

Thus, the probability that $\mathcal{A}$ finds can forge tag t for any message M in each VALIDATE query is bounded by

$$\max(2^{-\mu\lambda}, 2^{-l_2}, 2^{-2\lambda}, 2^{-l_4}, 2^{-\mu'\lambda}, 2^{-l_7}) = 2^{-\min(\mu\lambda, 2\lambda, \mu'\lambda, l_2, l_4, l_7)}$$

Note that the attacker $\mathcal{A}$ can attempts only in VALIDATE oracle, which can be invoked at most $q_{\textsc{Validate}}$ times. By union bound, we have that

$$\mathsf{Adv}_{18} \leq q_{\textsc{Validate}} 2^{-\min(\mu\lambda, 2\lambda, \mu'\lambda, l_2, l_4, l_7)}$$

Combing all statements above, the proof is concluded by:

$$\begin{aligned}
\mathsf{Adv}_{\mathsf{ePACA}}^{\mathsf{SUF\text{-}t'}}(\mathcal{A}) \leq & (q_{\textsc{Setup}} + q_{\textsc{Execute}})\epsilon_{\mathsf{ECDH}}^{\mathsf{sCDH}} + \epsilon_{\mathsf{H}}^{\mathsf{coll\text{-}res}} \\
& + \binom{q_{\textsc{Setup}} + q_{\textsc{Execute}}}{2}(2^{2-\min(l_1, l_3, l_5, l_6)} + 2^{1-q}) \\
& + q_{\textsc{NewU}} 2^{-\alpha_{\mathfrak{D}}} + \binom{q_{\textsc{Send-Bind-T}}}{2} 2^{-\min(\mu, 2, \mu')\lambda} \\
& + q_{\textsc{Setup}} \max(\epsilon_{\mathsf{SKE}_1}^{\mathsf{ind\text{-}1cpa\text{-}H_2}}, \epsilon_{\mathsf{SKE}_2}^{\mathsf{ind\text{-}1cpa}}, \epsilon_{\mathsf{SKE}_3}^{\mathsf{ind\text{-}1cpa}}) \\
& + q_{\textsc{Execute}} \max(\epsilon_{\mathsf{SKE}_1}^{\mathsf{ind\text{-}1\$pa\text{-}lpc}}, \epsilon_{\mathsf{SKE}_2}^{\mathsf{ind\text{-}1\$pa\text{-}lpc}}, \epsilon_{\mathsf{SKE}_3}^{\mathsf{ind\text{-}1\$pa\text{-}lpc}}) \\
& + q_{\textsc{Setup}} \mathsf{pinRetriesMax} 2^{-\alpha_{\mathfrak{D}}} \\
& + q_{\textsc{Validate}} 2^{-\min(\mu\lambda, 2\lambda, \mu'\lambda, l_2, l_4, l_7)}
\end{aligned}$$

□

5.9.5 Proof of Theorem 23

Proof. We give the proof by a sequence of games. Each game is simulated between a challenger $\mathcal{C}$ and an attacker $\mathcal{A}$. Let Adv_i denote the attacker $\mathcal{A}$'s advantage in winning game i.

Game 0. This game is identical to the $\mathrm{Expr}_{\mathsf{ePACA}}^{\mathsf{SUF\text{-}t'}}$ experiment. Hence, it holds that

$$\mathsf{Adv}_0 = \mathsf{Adv}_{\mathsf{ePACA}}^{\mathsf{SUF\text{-}t'}}(\mathcal{A})$$

Game 1. This game is identical to **Game** 0 except that the following modifications:

1. Whenever a client executes $\mathsf{puvProtocol}_3.\mathsf{encapsulate}_3$ on a token's public key pk', the challenger $\mathcal{C}$ executes the following steps:

   (a) Parse $(pk'_1, pk'_2) \leftarrow pk'$

(b) Run $(c_2, Z_2) \xleftarrow{\$} \mathsf{KEM.KEM.Encaps}(pk'_2)$

(c) Sample a random $\tilde{Z}_2$ in the key space of KEM

(d) Replace Z_2 by $\tilde{Z}_2$ for the subsequent execution.

(e) Finally, output a ciphertext $c = (c_1, c_2)$ for some c_1.

2. Whenever a token holding $pk'' = (pk''_1, pk''_2)$ such that $pk'_2 = pk''_2$ and needs to execute $\mathsf{decapsulate}_3$ on $c'' = (c''_1, c''_2)$ such that $c''_2 = c_2$, the challenger executes $\mathsf{decapsulate}_3$ honestly except that $\mathcal{C}$ directly sets $Z_2 \leftarrow \tilde{Z}_2$.

We prove that $\mathcal{A}$ cannot distinguishable **Game** 0 and **Game** 1 by n hybrid games, where n denotes the number of encapsulations that was output by all tokens in the Setup and Execute oracles. Then, we have that $n \leq q_{\text{Setup}} + q_{\text{Execute}}$. Let (sk_T^y, pk_T^y) denote the y-th KEM public-private key pair among all tokens. The **Game** $\mathsf{hy}.y$ for $y \in [n]$ is defined as follows:

Game $\mathsf{hy}.0$. This game is identical to **Game** 0 and we have that

$$\mathsf{Adv}_0 = \mathsf{Adv}_{\mathsf{hy}.0}$$

Game $\mathsf{hy}.y$. This game is identical to **Game** $\mathsf{hy}.(y\text{-}1)$ except that the following modifications:

1. Whenever $\mathcal{A}$ queries Setup and Execute oracles where $\mathcal{C}$ needs to returns y-th KEM public key pk_T^y among all tokens and executes $\mathsf{KEM.Encaps}(pk_T^y)$, the challenger executes $(c_2, Z_2) \xleftarrow{\$} \mathsf{KEM.Encaps}(pk_T^y)$ and samples $\tilde{Z}_2$ in the key space of KEM. Next, $\mathcal{C}$ replaces Z_2 by $\tilde{Z}_2$ for the subsequent execution.
2. Whenever $\mathcal{C}$ needs to execute $\mathsf{KEM.Decaps}(sk_T^y, c_2)$, it directly uses $Z_2 \leftarrow \tilde{Z}_2$ for the subsequent execution instead of computing Z_2 using KEM.

If $\mathcal{A}$ can distinguish **Game** $\mathsf{hy}.y$ from **Game** $\mathsf{hy}.(y\text{-}1)$, then we can construct an attacker $\mathcal{B}_1$ that breaks $\mathsf{IND\text{-}CCA}$ security of KEM. The $\mathsf{IND\text{-}CCA}$ experiment executes $(pk, sk) \xleftarrow{\$} \mathsf{KEM.KGen}()$ and $(c^\star, k_0^\star) \xleftarrow{\$} \mathsf{KEM.Encaps}(pk)$ honestly and samples $\mathsf{b} \xleftarrow{\$} \{0, 1\}$ and $k_1^\star$ from the key space $\mathcal{K}$ randomly. On input $(pk, c^\star, k_{\mathsf{b}}^\star)$, $\mathcal{B}_1$ runs **Game** $\mathsf{hy}.(y\text{-}1)$ honestly except the following modification:

1. When the algorithm $\mathsf{obtainSharedSecret}\text{-}T$ needs to output y-th KEM public key pk_T^y, $\mathcal{B}_1$ uses $pk_T^y \leftarrow pk$ instead of sampling it using $\mathsf{KEM.KGen}()$

2. When $\mathcal{B}_1$ needs to execute $\mathsf{KEM.Encaps}(pk_T^y)$ in $\mathsf{encapsulate}_3$, $\mathcal{B}_1$ simply uses $(c_2, Z_2) \leftarrow (c^\star, k_{\mathsf{b}}^\star)$ for the subsequent execution.

3. When $\mathcal{B}_1$ needs to execute $Z_2 \leftarrow \mathsf{KEM.Decaps}(sk_T^y, c)$ in $\mathsf{decapsulate}_3$ algorithm, $\mathcal{B}_1$ does not know sk_T^y and performs as follows instead:

- If $c = c^\star$, then $\mathcal{B}_1$ simply uses $Z_2 \leftarrow k^\star_{\mathsf{b}}$.
- If $c \neq c^\star$, then $\mathcal{B}_1$ queries its decapsulation oracle $\mathcal{O}_{\mathsf{Decaps}}$ on c. When receiving an answer k, $\mathcal{B}_1$ sets $Z_2 \leftarrow k$ for the remaining computation.

It is straightforward that $\mathcal{B}_1$ perfectly simulates **Game** $\mathsf{hy}.(y\text{-}1)$ if $\mathsf{b} = 0$ and **Game** $\mathsf{hy}.y$ if $\mathsf{b} = 1$. So, $\mathcal{B}_1$ can win $\mathsf{IND\text{-}CCA}$ experiment whenever $\mathcal{A}$ can distinguish **Game** $\mathsf{hy}.(y\text{-}1)$ and **Game** $\mathsf{hy}.y$. Thus, it holds that

$$\mathsf{Adv}_{\mathsf{hy}.(y\text{-}1)} - \mathsf{Adv}_{\mathsf{hy}.y} \leq \epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}}$$

Moreover, when all the encapsulated keys of KEM are replaced by random keys, **Game** $\mathsf{hy}.n$ is identical to **Game** 1. Then, we have that

$$\mathsf{Adv}_{\mathsf{hy}.n} = \mathsf{Adv}_1$$

Note that all hybrid games above are independent, by union bound, we have that

$$\mathsf{Adv}_0 - \mathsf{Adv}_1 \leq n\epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}} \leq (q_{\text{Setup}} + q_{\text{Execute}})\epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}}$$

Game 2. This game is identical to **Game** 1 except the following modification:

1. The challenger replaces each function $\mathsf{H}_5(\cdot, \tilde{Z}_2)$ by a truly random function $f_{\tilde{Z}_2}$, where $\tilde{Z}_2$s are sampled in **Game** 1.

We can easily reduce the indistinguishability between **Game** 1 and **Game** 2 to the $\epsilon^{\mathsf{swap}}_{\mathsf{H}_5}$-$\mathsf{swap}$ security of H_5 in n hybrid games, where n denotes the number of $\tilde{Z}_2$ in **Game** 1. Obviously, it holds that $n \leq q_{\text{Setup}} + q_{\text{Execute}}$. Thus, we have that

$$\mathsf{Adv}_1 - \mathsf{Adv}_2 \leq (q_{\text{Setup}} + q_{\text{Execute}})\epsilon^{\mathsf{swap}}_{\mathsf{H}_5}$$

In particular, for any $\tilde{Z} \leftarrow f_{\tilde{Z}_2}(Z_1)$, we know that $\tilde{Z}$ is uniformly at random, since $f_{\tilde{Z}_2}$ is a truly random function for any $\tilde{Z}_2$ that is sampled by the challenger in **Game** 1 and not leaked to the attacker $\mathcal{A}$.

Game 3. This game is identical to **Game** 2 except the following modification:

1. The challenger replaces each function $\mathsf{H}_6(\tilde{Z}, \cdot)$ by a truly random function $f'_{\tilde{Z}}$, where $\tilde{Z}$s are derived in **Game** 2.

Similarly to **Game** 2, we can easily reduce the indistinguishability between **Game** 2 and **Game** 3 to the $\epsilon^{\mathsf{prf}}_{\mathsf{H}_6}$-$\mathsf{prf}$ security of H_6 in n hybrid games, where n denotes the number of $\tilde{Z}$ produced in **Game** 2. Obviously, we have that $n \leq q_{\text{Setup}} + q_{\text{Execute}}$. Thus, we have that

$$\mathsf{Adv}_2 - \mathsf{Adv}_3 \leq (q_{\text{Setup}} + q_{\text{Execute}})\epsilon^{\mathsf{prf}}_{\mathsf{H}_6}$$

In particular, for any $K_1 \leftarrow f'_{\tilde{Z}}$("CTAP2 HMAC key") and $K_2 \leftarrow f'_{\tilde{Z}}$("CTAP2 AES key"), we know that K_1s and K_2s are uniformly at random, since $f'_{\tilde{Z}}$ is a truly random function in **Game** 2 and $\tilde{Z}$s are not leaked to the attacker $\mathcal{A}$.

Game 4. In this game, the challenger $\mathcal{C}$ aborts and lets $\mathcal{A}$ immediately win if there exists collision between $\tilde{K}_1$s or $\tilde{K}_2$s in **Game** 3. Note that $\tilde{K}_1$ as well as $\tilde{K}_2$ is derived only in the Setup and Execute oracles. So, there are at most $q_{\textsc{Setup}} + q_{\textsc{Execute}}$ keys and $\binom{q_{\textsc{Setup}}+q_{\textsc{Execute}}}{2}$ pairs. The collision of every two keys happens $\tilde{K}_1$ with probability 2^{-l_6}. The same holds for $\tilde{K}_2$.

Thus, we have that

$$\mathsf{Adv}_3 - \mathsf{Adv}_4 \leq \binom{q_{\textsc{Setup}} + q_{\textsc{Execute}}}{2} 2^{1-l_6}$$

Game 5. In this game, the challenger $\mathcal{C}$ aborts and lets $\mathcal{A}$ immediately win if there exists two distinct inputs pin, pin' during $\mathcal{C}$'s execution such that $\mathsf{H}(\mathsf{pin}) = \mathsf{H}(\mathsf{pin}')$. Note that this abortion indicates the violation of collision resistance of H by definition. Since H is $\epsilon_{\mathsf{H}}^{\mathsf{coll\text{-}res}}$-collision resistant, we have that

$$\mathsf{Adv}_4 - \mathsf{Adv}_5 \leq \epsilon_{\mathsf{H}}^{\mathsf{coll\text{-}res}}$$

Game 6. This game is identical to **Game** 5 except that the following modifications:

1. The challenger $\mathcal{C}$ samples a random $\tilde{\mathsf{pin}} \xleftarrow{\$} \mathfrak{D}$ at the beginning of the game but never uses it. The challenger aborts and lets $\mathcal{A}$ immediately win if $\tilde{\mathsf{pin}}$ collides with any user pin pin_U for any user U.
2. Whenever the attacker $\mathcal{A}$ queries oracle Setup inputting any (T, i, C, j, U), the challenger replaces $\mathsf{pin} \leftarrow \mathsf{st}_T.\mathsf{puvProtocol.decrypt}(K, c_p)$ in the $\mathsf{setPIN}\text{-}T$ algorithm by
$$\mathsf{pin} \leftarrow \mathsf{pin}_U$$
3. The challenger aborts the game and lets $\mathcal{A}$ immediately win if there exits a collision between pts used in Send-Bind-T oracles.

The analysis for this game is also identical to the **Game** 10 in the proof of Theorem 22. The only difference is that each pt is sampled only in $\{0,1\}^{\mu'\lambda}$ and there are at most $\binom{q_{\textsc{Send-Bind-T}}}{2}$ pairs of used pts.

So, we have that

$$\mathsf{Adv}_5 - \mathsf{Adv}_6 \leq q_{\textsc{NewU}} 2^{-\alpha_{\mathfrak{D}}} + \binom{q_{\textsc{Send-Bind-T}}}{2} 2^{-\mu'\lambda}$$

Game 7. This game is identical to **Game** 6 except the following modification:

1. When $\mathcal{A}$ queries $\textsc{Setup}(T, i, C, j, U)$ and the challenger $\mathcal{C}$ replaces $c_p \leftarrow \mathsf{SKE}_3.\mathsf{Enc}(K_2, \mathsf{pin}_U)$ in $\mathsf{setPIN}\text{-}C(\pi_C^j, \mathsf{pin}_U)$ by $\tilde{c}_p \leftarrow \mathsf{SKE}_3.\mathsf{Enc}(K_2, \tilde{\mathsf{pin}})$, where K_2 is the corresponding random key derived in **Game** 3 and $\tilde{\mathsf{pin}}$ is sampled in **Game** 6.

Similar to the discussion in **Game** 13 in the proof of Theorem 22, we have that

$$\mathsf{Adv}_6 - \mathsf{Adv}_7 \leq q_{\textsc{Setup}} \epsilon^{\mathsf{ind\text{-}1cpa}}_{\mathsf{SKE}_3}$$

Game 8. This game is identical to **Game** 7 except the following modification:

1. When $\mathcal{A}$ queries $\textsc{Execute}(T, i, C, j, U)$, the challenger replaces $c_{ph} \leftarrow \mathsf{SKE}_3.\mathsf{Enc}(K_2, \mathsf{H}(\mathsf{pin}_{\tilde{U}}))$ in obtainPinUvAuthToken-C-start by $\tilde{c}_{ph} \leftarrow \mathsf{SKE}_3.\mathsf{Enc}(K_2, \mathsf{H}(\tilde{\mathsf{pin}}))$, where K_2 is the corresponding key computed in **Game** 3 and $\tilde{\mathsf{pin}}$ is the one sampled in **Game** 6.

Similar to the discussion in **Game** 17 in the proof of Theorem 22, we have that

$$\mathsf{Adv}_7 - \mathsf{Adv}_8 \leq q_{\textsc{Execute}} \epsilon^{\mathsf{ind\text{-}1\$pa\text{-}lpc}}_{\mathsf{SKE}_3}$$

Game 9. In this game, the challenger $\mathcal{C}$ aborts the game and let $\mathcal{A}$ immediately win if there exists a token session π^i_T that accepts a malicious c_{ph} sent by $\mathcal{A}$ via $\textsc{Send-Bind-T}$ query without corrupting the pin of the user $\mathsf{st}_T.\mathsf{user}$. More formally and concretely, the challenger $\mathcal{C}$ aborts the game and let $\mathcal{A}$ immediately win if there exists a token session π^i_T such that

1. the attacker has queried $\textsc{Send-Bind-T}(T, i, m)$ such that $m = pk \parallel c_{ph}$ is not included in the output of any query $\textsc{Execute}(T, i, C, j, U)$ for any C, j, U.
2. $\pi^i_T.\mathsf{pinCorr} = \mathsf{false}$
3. $\pi^i_T.\mathsf{st}_{\mathsf{exe}} = \mathsf{bindDone}$ and $\pi^i_T.\mathsf{bs} \neq \perp$

The analysis for this game is identical to the one in **Game** 18 in the proof of Theorem 22. Thus, we can easily have that

$$\mathsf{Adv}_8 - \mathsf{Adv}_9 \leq q_{\textsc{Setup}} \mathsf{pinRetriesMax} 2^{-\alpha_{\mathfrak{D}}}$$

Final Analysis. Now, let's finally check $\mathcal{A}$ can satisfy the winning conditions.

Note that the win-SUF-t′ is set to true in the $\textsc{Validate}(T, i, M, t, d)$ query only when at least one of the following four winning conditions

1. the user decision $d \neq \mathsf{accepted}$, or
2. two distinct client sessions that completed Bind have the same session identifiers, or
3. two distinct token sessions that completed Bind have the same session identifiers, or
4. (M, t) was not output by any of π^i_T's uncompromised valid partners π^j_C before the setup user of token T is corrupted

However, we can observe that

1. $d \neq \mathsf{accepted}$: This is always false by definition, see validate-T algorithm in Figure 5.8.

2. $\exists (C_1, j_1), (C_2, j_2)$ such that $(C_1, j_1) \neq (C_2, j_2)$ **and** $\pi_{C_1}^{j_1}.\mathsf{st}_{\mathsf{exe}} = \pi_{C_2}^{j_2}.\mathsf{st}_{\mathsf{exe}} = \mathsf{bindDone}$ **and** $\pi_{C_1}^{j_1}.\mathsf{sid} = \pi_{C_2}^{j_2}.\mathsf{sid}$: Recall that the client receives the honest token's public key and sends the encapsulation to the tokens in the EXECUTE oracles. Note that the KEM has public key entropy α_{pk} and ciphertext entropy α_c. Note also that EXECUTE can be invoked at most q_{EXECUTE} times, which means there are at most $\binom{q_{\text{EXECUTE}}}{2}$ public key and encapsulation pair. The attacker can win via this condition with probability at most $\binom{q_{\text{EXECUTE}}}{2}(2^{-\alpha_{pk}} + 2^{-\alpha_c})$.
3. $\exists (T_1, i_1), (T_2, i_2)$ such that $(T_1, i_1) \neq (T_2, i_2)$ **and** $\pi_{T_1}^{i_1}.\mathsf{st}_{\mathsf{exe}} = \pi_{T_2}^{i_2}.\mathsf{st}_{\mathsf{exe}} = \mathsf{bindDone}$ **and** $\pi_{T_1}^{i_1}.\mathsf{sid} = \pi_{T_2}^{i_2}.\mathsf{sid}$: Recall that the session identifier of token sessions includes the token's public key and the client's encapsulation. $\pi_{T_1}^{i_1}.\mathsf{sid} = \pi_{T_2}^{i_2}.\mathsf{sid}$ indicates that $\pi_{T_1}^{i_1}$ and $\pi_{T_2}^{i_2}$ agree on the token's public key and the client's encapsulation, which further implies the agreement on the symmetric encryption key of pt. For the correct symmetric key, this further indicates that the $\pi_{T_1}^{i_1}$ and $\pi_{T_2}^{i_2}$ produces the same pt, which violates the assumption that there are no collision between the used pts in **Game** 6. Thus, this condition is always false.
4. for $(C', j') \leftarrow \mathsf{bindPartner}(T, i)$, all of the following conditions must hold: (a) $(C', j', M, t) \notin \mathcal{L}_{\text{AUTH}}$, (b) $\pi_{C'}^{j'} = (\perp, \perp)$ **or** $\pi_{C'}^{j'}.\mathsf{compromised} = \mathsf{false}$, (c) $\pi_T^i.\mathsf{pinCorr} = \mathsf{false}$: According to the condition (2), we know that the attacker $\mathcal{A}$ is not allowed to compromise the binding state of any client (C', j') such that $\pi_{C'}^{j'}.\mathsf{bs} = \pi_T^i.\mathsf{bs}$ According to the condition (3)$\pi_T^i.\mathsf{pinCorr} = \mathsf{false}$ and **Game** 9, we know that the attacker $\mathcal{A}$ cannot execute active attack against token to obtain the token biding state $\pi_T^i.\mathsf{bs}$. Thus, the attacker $\mathcal{A}$ has no idea about the $\pi_T^i.\mathsf{bs}$.

   According to the condition (1) $(C', j', M, t) \notin \mathcal{L}_{\text{AUTH}}$, we know that (M, t) was never output by any of the session π_T^i's partner. The attacker therefore has to forge the message-tag pair (M, t). Recall that the tag t is computed by applying a function H_7 to the corresponding binding state $\pi_T^i.\mathsf{bs}$ and message M. The attacker can only guess the either the tag directly or the toke binding state $\pi_T^i.\mathsf{bs}$. Moreover, recall that:

   (a) The binding state $\pi_T^i.\mathsf{bs}$ is sampled from $\{0,1\}^{\mu'\lambda}$. The attacker $\mathcal{A}$ can guess $\pi_T^i.\mathsf{bs}$ with probability $2^{-\mu'\lambda}$.
   (b) The tag is computed by $t \leftarrow \mathsf{H}_7(\pi_T^i.\mathsf{bs}, M)$ for some message M chosen by the attacker $\mathcal{A}$. Unless the attacker $\mathcal{A}$ can guess the token binding state $\pi_T^i.\mathsf{bs}$ correctly, it is random from the attacker's view, which further implies that t indistinguishable from a random string due to the $\epsilon_{\mathsf{H}_7}^{\mathsf{prf}}$-$\mathsf{prf}$ security of H_7. Thus, the attacker can guess tag t correctly with probability at most 2^{-l_7}.

   Thus, the probability that $\mathcal{A}$ finds can forge tag t for any message M in each VALIDATE query is bounded by

$$2^{-\mu'\lambda} + \epsilon_{\mathsf{H}_7}^{\mathsf{prf}} + 2^{-l_7}$$

Note that the attacker $\mathcal{A}$ can attempts only in VALIDATE oracle, which can be invoked at most q_{VALIDATE} times. By union bound, the advantage that the attacker $\mathcal{A}$ wins via condition 4) is bounded by

$$q_{\text{VALIDATE}}(2^{-\mu'\lambda} + \epsilon^{\mathsf{prf}}_{\mathsf{H}_7} + 2^{-l_7})$$

To sum up, we have that

$$\mathsf{Adv}_9 \leq \binom{q_{\text{EXECUTE}}}{2}(2^{-\alpha_{pk}} + 2^{-\alpha_c}) + q_{\text{VALIDATE}}(2^{-\mu'\lambda} + \epsilon^{\mathsf{prf}}_{\mathsf{H}_7} + 2^{-l_7})$$

Combing all statements above, the proof is concluded by:

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{SUF\text{-}t'}}_{\mathsf{ePACA}}(\mathcal{A}) \leq & (q_{\text{SETUP}} + q_{\text{EXECUTE}})(\epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}} + \epsilon^{\mathsf{swap}}_{\mathsf{H}_5} + \epsilon^{\mathsf{prf}}_{\mathsf{H}_6}) \\
& + \binom{q_{\text{SETUP}} + q_{\text{EXECUTE}}}{2} 2^{1-l_6} + \epsilon^{\mathsf{coll\text{-}res}}_{\mathsf{H}} + q_{\text{NEWU}} 2^{-\alpha_{\mathfrak{D}}} \\
& + \binom{q_{\text{SEND-BIND-T}}}{2} 2^{-\mu'\lambda} + \binom{q_{\text{EXECUTE}}}{2}(2^{-\alpha_{pk}} + 2^{-\alpha_c}) \\
& + q_{\text{SETUP}} \epsilon^{\mathsf{ind\text{-}1cpa}}_{\mathsf{SKE}_3} + q_{\text{EXECUTE}} \epsilon^{\mathsf{ind\text{-}1\$pa\text{-}lpc}}_{\mathsf{SKE}_3} \\
& + q_{\text{SETUP}} \mathsf{pinRetriesMax} 2^{-\alpha_{\mathfrak{D}}} + \binom{q_{\text{EXECUTE}}}{2}(2^{-\alpha_{pk}} + 2^{-\alpha_c}) \\
& + q_{\text{VALIDATE}}(2^{-\mu'\lambda} + \epsilon^{\mathsf{prf}}_{\mathsf{H}_7} + 2^{-l_7})
\end{aligned}$$

□

5.9.6 Proof of Theorem 24

Proof. The proof is given by reduction. If $\mathcal{A}$ can break the ua security of $\Sigma + \Pi$, then there must exist attackers $\mathcal{A}_1$ against auth security of Σ and $\mathcal{A}_2$ against $\mathsf{SUF\text{-}t'}$ security of Π such that either or both can win. Let $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively denote the challengers in auth and $\mathsf{SUF\text{-}t'}$ experiments. The attackers $\mathcal{A}_1$ and $\mathcal{A}_2$ simulate the ua experiment to $\mathcal{A}$ as follows:

1. $\mathcal{A}_1$ and $\mathcal{A}_2$ initialize lists $\mathcal{L}_{\mathsf{frsh}}$, $\mathcal{L}_{\text{AUTH}}$, $\mathcal{L}_{\text{REGISTER}}$, $\mathcal{L}_{\text{CHALLENGE}}$, and $\mathcal{L}_{\text{RESPONSE}}$ to $\emptyset$.
2. When $\mathcal{A}$ queries $\text{REGISTER}((S,i),(T,j,j'),(C,k),\mathsf{tb},UV,d)$, $\mathcal{A}_1$ first sends the query $\text{REGISTER}((S,i),(T,j),\mathsf{tb},UV)$ to $\mathcal{C}_1$ and receives $(m_{\mathsf{rch}}, m_{\mathsf{rcl}}, m_{\mathsf{rcom}}, m_{\mathsf{rrsp}}, d')$. Then, $\mathcal{A}_2$ sends its challenger $\mathcal{C}_2$ the queries $(m_{\mathsf{rcom}}, t) \leftarrow \text{AUTH}(C,k,m_{\mathsf{rcom}})$ and $\mathsf{status} \leftarrow \text{VALIDATE}(T,j',m_{\mathsf{rcom}},t,d)$. Finally, $\mathcal{A}_1$ and $\mathcal{A}_2$ add $(S,i,T,j,j',C,k,m_{\mathsf{rch}},m_{\mathsf{rcl}},m_{\mathsf{rcom}}, t, m_{\mathsf{rrsp}})$ into $\mathcal{L}_{\text{REGISTER}}$ and return $(m_{\mathsf{rch}}, m_{\mathsf{rcl}}, m_{\mathsf{rcom}}, t, m_{\mathsf{rrsp}}, d')$.
3. When $\mathcal{A}$ queries $\text{CHALLENGE}((S,i),(C,k),\mathsf{tb},UV)$, the attacker $\mathcal{A}_1$ first queries $m_{\mathsf{acom}} \xleftarrow{\$} \text{CHALLENGE}((S,i),\mathsf{tb},UV)$ to $\mathcal{C}_1$ followed by executing $(m_{\mathsf{acom}}, t) \leftarrow \mathsf{aCom}(\mathsf{id}_S, m_{\mathsf{ach}}, \mathsf{tb})$. Then, $\mathcal{A}_2$ sends its challenger $\mathcal{C}_2$ the query $(m_{\mathsf{acom}}, t) \leftarrow \text{AUTH}(C,k,m_{\mathsf{acom}})$. Finally, $\mathcal{A}_1$ and $\mathcal{A}_2$ add $(S,i,C,k,m_{\mathsf{ach}},m_{\mathsf{acl}},m_{\mathsf{acom}},t)$ into $\mathcal{L}_{\text{CHALLENGE}}$ and return $(m_{\mathsf{ach}}, m_{\mathsf{acl}}, m_{\mathsf{acom}}, t)$.

4. When $\mathcal{A}$ queries $\textsc{Response}((T, j, j'), m_{\mathsf{acom}}, t, d)$, the attacker $\mathcal{A}_2$ first queries $\mathsf{status} \leftarrow \textsc{Validate}(T, j', m_{\mathsf{acom}}, t, d)$ to its challenger $\mathcal{C}_2$ and directly returns $\perp$ if $\mathsf{status} \neq \mathsf{accepted}$. Then, $\mathcal{A}_1$ queries $m_{\mathsf{arsp}} \overset{\$}{\leftarrow} \textsc{Response}((T, j), m_{\mathsf{acom}})$ to its challenger $\mathcal{C}_1$. Finally, $\mathcal{A}_1$ and $\mathcal{A}_2$ add $(T, j, j', m_{\mathsf{acom}}, t, m_{\mathsf{arsp}})$ into $\mathcal{L}_{\textsc{Response}}$ and return m_{arsp}.
5. When $\mathcal{A}$ queries $\textsc{Complete}((S, i), m_{\mathsf{acl}}, m_{\mathsf{arsp}})$, the attacker $\mathcal{A}_1$ forwards this query to its challenger $\mathcal{C}_1$ and receives a boolean value d. If $d = 1$, then $\mathcal{A}_1$ additionally sets the winning predicate $\mathsf{win\text{-}ua}$ to be $\mathsf{Win\text{-}ua}(S, i)$ and returns d.
6. For all other queries from $\mathcal{A}$, $\mathcal{A}_1$ and $\mathcal{A}_2$ simply forward them to $\mathcal{C}_1$ or $\mathcal{C}_2$ depending whether they are defined in auth or $\mathsf{SUF\text{-}t'}$ experiment and return the results back to $\mathcal{A}$.
7. When $\mathcal{A}$ terminates at some point, $\mathcal{A}_1$ and $\mathcal{A}_2$ both terminate.

Now, we analyze the winning probability of $\mathcal{A}_1$ and $\mathcal{A}_2$ when $\mathcal{A}$ wins. Note that $\mathcal{A}$ can win by violating one of the following cases:

1. The non-$\perp$ session identifiers of ePIA token (resp. server) sessions do not collide with each other, see Line 37 - 40 in Figure 5.14.

   In this case, $\mathcal{A}_1$ also wins by Line 8 - 9 in Figure 5.5.
2. The partnered token and server sessions must have the identical agreed content unless the registration context on the token is corrupted, see Line 42 in Figure 5.14.

   In this case, $\mathcal{A}_1$ also wins by Line 12 in Figure 5.5.
3. The non-$\perp$ session identifiers of ePACA token (resp. client) sessions that completed Bind algorithm do not collide with each other, see Line 44 - 45 in Figure 5.14.

   In this case, $\mathcal{A}_2$ also wins by Line 9 - 10 in Figure 5.12.
4. During the registration interaction, The ePIA token and server sessions must partner with each other and the authorized command message and tag must have been output by one of the non-compromised partners of the ePACA token session without corrupting its setup user, see Line 47 - 50 in Figure 5.14.

   In this case, we separately consider the case whether the condition regarding PIA or PACA sessions is violated. For each $(S', x, T', y, y', C', z, m_{\mathsf{rch}}, m_{\mathsf{rcom}}, t_{\mathsf{rcom}}, m_{\mathsf{rrsp}}) \in \mathcal{L}_{\textsc{Register}}$

   (a) If $\bar{\pi}^x_S.\mathsf{sid} \neq \bar{\pi}^y_{T'}.\mathsf{sid}$, this is impossible since it is orthogonal to the definition of session partner (and session identifiers). Recall that partnering identifies token and server sessions that are successfully communicate with each other and is achieved via the coincidence of the session identifiers, as described in Section 5.4.4.

   (b) Otherwise, $\mathcal{A}_2$ can trivially win by the Line 11 - 14 in Figure 5.12.

5. The token T that was registered with S, must own an ePIA session $\bar{\pi}_T^i$ that is partnered with $\bar{\pi}_S^i$ and produce a response message unless T's registration context of S is corrupted,see Line 52 - 56 in Figure 5.14.

   In this case, we separately consider whether there exists j such that $\bar{\pi}_T^i$ is partnered with $\bar{\pi}_S^i$.

   (a) If such j does not exist, then $\mathcal{A}$ can win only when $(S, T) \in \mathcal{L}_{\mathsf{frsh}}$. This means, $\mathcal{A}_1$ can also win auth experiment by Line 8 - 8 in Figure 5.12.

   (b) Otherwise, such j exists. This means, the attacker $\mathcal{A}_1$ must have queried the oracle $\textsc{Response}(T, j, m_{\mathsf{acom}})$ to its challenger $\mathcal{C}_1$ for some command message m_{acom}. Recall that such query can only be made when $\mathcal{A}$ queries $\textsc{Response}((T, j, j'), m_{\mathsf{acom}}, t, d)$ for some $(j', m_{\mathsf{acom}}, t, d)$. So, the attacker $\mathcal{A}$ wins by the condition $\nexists(j', m_{\mathsf{acom}}, t, d, m_{\mathsf{arsp}})$ such that $(T, j, j', m_{\mathsf{acom}}, t, d, m_{\mathsf{arsp}}) \in \mathcal{L}_{\textsc{Response}}$ with probability 0.

6. The above response message must be produced after an ePACA session $\pi_T^{j'}$ validates some authorized command m_{acom} and tag t with the approval from user, see Line 58 - 58 in Figure 5.14.

   In this case, the attacker $\mathcal{A}_2$ can trivially win $\mathsf{SUF\text{-}t'}$ experiment by Line 8 - 8 in Figure 5.12.

7. The above command m_{acom} and tag t must be authorized by a client ePACA session π_C^k that is partnered with $\pi_T^{j'}$ for some challenge message m_{rch} that was produced by the ePIA session $\bar{\pi}_S^i$, unless π_C^k is compromised or the PIN that sets up token T has been corrupted, see Line 61 - 66 in Figure 5.14.

   We separately consider the cases whether m_{acom} and tag t are authorized by a client ePACA session π_C^k that is partnered with $\pi_T^{j'}$:

   (a) If $(C, k, m_{\mathsf{acom}}, t) \notin \mathcal{L}_{\textsc{Auth}}$ for $(C, k) \leftarrow \mathsf{bindPartner}(T, j')$, then $\mathcal{A}_2$ can trivially win the $\mathsf{SUF\text{-}t'}$ experiment by the Line 11 - 14 in Figure 5.12.

   (b) Otherwise $(C, k, m_{\mathsf{acom}}, t) \in \mathcal{L}_{\textsc{Auth}}$. Note that $\textsc{Auth}$ oracle is only queried by $\mathcal{A}_2$ when $\mathcal{A}$ queries $\textsc{Challenge}$ or $\textsc{Register}$ oracles and that m_{acom} is the command message at authentication phase. This means, $\mathcal{A}$ must have queried $\textsc{Challenge}((S', i'), (C, k), \mathsf{tb}, UV)$ for some $(S', i', \mathsf{tb}, UV)$ that outputs m_{acom} and t. Moreover, recall that $\mathcal{A}$ has queried $\textsc{Response}((T, j, j'), m_{\mathsf{acom}}, t, d)$. By the definition of PIA session identifiers, we have that $\bar{\pi}_T^j.\mathsf{sid} = \bar{\pi}_{S'}^{i'}.\mathsf{sid}$. Furthermore, recall that $\bar{\pi}_T^j.\mathsf{sid} = \bar{\pi}_S^i.\mathsf{sid}$. It then holds that

   $$\bar{\pi}_S^i.\mathsf{sid} = \bar{\pi}_T^j.\mathsf{sid} = \bar{\pi}_{S'}^{i'}.\mathsf{sid}$$

   Recall that we have ensured that the non-$\perp$ session identifiers of PIA token (resp. server) sessions do not collide with each other in the Condition 1. So, it holds that $(S', i') = (S, i)$.

This means, $\mathcal{A}$ has queried $\textsc{Challenge}((S,i),(C,k),\mathsf{tb},UV)$ for some tb and UV. Consequently, there must exist $(S,i,C,k,m_{\mathsf{ach}},m_{\mathsf{acl}},m_{\mathsf{acom}},t) \in \mathcal{L}_{\textsc{Challenge}}$ for some m_{ach} and m_{acl}. The attacker $\mathcal{A}$ wins via this case with probability 0.

To sum up, when every Compl attacker $\mathcal{A}$ wins ua experiment against the composition of the ePIA scheme Σ and the ePACA scheme Π, then the Compl attackers $\mathcal{A}_1$ or $\mathcal{A}_2$ must be able to win in the auth experiment against the underlying ePIA scheme Σ or in the $\mathsf{SUF\text{-}t'}$ experiment against the underlying ePACA scheme Π, which implies the following inequality and concludes the proof.

$$\mathsf{Adv}^{\mathsf{ua}}_{\Sigma+\Pi}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{auth}}_{\Sigma}(\mathcal{A}_1) + \mathsf{Adv}^{\mathsf{SUF\text{-}t'}}_{\Pi}(\mathcal{A}_2)$$

□

Chapter 6

Provable Security of Zoom and Full End-to-End Security

This chapter is based on the paper:

Cas Cremers, Eyal Ronen, and Mang Zhao, "Multi-Stage Group Key Distribution and PAKEs: Securing Zoom Groups against Malicious Servers without New Security Elements", in 2024 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, US, 2024.

This paper was joint work with Cas Cremers and Eyal Ronen. I lead the research on this paper and the substantial contributions in this chapter are my own. My co-authors principally contributed to the initial conception of the work and the final write up of the paper.

6.1 Introduction

Video conferencing apps such as Zoom are globally used by hundreds of millions of users on a daily basis [109], and aim to use cryptographic protocols to achieve some forms of end-to-end encryption. While there have been many recent advancements in highly-secure messaging protocols such as Signal, their core protocols are typically not suitable for real-time group applications, such as video conferencing, which have have fundamentally different requirements involving real-time constraints, robustness, and usability.

In practice, real-time group protocols used in real-world widely deployed applications such as Zoom incorporate design choices based on real-time requirements that include assigning a group leader role to some participants, relying on key distribution instead of key agreement, and using simplified key evolution mechanisms. These design choices improve features like robustness and usability, and enable real-time communication, at the cost of lower security guarantees compared to some state-of-the-art secure messaging protocols.

Nevertheless, many real-time group protocols explicitly claim that they can provide a form of end-to-end security. For example, in Zoom's case, the 'end-to-end security' option in the app's settings is explained as "Encryption key stored on your local device. No one else can obtain your encryption key, not even Zoom." However, it was shown that a malicious Zoom server can eavesdrop or impersonate in groups [113, 114]. The underlying reason is that in practice, a Zoom server acts as the *sole root of trust* for the authenticity of users' public keys and messages, and implicitly to those that are used to distribute group-specific public keys, which in turn are used by the leader to distribute the group key. Thus, if the Zoom server replaces some of these public keys, it can in fact learn your encryption key.

In this work, we propose a transformation to improve the security of a class of protocols against malicious servers, without introducing new security elements or even new message flows. We achieve this by reworking the way in which such protocols use passwords (known as *passcodes* in Zoom). In the Zoom protocol, the server inherently needs to know the group password and uses it to enforce access control for the group. We propose a modification in which the server no longer knows the password, which is distributed only to the group members and is used by them directly for access control. These passwords can be distributed as one would have done currently (e-mail, messaging app, phone, calendar appointment).

In our new threat model, we can no longer rely on the server providing a priori secure channels between group members. Instead, we employ password-authenticated key exchange to prevent offline guessing attacks on the protocol. We then formally prove that the transformed Zoom protocol achieves a strong form of security even in the presence of malicious servers.

At the technical level, we develop syntax and security notions for multi-stage group key distribution protocols, of which Zoom can be seen as an instance. We show under which assumptions the Zoom protocol (version 4.0) can be proven secure in our multi-stage group key distribution model, and how the known attacks fit into this picture. We design a generic transformation that can turn any protocol in our class into a more secure protocol, for which we define a stronger security notion. We show that our transformation preserves the original security property, and provides security against malicious servers. We apply our transformation to the Zoom protocol, resulting in the ZoomPAKE protocol. We also show how the ZoomPAKE protocol prevents the attacks possible on the Zoom protocol. Our main contributions are:

- We develop a solution to improve the security of Zoom-like apps against malicious servers, without introducing new security elements. The core observation is that Zoom already uses group-specific passwords, but they are by design known to the server. By leveraging techniques from password-authenticated key exchange, we can get rid of the reliance on the server for trusted channels.
- To formally prove the security of our solution, we need to develop substantial machinery. We propose a formal model and syntax of multi-stage group key distribution protocols, called mGKD, of which Zoom can be seen as an instance. For such protocols, we develop a basic security notion $\mathsf{Sec\text{-}mGKD\text{-}pki}$, which assumes the server did not interfere with the public keys of a group's participants, and prove that Zoom meets this notion. We show how real-world attacks manifest in this basic notion and notably how malicious zoom servers can manipulate groups.
- We formally prove that our transformation turns a protocol that is $\mathsf{Sec\text{-}mGKD\text{-}pki}$ secure into one that is also secure in a model that makes no assumptions on the server but only on the password, which we call $\mathsf{Sec\text{-}mGKD\text{-}pw}$.
- We show how to efficiently apply our transformation to the Zoom version 4.0 protocol to obtain the ZoomPAKE protocol, in which the server no longer knows the password, and groups are protected against malicious servers.

Outline We discuss related work in Section 6.2 and additional preliminaries in Section 6.3. In Section 6.4 we present our syntax for multi-stage group key distribution (mGKD) protocols and three security notions: basic $\mathsf{Sec\text{-}mGKD\text{-}pki}$ security, full end-to-end $\mathsf{Sec\text{-}mGKD\text{-}pw}$ security, and the combined $\mathsf{Sec\text{-}mGKD\text{-}pw+}$ security. We show in Section 6.5 that the Zoom library can be modeled as a mGKD protocol and provably satisfies the basic $\mathsf{Sec\text{-}mGKD\text{-}pki}$ security, and show how impersonation attacks prevent it from satisfying the stronger $\mathsf{Sec\text{-}mGKD\text{-}pw}$ notion. In Section 6.6, we develop a generic transformation on any $\mathsf{Sec\text{-}mGKD\text{-}pki}$ secure mGKD protocol to achieve $\mathsf{Sec\text{-}mGKD\text{-}pw}$ and $\mathsf{Sec\text{-}mGKD\text{-}pw+}$ security and apply it to Zoom. We compare our work with the the concurrent work in Section 6.7 and offer a technical summary in Section 6.8. We provide full proofs of all theorems in this chapter in Section 6.9.

6.2 Related Work

While there is a lot of adjacent related work that we will mention below, it turns out that there is surprisingly little directly related work in analysis of real-time group protocols. A number of surveys [58, 133, 140, 144, 155, 160] examine numerous historical designs for secure group key establishment in different application scenarios. We identify following three categories:

- centralized group key management/distribution protocols, where each group has a single trusted authority for group key generation and distribution.
- (continuous) group key agreement and distributed key management protocols, where every party in a group contributes to the group key generation and distribution.
- multi-factor key agreement and password-authenticated key exchange protocols, where the group key generation and distribution relies only on a secret that is used for authorization to a group.

We review each of these categories below.

6.2.1 Centralized Group Key Management Protocols

A centralized group key management (CGKM) protocol starts every group with a trusted authority, often referred to as the "Key Distribution Center" (KDC). The KDC is responsible for controlling for the whole group, e.g., member authentication, access control, and group key generation and distribution.

One of the first CGKM schemes is [104, 105]. In this approach, the KDC creates a "Group Key Packet" (GKP) for encrypting the communication payload with the help from the first group participant. The KDC sends the GKP to every party that wants to join the group and encrypts the new GKP to all group participants using the old one. To achieve forward secrecy, the KDC has to recreate the group whenever a participant leaves the group. After that, numerous tree-like CGKM constructions [95, 136, 137, 169, 176, 177] were proposed to reduce computation cost. In these approaches, all trust resides in the KDC, which forms a single point of failure for compromise.

6.2.2 (Continuous) Group Key Agreement

Two important canonical group key agreement protocols are [127, 150]. Their constructions have a binary tree-like hierarchy, where the keying material of each party is a leaf node at the bottom of the tree and the shared group key is the top node of the tree. Every party can compute the key material on the path from its associated leaf node to the top using Diffie-Hellman Exchange (DHE). However, these designs are inherently synchronous: the initialization of the tree requires all parties to be online.

In the asynchronous secure messaging context, where participants might be offline and group keys need to be evolved, this approach does not work without modification. These drawbacks were lifted by the design in [74], leading to a line of papers in the continuous GKA (CGKA) domain, including [13, 22, 128] that focus on continuously evolving (ratcheting) group keys after the group establishment.

In general, ratcheting-like protocols such as CGKAs are impractical for real-time group applications, as they have to tolerate high amounts of packet loss and still being able to continue immediately when some packets arrive on time.

6.2.3 Multi-factor Key Agreement and Password-Authenticated Key Exchange

Multi-factor key agreement protocols often rely on three classes of human authentication factors: (1) something you know, e.g., passwords, (2) something you have, e.g., secure devices, and (3) something you are, e.g., biometric date. Among them, the password is possibly one of the most convenient means for sharing in practice, as it can be easily sent out-of-band, e.g., via email, in letters, or even in-person.

The human-chosen passwords are often low-entropy rather than uniformly at random. The Password-Authenticated Key Exchange (PAKE) protocols are designed to allow some parties to establish a high-entropy session key with authentication based on a low-entropy shared password without being subject to offline guessing attacks. There are numerous modern and efficient 2-party PAKE constructions in the literature, such as CPace [7, 101] and SPAKE2 [2, 3, 9]. The are also several known group PAKE protocols [5, 6, 8, 66]. However, the existing group PAKE protocols always require multiple rounds for the key agreement and is restricted to static groups. Thus, these group PAKE protocols are impractical for the real-time group applications, where the participants can freely join and leave the groups.

6.2.4 Existing Security Analysis for Zoom

In [113, 114], the authors describe several specific classes of impersonation attacks on end-to-end Zoom (version 2.3.1). First, a malicious meeting participant can impersonate any other participant inside this group, since there is no entity authentication in a group meeting. Second, the Zoom server can replay some messages and impersonate a legitimate user for a meeting. Third, if multiple users share a device, the Zoom server colluding with any user can impersonate any other users on the same device. Moreover, the authors also present a tampering attack based on potential implementation flaws and a Denial-of-Service attack. [113, 114] provide feasible countermeasures for each of above specific attacks. However, more general impersonation attacks by a malicious server are not considered in [113, 114].

We notice a recent concurrent work [83] that formally analyzes the security of the end-to-end Zoom protocol assuming the existence of a trusted PKI. This work captures various frameworks underlying the Zoom protocol, including joining and leaving a group, a so-called "heartbeat" liveness mechanism, and change of group leader/host. We give a more detailed comparison between our work and [83] in Section 6.7.

6.3 Additional Preliminaries

6.3.1 $\mathsf{lr\text{-}prf\text{-}ODH}$ Assumption

We recall the generic $\mathsf{lr\text{-}prf\text{-}ODH}$ definition in [64].

Definition 51. *Let* ECDH *denote a cyclic group G with order q over an elliptic curve EC with a generator g. Let* $\mathsf{H} : \mathbb{G} \times \{0,1\}^\star \to \{0,1\}^{l_\mathsf{H}}$ *denote a function. We define a generic security notion* $\mathsf{lr\text{-}prf\text{-}ODH}$ *which is parameterized by* $\mathsf{l}, \mathsf{r} \in \{\mathsf{n}, \mathsf{s}, \mathsf{m}\}$ *indicating how often the attacker is allowed to query a certain "left" resp. "right" oracle (*ODH_u *resp.* ODH_v*) where* n *indicates that no query is allowed,* s *that a single query is allowed, and* m *that multiple (polynomially many) queries are allowed to the respective side. Consider the following security game* $\mathrm{Expr}^{\mathsf{lr\text{-}prf\text{-}ODH}}_{\mathsf{ECDH},\mathsf{H}}$ *between a challenger $\mathcal{C}$ and a* PPT *attacker $\mathcal{A}$.*

1. *The challenger $\mathcal{C}$ samples $u \xleftarrow{\$} \mathbb{Z}_q$ and provides $\mathbb{G}$, g, and g^u to the attacker $\mathcal{A}$.*
2. *If $\mathsf{l} = \mathsf{m}$, $\mathcal{A}$ can issue arbitrarily many queries to the following oracle ODH_u.*

   ODH_u **oracle.** *On a query of the form (S, x), $\mathcal{C}$ first checks if $S \notin \mathbb{G}$ and returns $\perp$ if this is the case. Otherwise, it computes $y \leftarrow \mathsf{H}(S^u, x)$ and returns y.*

3. *Eventually, $\mathcal{A}$ issues a challenge query $x^\star$. On this query, $\mathcal{C}$ samples $v \xleftarrow{\$} \mathbb{Z}_q$ and a bit $\mathsf{b} \xleftarrow{\$} \{0,1\}$ uniformly at random. It then computes $y_0^\star = \mathsf{H}(g^{uv}, x^\star)$ and samples $y_q^\star \xleftarrow{\$} \{0,1\}^{l_\mathsf{H}}$ uniformly at random. The challenger returns $(g^v, y_\mathsf{b}^\star)$ to $\mathcal{A}$.*
4. *Next, $\mathcal{A}$ may issue (arbitrarily interleaved) queries to the following oracles ODH_u and ODH_v (depending on l and r).*

   ODH_u **oracle.** *The attacker $\mathcal{A}$ may ask no ($\mathsf{l} = \mathsf{n}$), a single ($\mathsf{l} = \mathsf{s}$), or arbitrarily many ($\mathsf{l} = \mathsf{m}$) queries to this oracle. On a query of the form (S, x), the challenger first checks if $S \notin G$ or $(S, x) = (g^v, x^\star)$ and returns $\perp$ if this is the case. Otherwise, it computes $y \leftarrow \mathsf{H}(S^u, x)$ and returns y.*

ODH_v **oracle.** *The attacker $\mathcal{A}$ may ask no ($\mathsf{r} = \mathsf{n}$), a single ($\mathsf{r} = \mathsf{s}$), or arbitrarily many ($\mathsf{r} = \mathsf{m}$) queries to this oracle. On a query of the form (T, x), the challenger first checks if $T \notin G$ or $(T, x) = (g^u, x^\star)$ and returns $\perp$ if this is the case. Otherwise, it computes $y \leftarrow \mathsf{H}(T^v, x)$ and returns y.*

5. *At some point, $\mathcal{A}$ stops and outputs a guess $\mathsf{b}' \in \{0, 1\}$.*

We say that the attacker wins the lr-prf-ODH *game if $\mathsf{b}' = \mathsf{b}$. We say the* lr-prf-ODH *problem is ϵ-hard over* ECDH *and* H*, if the advantage of any* PPT *attacker $\mathcal{A}$ that wins above $\mathrm{Expr}^{\mathsf{lr\text{-}prf\text{-}ODH}}_{\mathsf{ECDH},\mathsf{H}}$ experiment is bounded by ϵ.*

In this chapter, we only need the mn-prf-ODH assumption.

6.3.2 Password-Authenticated Key Exchange

The password-authenticated key exchange (PAKE) protocols allow two parties to establish a high-entropy key over an insecure channel using a shared low-entropy password. Below, we first define a weak security (w-PAKE) security model against a PAKE protocol. This model is weaker than and therefore implied by the security model defined in [2, 3, 7]. Thus, some modern and widely used PAKE schemes, including CPace [101] or SPAKE2 [9] that are respectively proven secure in [7] and [2, 3], are provably secure in this w-PAKE model.

Protocol Members. This weak semantic security model only considers the two-party setting. I.e., the PAKE protocol has only two members: either an initiator `init` or a responder `resp`.

The initiator `init` indeed captures the behaviors of (all) participants in each group gid in our mGKD protocol. The responder `resp` indeed captures the behaviors of the (unique) leader in each group gid in our mGKD protocol.

Long-Lived Keys / Passwords. The initiator `init` and the responder `resp` hold the same password pw, which is sampled from a distribution $\mathcal{D}$. In many literature, the password is also called the "long-lived key".

Protocol Execution. The interaction between an attacker $\mathcal{A}$ and the protocol members occurs only via oracle queries, which model the attacker capabilities in a real attack. During the execution, the attacker may create several instances of a member. We consider the concurrent model, i.e., several instances may be active at any given time. Let U^{id} denote the instance with identifier id of a member U. Let $\mathsf{b} \in \{0, 1\}$ be a bit chosen uniformly at random. The attacker $\mathcal{A}$ can query following three oracles:

- $\textsc{SendPAKE}(U^{\mathsf{id}}, m)$: This query models an active attack, in which the attacker may tamper with the message being sent over the public channel. The output of this query is the message that the member instance U^{id} would generate upon receipt of message m.

- CompromisePAKE(U^{id}): This query models the misuse of session keys by a member. If a session key is not defined for instance U^{id} or if a TestPAKE query was asked to either U^{id} or to its partner, then return $\perp$. Otherwise, return the session key held by the instance U^{id}.
- TestPAKE(U^{id}): This query tries to capture the attacker's ability to tell apart a real session key from a random one. If no session key for instance U^{id} is defined or U^{id} is not fresh (which is defined below), then return the undefined symbol $\perp$. Otherwise, return the session key for instance U^{id} if $\mathsf{b} = 1$ or a random key of the same size if $\mathsf{b} = 0$.

Notation. We say an instance U^{id} *opened* if a query CompromisePAKE(U^{id}) has been made by the attacker. We say an instance U^{id} is *unopened* if it is not opened. We say an instance U^{id} *tested* if a query TestPAKE(U^{id}) has been made by the attacker. We say an instance U^{id} is *untested* if it is not tested. We say an instance U^{id} has *accepted* if it goes into an accept mode after receiving the last expected protocol message.

Session Identifiers and Partnering. We define the *session identifiers* (sid) as the transcript of the conversation between the initiator and the responder instances before acceptance.

We say two instances $\mathtt{init}^{\mathsf{id}_1}$ and $\mathtt{resp}^{\mathsf{id}_2}$ to be *partners* if the following conditions are met:

(a) Both $\mathtt{init}^{\mathsf{id}_1}$ and $\mathtt{resp}^{\mathsf{id}_2}$ accept, and

(b) Both $\mathtt{init}^{\mathsf{id}_1}$ and $\mathtt{resp}^{\mathsf{id}_2}$ share the same identifiers sid.

Freshness. The notion of freshness is defined to avoid cases in which attacker can trivially break the security of the scheme. The goal is to only allow the attacker to ask TestPAKE queries to fresh oracle instances. More specifically, we say an instance U^{id} is fresh if it has accepted and if both U^{id} and its partner are unopened and untested.

Semantic Security. Consider an execution of the above experiment for an attacker $\mathcal{A}$ against a PAKE protocol Π. The attacker $\mathcal{A}$ wins the experiment if and only if $\mathcal{A}$ guesses $\mathsf{b}' = \mathsf{b}$, where b is the hidden bit used by the TestPAKE oracle. The advantage of $\mathcal{A}$ breaking the weak security of Π is defined as

$$\mathsf{Adv}^{\mathsf{w\text{-}PAKE}}_{\Pi,\mathcal{D}}(\mathcal{A}) := |\Pr[\mathcal{A} \text{ wins}] - \frac{1}{2}|$$

We say that Π is $\epsilon^{\mathsf{w\text{-}PAKE}}_{\mathsf{PAKE},\mathcal{D}}$-$\mathsf{w\text{-}PAKE}$ secure if for any PPT attackers $\mathcal{A}$ it always holds that

$$\mathsf{Adv}^{\mathsf{w\text{-}PAKE}}_{\Pi,\mathcal{D}}(\mathcal{A}) \leq \epsilon^{\mathsf{w\text{-}PAKE}}_{\Pi,\mathcal{D}}$$

6.4 Multi-stage Group Key Distribution Protocols

In this section, we define our syntax for multi-stage Group Key Distribution mGKD protocols. Our class of mGKD protocols covers behaviors of a party for using real-time group services, such as long-term identity information generation, joining groups, group key rotation, and leaving groups. Then, we propose three security models that capture distinct security guarantees for real-time group services.

6.4.1 mGKD Definition

mGKD protocols are stateful interactive group communication protocols executed by a set of parties $\mathcal{P}$. Each party $P \in \mathcal{P}$ must be uniquely identified by an identifier id_P. Each group is uniquely identified by an identifier gid. In each group, one party performs the leader role; all other group members perform the participant role. The role of each party in different groups might be distinct: a party can be leader in one group and participant in others. In practice, the leader is typically the so-called *host* that initiated the group. In this paper, we assume that each group gid is associated with a unique leader and that the group's leader stays in the group for its entire duration. While one can in theory implement changing leaders by starting a new group, we leave the modeling and efficient implementation of multiple and changing leaders to future work.

Definition 52. *A multi-stage group key distribution protocol* $\mathsf{mGKD} = (\mathsf{SignUp}, \mathsf{Schedule}, \mathsf{Register}, \mathsf{Join}, \mathsf{Leave}, \mathsf{KeyRotat})$ *consists of the following algorithms:*

Sign Up: $m_{\mathsf{SignUp}} \xleftarrow{\$} \mathsf{SignUp}(P)$ *allows a (stateful) party P to initialize a long-term identity information for signing up. The private portion is locally stored. The public portion is output as an outgoing sign-up message m_{SignUp}.*

Group Schedule: $m^{\mathsf{gid}}_{\mathsf{GSch}} \xleftarrow{\$} \mathsf{Schedule}(P, \mathsf{gid}, gs)$ *allows a (stateful) party P to take the role of the leader for scheduling a group* gid *using a group secret gs. The output is an outgoing group schedule message $m^{\mathsf{gid}}_{\mathsf{GSch}}$ for the server. The group secret gs is expected to be sent to authorized participants over secure out-of-band channels.*

Register: $\mathsf{Register} = (\mathsf{Register\text{-}L}, \mathsf{Register\text{-}P})$ *consists of two sub-algorithms depending on the role of the caller:*

- $m' \xleftarrow{\$} \mathsf{Register\text{-}L}(P, \mathsf{gid}, gs, m)$ *(resp. $m' \xleftarrow{\$} \mathsf{Register\text{-}P}(P, \mathsf{gid}, gs, m)$) allows a (stateful) party P to register for a group* gid *as leader (resp. participant) using a group secret gs and an incoming message m followed by group initialization. The output is an outgoing message m'.*

Participant Join: $\mathsf{Join} = (\mathsf{Join\text{-}L}, \mathsf{Join\text{-}P})$ *allows a participant to interact with a leader for joining a group. This interactive phase consists of two sub-algorithms depending on the role of the caller:*

- $m' \xleftarrow{\$} \mathsf{Join\text{-}L}(P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m)$ *(resp.* $m' \xleftarrow{\$} \mathsf{Join\text{-}P}(P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m)$*) allows a leader P (resp. a participant P) of the group* gid *to input a group secret gs and an incoming message m and to output an outgoing message m' for the participant (resp. the leader) P'.*

Member Leave: $\mathsf{Leave} = (\mathsf{Leave\text{-}L}, \mathsf{Leave\text{-}P})$ *consists of two sub-algorithms depending on the role of the caller:*

- $\mathsf{Leave\text{-}L}(P, \mathsf{gid}, \mathsf{id}_{P'})$ *(resp.* $\mathsf{Leave\text{-}P}(P, \mathsf{gid}, \mathsf{id}_{P'})$*) allows the leader (resp. a participant) P of the group* gid *to react to a party P''s leaving. If* $\mathsf{id}_P = \mathsf{id}_{P'}$*, the leader (resp. the participant) P leaves the group* gid *and erases the corresponding state information.*

Key Rotation: $\mathsf{KeyRotat} = (\mathsf{KeyRotat\text{-}L}, \mathsf{KeyRotat\text{-}P})$ *consists of two sub-algorithms depending on the role of the caller:*

- $m_{\mathsf{KRot}} \xleftarrow{\$} \mathsf{KeyRotat\text{-}L}(P, \mathsf{gid}, m)$ *allows the leader P of the group* gid *to input an incoming message m and to locally update the group key. The output is an outgoing message m_{KRot} that enables all participants of the same group* gid *to update group keys correspondingly.*
- $m_{\mathsf{KRot}} \xleftarrow{\$} \mathsf{KeyRotat\text{-}P}(P, \mathsf{gid}, m)$ *allows a participant P of the group* gid *to input an incoming message m and to locally update the group key. The output is an (optionally empty) outgoing message m_{KRot}.*

We assume all incoming and outgoing messages of an mGKD protocol are publicly accessible; we leave this implicit as the concrete mechanisms can differ substantially between protocols, but could for example be a PKI or a "bulletin board" on the server. In contrast, the input group secret gs of the Schedule algorithm is expected to be chosen by the leader and be sent to authorized parties for joining the group over secure out-of-band channels. Before a party P joins a group gid, P has to register for this group, no matter whether P has previously joined the group gid and left. The Member Leave phase enables every party P to react to a participant P' leaving the group, notably, without any additional incoming message. This captures the scenario where the server might notify group members that a participant has left the group without sending any leave request due to unexpected network disconnection. The KeyRotat algorithm enables every party to update their group key, the execution frequency of which can be decided in advance, in a regular schedule and/or when a party joins or leaves the group.

To model concurrent or sequential groups of a party P, let π_P^{gid} denote party P's session with respect to the short-term group gid. In addition, each party P has an associated long-term state st_P that is shared by all of P's sessions.

Definition 53. *In a* mGKD *protocol, each party* P *has the following state variables. The long-term state variables are initialized during the Sign Up phase:*

- $\mathsf{st}_P.\mathsf{id}$*: the associated and unique identifier of the party* P*. In this paper, we assume it equal to* id_P*.*
- $\mathsf{st}_P.isk$*: the identity secret key of the party* P*.*
- $\mathsf{st}_P.ipk$*: the identity public key of the party* P*.*

The short-term per-group state variables are initialized during the Register phase:

- $\pi_P^{\mathsf{gid}}.sk$*: the group-specific secret key of the party* P *for joining the group* gid*.*
- $\pi_P^{\mathsf{gid}}.pk$*: the group-specific public key of the party* P *for joining the group* gid*.*
- $\pi_P^{\mathsf{gid}}.gk$*: the current group key used by party* P*, which is supposed to be shared by all parties in the group* gid*. This variable is initialized with* $\perp$*.*
- $\pi_P^{\mathsf{gid}}.\mathsf{gkid}$*: the index of the current group key of the party* P *in the group* gid*. This variable is initialized with* 0*.*
- $\pi_P^{\mathsf{gid}}.GP$*: The set of all parties in the group* gid*. This variable is initialized with the empty set* $\emptyset$*.*
- $\pi_P^{\mathsf{gid}}.\mathsf{status} \in \{\perp, \mathsf{registered}, \mathsf{joined}\}$*: the status that indicates whether the party* P *has initialized the state for (but not yet registered for), or registered for (but not yet joined), or joined the group* gid*. This variable is* $\perp$ *by default.*

Definition 54 (Correctness)**.** *Consider any group* gid *with an associated leader* P*, any sequence of parties* $\{P^i\}_i$*, and a sequence of executions that includes the following (perhaps not consecutive) algorithms: i) a Sign Up of the leader* P *or a participant* P^i *for any* i*, ii) a Group Schedule of the group* gid *and the leader* P*, iii) a Register of the leader* P *or a participant* P^i *for any* i *to the group* gid*, iv) a successful Participant Join between the leader* P *and a participant* P^i *for any* i*, v) a Leaving of participant* P^i *for any* i *to all other parties in the group* gid*, vi) a successful Key Rotation for the leader* P *and every participant* P^i *in the group* gid*, i.e.,* $\pi_{Pi}^{\mathsf{gid}}.\mathsf{status} = \mathsf{joined}$*. Correctness requires that* $\pi_P^{\mathsf{gid}}.\mathsf{gkid} = \pi_{Pi}^{\mathsf{gid}}.\mathsf{gkid}$ *and* $\pi_P^{\mathsf{gid}}.gk = \pi_{Pi}^{\mathsf{gid}}.gk$ *for any* P^i *with* $\pi_{Pi}^{\mathsf{gid}}.\mathsf{status} = \mathsf{joined}$ *at any time.*

6.4.2 A Generic Security Model

We next define a generic $\mathsf{Sec\text{-}mGKD\text{-}X}$ security model. By presenting different instantiations of the freshness conditions that the attacker must obey and of the winning conditions that the attacker must pursue, we then introduce three distinct concrete models for $\mathsf{X} \in \{\mathsf{pki}, \mathsf{pw}, \mathsf{pw+}\}$ in Section 6.4.3, Section 6.4.4, and Section 6.4.5.

Trust Model. We assume that all parties' sampled randomness is independent, uniform, and unpredictable. For simplicity, we assume every leader samples group secret from a same distribution $\mathcal{D}$ (according to the underlying protocol). We assume that every party joins every group at most once. We assume that every leader stays in the corresponding group for the entire duration and leaves only when all other participants have left. This means, once the leader leaves the corresponding group, this group is immediately marked as "invalid" and no party is allowed to register for or join this group anymore. We assume that every party can send register request for every group at most once. Our model assumes a single shared group key for communication. Thus, impersonation attacks between parties inside the group is out of scope of this work.

Threat Model. We allow the attacker to have full control over the network and can eavesdrop, drop, and insert messages during all phases. We allow the attacker to corrupt the long-term state st_P for any participant P to capture the real-world scenarios where the hardware devices might be stolen. Moreover, we allow attackers to compromise the short-term per-group state π_P^{gid} for any participant P after joining any group gid to capture attacks during the ongoing communications. We also allow attackers to leak arbitrary group keys (to analyze the impact on the security of previous and future group keys). We allow attackers to reveal the group secret for any group to capture the real-world scenarios where the the out-of-band channels might be vulnerable.

Security Experiment. The security experiment is conducted between a challenger $\mathcal{C}$ and an attacker $\mathcal{A}$ against a mGKD protocol Π. At the beginning of the experiment, the challenger $\mathcal{C}$ samples a random challenge bit $\mathsf{b} \in \{0, 1\}$. During the experiment, $\mathcal{C}$ produces two sequences of variables $\{GP^{(\mathsf{gid},\mathsf{gkid})}\}_{\mathsf{gid},\mathsf{gkid}}$ and $\{gk_P^{(\mathsf{gid},\mathsf{gkid})}\}_{\mathsf{gid},\mathsf{gkid},P}$. The variable $GP^{(\mathsf{gid},\mathsf{gkid})}$ aims to record the identifier of every party that is expected to know the gkid-th group key in the group gid from the leader's view. The challenger $\mathcal{C}$ monitors the states of the leader P of any group gid. Whenever $\pi_P^{\mathsf{gid}}.GP$ changes, $\mathcal{C}$ records $GP^{(\mathsf{gid},\pi_P^{\mathsf{gid}}.\mathsf{gkid})} \leftarrow GP^{(\mathsf{gid},\pi_P^{\mathsf{gid}}.\mathsf{gkid})} \cup \pi_P^{\mathsf{gid}}.GP$. The variable $gk_P^{(\mathsf{gid},\mathsf{gkid})}$ records the gkid-th group key derived by any party P in the group gid. Whenever $\pi_P^{\mathsf{gid}}.\mathsf{gkid}$ changes for any party P and group gid during the experiment, $\mathcal{C}$ stores the new group key $gk_P^{(\mathsf{gid},\pi_P^{\mathsf{gid}}.\mathsf{gkid})} \leftarrow \pi_P^{\mathsf{gid}}.gk$. The attacker $\mathcal{A}$ can interact with $\mathcal{C}$ by querying the following oracles, where $\mathcal{C}$ responds according to Π. To simplify the explanation, we partition the oracles into categorizes.

Oracle Category 1: Setup of groups and parties. This category includes a NewParty oracle that simulates the Sign Up phase of a party, a NewGroup that simulates the Group Schedule phase of a group, and a Auth oracle that simulates the group secret transmission from the leader to the authorized participants over out-of-band channels.

- NewParty(id_P): This oracle can be queried at most once on each input. The challenger $\mathcal{C}$ initializes a state st_P by setting $\mathsf{st}_P.\mathsf{id} \leftarrow \mathsf{id}_P$. Then, $\mathcal{C}$ runs $m^P_{\mathsf{SignUp}} \xleftarrow{\$} \mathsf{SignUp}(\mathsf{st}_P)$ and followed by forwarding the sign-up message m^P_{SignUp} to $\mathcal{A}$. The party P is marked as "created".

- NEWGROUP($\mathsf{id}_P, \mathsf{gid}$): This oracle can be invoked at most once for each gid. The input party P must be marked as "created". The challenger $\mathcal{C}$ samples the secret of the group gid by $gs^{\mathsf{gid}} \xleftarrow{\$} \mathcal{D}$ and runs $m_{\mathsf{GSch}}^{\mathsf{gid}} \xleftarrow{\$} \mathsf{Schedule}(P, \mathsf{gid}, gs^{\mathsf{gid}})$ for an associated outgoing message $m_{\mathsf{GSch}}^{\mathsf{gid}}$. Then, $\mathcal{C}$ marks the group gid as "created" and "valid" and marks P as the leader of the group gid and "authorized". Finally, $\mathcal{C}$ returns $m_{\mathsf{GSch}}^{\mathsf{gid}}$ to $\mathcal{A}$.
- AUTH($\mathsf{gid}, \mathsf{id}_P$): The group gid must be marked as both created and valid and the party P must be marked as created and have not registered for the group gid, i.e., $\pi_P^{\mathsf{gid}}.\mathsf{status} = \perp$. The challenger $\mathcal{C}$ marks P as authorized for the group gid.

Oracle Category 2: Register phase. This category includes a REGISTERAUTH oracle, which simulates that an authorized party registers for a group using honest group secret, and a REGISTERINJECT oracle, which simulates that an unauthorized (malicious) party registers for a group with an input using some chosen group secret.

- REGISTERAUTH($\mathsf{id}_P, \mathsf{gid}, m$): This oracle can be queried at most once for each tuple ($\mathsf{id}_P, \mathsf{gid}$). The group gid must be marked as both created and valid and the party P must be authorized for the group gid. The challenger $\mathcal{C}$ runs $\mathsf{Register\text{-}L}(P, \mathsf{gid}, gs^{\mathsf{gid}}, m)$ if P is the leader of the group gid and $\mathsf{Register\text{-}P}(P, \mathsf{gid}, gs^{\mathsf{gid}}, m)$ otherwise. In both cases, $\mathcal{C}$ forwards the output message m' to $\mathcal{A}$.
- REGISTERINJECT($\mathsf{id}_P, \mathsf{gid}, gs, m$): This oracle can be queried at most once for each tuple ($\mathsf{id}_P, \mathsf{gid}$). The group gid must be marked as both created and valid and the party P must be unauthorized for the group gid. The challenger $\mathcal{C}$ runs $\mathsf{Register\text{-}P}(P, \mathsf{gid}, gs, m)$ and forwards the output message m' to $\mathcal{A}$.

Oracle Category 3: Participant Join phase. This category includes a SENDJOINAUTH oracle, which simulates that an authorized party (as either leader or participant) sends messages to another party (either authorized or unauthorized) during the Participant Join phase in the group, and a SENDJOININJECT oracle, which simulates that an unauthorized (malicious) participant sends messages (to the leader) during the Participants Join phase in the group.

- SENDJOINAUTH($\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, m$): The challenger $\mathcal{C}$ first checks
  - whether gid is marked as created and valid,
  - whether P is authorized for this group gid,
  - whether both parties P and P' have been created and registered for this group,
  - whether either P or P' is the leader of the group gid,
  - whether the leader of the group, either P or P', has joined the group, and that the other party hasn't joined the group yet.

If any of the check fails, $\mathcal{C}$ directly returns $\perp$ to $\mathcal{A}$. Otherwise, $\mathcal{C}$ runs $\mathsf{Join\text{-}L}(P, \mathsf{id}_{P'}, \mathsf{gid}, gs^{\mathsf{gid}}, m)$ if P is the leader of the group gid or $\mathsf{Join\text{-}P}(P, \mathsf{id}_{P'}, \mathsf{gid}, gs^{\mathsf{gid}}, m)$ if P is a participant. Then, the output message m' of either $\mathsf{Join\text{-}L}$ or $\mathsf{Join\text{-}P}$ is returned to $\mathcal{A}$.

- $\textsc{SendJoinInject}(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m)$: The challenger $\mathcal{C}$ first checks
  - whether gid is marked as created and valid,
  - whether P is unauthorized for the group gid,
  - whether P' is the leader of the group gid,
  - whether the participant P has been created and registered for this group gid but has not joined the group yet, and
  - whether the leader P' has joined the group gid.

  If any of the check fails, $\mathcal{C}$ directly returns $\perp$ to $\mathcal{A}$. Otherwise, $\mathcal{C}$ runs $\mathsf{Join\text{-}P}(P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m)$ and returns the output message m' of $\mathsf{Join\text{-}P}$ to $\mathcal{A}$.

Oracle Category 4: Member Leave phase. This category includes a $\textsc{SendLeave}$ oracle, which simulates that a party notices another party leaving the group gid, and an $\textsc{EndGroup}$ oracle, which simulates that a leader leaves and ends the group.

- $\textsc{SendLeave}(\mathsf{id}_P, \mathsf{gid}, \mathsf{id}_{P'})$: The challenger $\mathcal{C}$ aborts if
  - the gid is not marked as both created and valid, or
  - the leaving party P' is the leader of the group gid, or
  - the party P has not joined the group.

  Otherwise, $\mathcal{C}$ runs $\mathsf{Leave\text{-}L}(P, \mathsf{gid}, \mathsf{id}_{P'})$ if P is the leader of the group gid and $\mathsf{Leave\text{-}P}(P, \mathsf{gid}, \mathsf{id}_{P'})$ otherwise.

- $\textsc{EndGroup}(\mathsf{gid})$: The challenger $\mathcal{C}$ first checks
  - whether the group gid is created and valid, and
  - whether the leader P of the group gid is the unique party in his local party list, i.e., $\pi_P^{\mathsf{gid}}.GP = \{\mathsf{id}_P\}$.

  If either check fails, $\mathcal{C}$ aborts. Otherwise, $\mathcal{C}$ runs $\mathsf{Leave\text{-}L}(P, \mathsf{gid}, \mathsf{id}_P)$ and marks the group gid as "invalid".

Oracle Category 5: Key Rotation phase. This category includes only one oracle $\textsc{SendKeyRotat}$ that simulates the process where a party updates their local group key.

- $\textsc{SendKeyRotat}(\mathsf{id}_P, \mathsf{gid}, m)$: The party P must have joined the group gid. The challenger $\mathcal{C}$ runs $\mathsf{KeyRotat\text{-}L}(P, \mathsf{gid}, m)$ if P is the leader of the group gid and $\mathsf{KeyRotat\text{-}P}(P, \mathsf{gid}, m)$ otherwise. The output message of either $\mathsf{KeyRotat\text{-}L}$ or $\mathsf{KeyRotat\text{-}P}$ is returned to attacker $\mathcal{A}$.

Oracle Category 6: Secret information leakage. This category includes four oracles $\textsc{Corrupt}$, $\textsc{Compromise}$, $\textsc{Leak}$, and $\textsc{Reveal}$, that respectively simulates that the attacker $\mathcal{A}$ knows the long-term state of a party, the short-term per-group of a party for a group, a group key of a party in a group, and the group secret of a group.

- $\textsc{Corrupt}(\mathsf{id}_P)$: The challenger $\mathcal{C}$ first checks whether the party P is created. If the check fails, $\mathcal{C}$ simply returns $\perp$. Otherwise, $\mathcal{C}$ returns st_P to the attacker $\mathcal{A}$ and marks st_P as "corrupted".
- $\textsc{Compromise}(\mathsf{id}_P, \mathsf{gid})$: The challenger $\mathcal{C}$ checks whether the party P has joined the group gid, i.e, $\pi_P^{\mathsf{gid}}.\mathsf{status} = \mathsf{joined}$. If the check fails, $\mathcal{C}$ simply returns $\perp$. Otherwise, $\mathcal{C}$ returns π_P^{gid} to the attacker $\mathcal{A}$, followed by marking π_P^{gid} as "compromised" and $gk_P^{(\mathsf{gid}, \pi_P^{\mathsf{gid}}.\mathsf{gkid})}$ as "leaked".
- $\textsc{Leak}(\mathsf{id}_P, \mathsf{gid}, \mathsf{gkid})$: The challenger $\mathcal{C}$ checks whether $gk_P^{(\mathsf{gid},\mathsf{gkid})}$ has been set. If $gk_P^{(\mathsf{gid},\mathsf{gkid})} = \perp$, then $\mathcal{C}$ simply returns $\perp$. Otherwise, $\mathcal{C}$ marks $gk_P^{(\mathsf{gid},\mathsf{gkid})}$ as "leaked" and returns $gk_P^{(\mathsf{gid},\mathsf{gkid})}$ to $\mathcal{A}$.
- $\textsc{Reveal}(\mathsf{gid})$: If the group gid is not created, then the challenger $\mathcal{C}$ simply returns $\perp$. Otherwise, $\mathcal{C}$ marks gid as "revealed" and returns gs^{gid} to $\mathcal{A}$.

Oracle Category 7: Test challenge bit. This category includes only one $\textsc{Test}$ oracle that returns either a real group key if the challenge bit $\mathsf{b} = 0$, or a random key if $\mathsf{b} = 1$.

- $\textsc{Test}(\mathsf{id}_P, \mathsf{gid}, \mathsf{gkid})$: This oracle can be queried at most once. If the party P is authorized for the group gid and the party P has produced gkid-th group key, i.e., $gk_P^{(\mathsf{gid},\mathsf{gkid})} \neq \perp$, then the challenger $\mathcal{C}$ returns $gk_P^{(\mathsf{gid},\mathsf{gkid})}$ to $\mathcal{A}$ if the challenge bit $\mathsf{b} = 0$, or a random key from the same space if $\mathsf{b} = 1$. Then, $\mathcal{C}$ further marks the party P, the group identifier gid, and the group key index gkid as "tested". Otherwise, $\mathcal{C}$ immediately returns $\perp$.

Advantage Measures. In the end, the attacker $\mathcal{A}$ outputs a bit $\mathsf{b}' \in \{0, 1\}$. Under two freshness conditions $\mathsf{frsh}_{\mathsf{KAuth}}^{\mathsf{Sec\text{-}mGKD\text{-}X}}$ and $\mathsf{frsh}_{\mathsf{KPriv}}^{\mathsf{Sec\text{-}mGKD\text{-}X}}$ that prevent the attacker $\mathcal{A}$ from trivially winning the experiment for $\mathsf{X} \in \{\mathsf{pki}, \mathsf{pw}, \mathsf{pw+}\}$, we say the attacker $\mathcal{A}$ wins the experiment $\mathsf{Sec\text{-}mGKD\text{-}X}$ against a mGKD protocol Π for $\mathsf{X} \in \{\mathsf{pki}, \mathsf{pw}, \mathsf{pw+}\}$, if either of the following events is triggered:

1. [Event E_{KAuth}] there exists any group gid with the leader P^{gid}, any party P' that is authorized for the group gid, and any group key index gkid, such that $gk_{P'}^{(\mathsf{gid},\mathsf{gkid})} \neq \perp$ but $gk_{P^{\mathsf{gid}}}^{(\mathsf{gid},\mathsf{gkid})} \neq gk_{P'}^{(\mathsf{gid},\mathsf{gkid})}$, without violating the freshness condition $\mathsf{frsh}_{\mathsf{KAuth}}^{\mathsf{Sec\text{-}mGKD\text{-}X}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$.
2. [Event E_{KPriv}] $\mathsf{b} = \mathsf{b}'$ without violating the freshness condition $\mathsf{frsh}_{\mathsf{KPriv}}^{\mathsf{Sec\text{-}mGKD\text{-}X}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$, where P', gid, and gkid are respectively the tested party, group identifier, and group key index.

We define $\mathsf{Adv}_{\Pi}^{\mathsf{Sec\text{-}mGKD\text{-}X}}(\mathcal{A})$ as the advantage that $\mathcal{A}$ can win the Sec-mGKD-X experiment against a mGKD protocol Π for $\mathsf{X} \in \{\mathsf{pki}, \mathsf{pw}, \mathsf{pw+}\}$, namely,

$$\mathsf{Adv}_{\Pi}^{\mathsf{Sec\text{-}mGKD\text{-}X}}(\mathcal{A}) := \max\left(\left|\Pr[E_{\mathsf{KPriv}}] - \frac{1}{2}\right|, \Pr[E_{\mathsf{KAuth}}]\right) .$$

Definition 55 (Sec-mGKD-X). *We say that a* mGKD *protocol* Π *is* Sec-mGKD-X-*secure for* $\mathsf{X} \in \{\mathsf{pki}, \mathsf{pw}, \mathsf{pw+}\}$, *if the above defined advantage* $\mathsf{Adv}_{\Pi}^{\mathsf{Sec\text{-}mGKD\text{-}X}}(\mathcal{A})$ *is negligible for any* PPT *attacker* $\mathcal{A}$.

6.4.3 The Sec-mGKD-pki Security Model

Our basic security model Sec-mGKD-pki captures the following security guarantees for an authorized party in a group assuming the honest distribution of long-term sign-up message of all parties within this group. Note that this basic model guarantees that only group members can learn the key, and where group membership is determined by the *server*. In reality, and in our model, the server may insert group members that are not authorized by the leader and do not know the passcode.

1. **(Implicit) Group Key Authentication**: If a group member accepts a group key, then the leader must have produced the same group with the same group key index.
2. **Group Key Secrecy**: If a group member accepts a group key, then an attacker cannot derive this key, even if it knows other group keys.
3. **Perfect Forward Secrecy**: An attacker that compromises a party's long-term keys, can not learn the group keys of any group the party was previously in.

Definition 56 (Sec-mGKD-pki Freshness Conditions). *We say the freshness condition* $\mathsf{frsh}_{\mathsf{KAuth}}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$, *where* gid *has a unique leader* P^{gid}, *holds if and only if*

1. the per-group states $\pi_{P^{\mathsf{gid}}}^{\mathsf{gid}}$ *and* $\pi_{P'}^{\mathsf{gid}}$ *are not compromised,*

2. the long-term states $\mathsf{st}_{P^{\mathsf{gid}}}$ *and* $\mathsf{st}_{P'}$ *are not corrupted before* P^{gid} *and* P' *joined the group* gid, *and*

3. the sign-up messages $m_{\mathsf{SignUp}}^{P^{\mathsf{gid}}}$ *and* $m_{\mathsf{SignUp}}^{P'}$ *of* P^{gid} *and* P' *are honestly delivered to the other.*

We say the freshness condition $\mathsf{frsh}_{\mathsf{KPriv}}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$ *holds if and only if*

1. *the group key* $gk_P^{(\mathsf{gid},\mathsf{gkid})}$ *is not leaked for all parties* P *in the group* gid *with* $\mathsf{id}_P \in GP^{(\mathsf{gid},\mathsf{gkid})}$*,*
2. *the short-term state* π_P^{gid} *is not compromised for all parties* P *in the group* gid *with* $\mathsf{id}_P \in GP^{(\mathsf{gid},\mathsf{gkid})}$*,*
3. *the long-term state* $\mathsf{st}_{P^{\mathsf{gid}}}$ *of the leader* P^{gid} *in the group* gid *is not corrupted before all parties* P *with* $\mathsf{id}_P \in GP^{(\mathsf{gid},\mathsf{gkid})}$ *joined the group* gid*,*
4. *the long-term state* st_P *of all participants* P *in the group* gid *with* $\mathsf{id}_P \in GP^{(\mathsf{gid},\mathsf{gkid})}$ *is not corrupted before* P *joined the group* gid*, and*
5. *the sign-up messages* m_{SignUp}^P *of all parties* P *with* $\mathsf{id}_P \in GP^{(\mathsf{gid},\mathsf{gkid})}$ *are honestly distributed within the group* gid*.*

Conclusion. Our Sec-mGKD-pki security model captures all guarantees listed at the beginning of this subsection:

1. **(Implicit) Group Key Authentication**: If authentication does not hold, the attacker $\mathcal{A}$ can win via E_{KAuth}.
2. **Group Key Secrecy**: The attacker is allowed to leak arbitrarily many group keys except for the tested one. If group key secrecy does not hold, $\mathcal{A}$ can win via E_{KPriv}.
3. **Perfect Forward Secrecy**: The attacker is allowed to corrupt the long-term state of the tested party. If perfect forward secrecy does not hold, $\mathcal{A}$ can win via E_{KPriv}.

6.4.4 The Sec-mGKD-pw Security Model

The basic Sec-mGKD-pki model has two restrictions:

- First, both freshness conditions $\mathsf{frsh}_{\mathsf{KPriv}}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}$ and $\mathsf{frsh}_{\mathsf{KAuth}}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}$ require the honest distribution of the sign-up messages. Since the sign-up messages are distributed by servers or by PKI in practice, this restriction is also known as "trusted PKI" assumption in the related literature. In the full end-to-end setting, i.e., no trusted PKI or server exists, a Sec-mGKD-pki secure mGKD protocol might still suffers from machine-in-the-middle attacks such that the attacker can easily impersonate any participant towards the group leader and impersonate any group leader towards any participant.
- Second, both freshness conditions $\mathsf{frsh}_{\mathsf{KPriv}}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}$ and $\mathsf{frsh}_{\mathsf{KAuth}}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}$ allow the attacker to reveal all group secrets but do not prevent unauthorized parties from knowing the group keys. Thus, this Sec-mGKD-pki model does not capture the security benefit provided by the group secret transmitted over secure out-of-band channels.

The goal of the $\mathsf{Sec\text{-}mGKD\text{-}pw}$ security model is to preserve the guarantees achieved in $\mathsf{Sec\text{-}mGKD\text{-}pki}$ model and to capture the following additional security guarantee, while getting rid of the "trusted PKI" assumption.

4. **(Implicit) Group Member Authentication**: If any party produces a same group key as the leader of a group, then this party must be authorized for this group, as long as the group secret is not revealed.

We replace the "trusted PKI" assumption with the arguably simpler assumption of a shared "secure (short) group secret". We assume that (short) group secrets (such as passwords or pin codes) can be distributed over out-of-band secure channels, e.g., by email, encrypted messaging application (e.g., Signal), or even in person. In fact, a similar passcode mechanism has been widely deployed in real life by many service providers such as Zoom for access control management (see Section 6.5.1). The only difference is that the current Zoom passcode mechanism hands over passcode and the rule of verifying it to the (possibly) untrusted server, while in our model the group secret is known only to the participants.

Definition 57 (Sec-mGKD-pw Freshness Conditions)**.** *We say the freshness condition* $\mathsf{frsh}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\mathsf{KAuth}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$*, where* gid *has a unique leader* P^{gid}*, holds if and only if*

1. *the per-group states* $\pi^{\mathsf{gid}}_{P^{\mathsf{gid}}}$ *and* $\pi^{\mathsf{gid}}_{P'}$ *are not compromised,*
2. *the long-term states* $\mathsf{st}_{P^{\mathsf{gid}}}$ *and* $\mathsf{st}_{P'}$ *are not corrupted before* P^{gid} *and* P' *joined the group* gid*, and*
3. *the group* gid *is not revealed.*

We say the freshness condition $\mathsf{frsh}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\mathsf{KPriv}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$ *holds if and only if*

1. *the group key* $gk^{(\mathsf{gid},\mathsf{gkid})}_{P}$ *is not leaked for all authorized parties* P *in the group* gid *with* $\mathsf{id}_P \in GP^{(\mathsf{gid},\mathsf{gkid})}$*,*
2. *the short-term state* π^{gid}_{P} *is not compromised for all authorized parties* P *in the group* gid *with* $\mathsf{id}_P \in GP^{(\mathsf{gid},\mathsf{gkid})}$*,*
3. *the long-term state* $\mathsf{st}_{P^{\mathsf{gid}}}$ *of the leader* P^{gid} *in the group* gid *is not corrupted before all authorized parties* P *with* $\mathsf{id}_P \in GP^{(\mathsf{gid},\mathsf{gkid})}$ *joined the group* gid*,*
4. *the long-term state* st_P *of all authorized participants* P *in the group* gid *with* $\mathsf{id}_P \in GP^{(\mathsf{gid},\mathsf{gkid})}$ *is not corrupted before* P *joined the group* gid*, and*
5. *the group* gid *is not revealed.*

Conclusion. Note that $\mathsf{Sec\text{-}mGKD\text{-}pki}$ and $\mathsf{Sec\text{-}mGKD\text{-}pw}$ models share the same oracles and similar freshness conditions. Thus, it is straightforward that our $\mathsf{Sec\text{-}mGKD\text{-}pw}$ model also satisfies all guarantees listed in Section 6.4.3. However, we stress two main distinctions of the freshness conditions in Sec-mGKD-pw and Sec-mGKD-pki models that make the security guarantees provided by these two models very different.

1. [**Full End-to-End Security**] Both $\mathsf{frsh}_{\mathsf{KAuth}}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}$ and $\mathsf{frsh}_{\mathsf{KPriv}}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}$ allow attackers to manipulate the transmission of all messages, including the sign-up messages of all parties in any group, which are forbidden by $\mathsf{frsh}_{\mathsf{KAuth}}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}$ sub-point 3) and $\mathsf{frsh}_{\mathsf{KPriv}}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}$ sub-point 5). Instead, $\mathsf{frsh}_{\mathsf{KAuth}}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}$ sub-point 3) and $\mathsf{frsh}_{\mathsf{KPriv}}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}$ sub-point 5) require no leakage of the group secrets. This indicates that our Sec-mGKD-pw security model captures the full end-to-end security, i.e., against a malicious server. Consequently, this new Sec-mGKD-pw model solves the first restriction of Sec-mGKD-pki model.
2. While the $\mathsf{frsh}_{\mathsf{KPriv}}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$ condition sub-points 1) - 4) require no group key leakage, no short-term per-group state compromise, and no long-term state corruption for all parties P in the group with $\mathsf{id}_P \in GP^{(\mathsf{gid},\mathsf{gkid})}$, our new $\mathsf{frsh}_{\mathsf{KPriv}}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$ condition has the same requirements in sub-points 1) - 4) but only for the authorized parties in every group. By this, our Sec-mGKD-pw model captures the following property:
    - **(Implicit) Group Member Authentication**: The attacker can leak arbitrarily many group keys of any unauthorized party in the tested group. If an unauthorized party can successfully produce a same group key as a leader, then the attacker can test this leader, leak the group key of this unauthorized party, and win via the event E_{KPriv}.

6.4.5 The Sec-mGKD-pw+ Security Model

Note that the Sec-mGKD-pki and Sec-mGKD-pw models rely on different assumptions: trusted PKI and secure group secret. We then define a third Sec-mGKD-pw+ model that incorporates the above two models. The goal of the Sec-mGKD-pw+ model is to capture the security of a mGKD protocol if either the "trusted PKI" or the "secure group secret" assumption holds.

Definition 58 (Sec-mGKD-pw+ Freshness Conditions)**.** *We say the freshness condition* $\mathsf{frsh}_{\mathsf{KAuth}}^{\mathsf{Sec\text{-}mGKD\text{-}pw+}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$ *holds if and only if* $\mathsf{frsh}_{\mathsf{KAuth}}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$ *or* $\mathsf{frsh}_{\mathsf{KAuth}}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$ *holds.*

We say the freshness condition $\mathsf{frsh}_{\mathsf{KPriv}}^{\mathsf{Sec\text{-}mGKD\text{-}pw+}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$ *holds if and only if* $\mathsf{frsh}_{\mathsf{KPriv}}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$ *or* $\mathsf{frsh}_{\mathsf{KPriv}}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$ *holds.*

The following corollary is straightforward by definition:

Corollary 1. *Let* Π *denote a* mGKD *protocol. If* Π *is* Sec-mGKD-pki *and* Sec-mGKD-pw *secure, then* Π *is also* Sec-mGKD-pw+ *secure, and vice versa.*

6.5 Zoom's Protocol is a mGKD Protocol

The Zoom library allows parties to establish end-to-end group meeting communication. In this section, we first introduce the Zoom overview [53] (version 4.0) in Section 6.5.1. Then, we show that Zoom library is Sec-mGKD-pki secure in Section 6.5.2 but not Sec-mGKD-pw secure in Section 6.5.3.

6.5.1 The Zoom End-to-End Connection Overview

For end-to-end encrypted meetings, Zoom only supports connecting from installed clients: Browser-based connections are not supported. Zoom distinguishes between users and devices by non-cryptographic user identifiers uid and hardware identifiers hid. We model the identifier of each party P by a pair of user and hardware identifiers, i.e., $\mathsf{id}_P = (\mathsf{uid}_P, \mathsf{hid}_P)$ and assume that party identifiers are unique[1].

Zoom deploys two infrastructures for transmitting cryptographic primitives: an identity management system and a multimedia router (MMR). While the identity management system distributes cryptographic public keys generated by individual clients, the MMR distributes cryptographic messages between parties in a meeting. The connection between parties and servers are established on TLS-tunnels over TCP and are encrypted with AES in GCM mode. In this paper, we assume the existence of Zoom servers but do not explicitly model them, because our goal is to consider them attacker-controlled. Zoom allows every party to set up a group meeting. Groups are uniquely identified by their group identifiers gid. Each group meeting is equipped with a specific "bulletin board", where all parties can post (their own) and retrieve (others') cryptographic messages. The server is able to control and tamper with the bulletin boards.

Below, we first recall the cryptographic algorithms ZSign and ZBox underlying the Zoom library in Section 6.5.1.1. Then, we introduce the Zoom end-to-end protocol in Section 6.5.1.2.

6.5.1.1 Cryptographic Algorithms

The Zoom library makes use of the interface and implementation of two building blocks in the NaCl [88]-inspired libsodium library [79]: *Signing* and *Authenticated Public-Key Encryption* (aka. Box).

Zoom Signing Algorithm: The construction of the Zoom Signing algorithm $\mathsf{ZSign} = (\mathsf{ZSign.KGen}, \mathsf{ZSign.Sign}, \mathsf{ZSign.Vrfy})$ is depicted in Figure 6.1.

- The key generation algorithm $\mathsf{ZSign.KGen}$ simply generates and outputs a DS key pair.

[1]The Zoom white-paper [53] states that the user identifiers uid are assigned by servers and the hardware identifiers hid are randomly sampled. Based on this, we assume that they are unique in practice.

```
ZSign.KGen:
    (pk_ZSign, sk_ZSign) <-$ DS.KGen()
    return (pk_ZSign, sk_ZSign)
```

```
ZSign.Sign(sk_ZSign, ctxt, m):
    $m' \leftarrow \mathsf{H}_1(\mathsf{ctxt}) \| \mathsf{H}_1(m)$
    $\sigma \leftarrow \mathsf{DS.Sign}(sk_{\mathsf{ZSign}}, m')$
    return $\sigma$
```

```
ZSign.Vrfy(pk_ZSign, σ, ctxt, m):
    $m' \leftarrow \mathsf{H}_1(\mathsf{ctxt}) \| \mathsf{H}_1(m)$
    return $\mathsf{DS.Vrfy}(pk_{\mathsf{ZSign}}, m', \sigma)$
```

Figure 6.1: The Zoom-Signing algorithm ZSign. Zoom instantiates H_1 with SHA256 and DS with EdDSA over Ed25519.

```
ZBox.KGen():
    $(pk_{\mathsf{ZBox}}, sk_{\mathsf{ZBox}}) \xleftarrow{\$} \mathsf{ECDH}$
    return $(pk_{\mathsf{ZBox}}, sk_{\mathsf{ZBox}})$
ZBox.Enc($sk^{\mathsf{S}}_{\mathsf{ZBox}}, pk^{\mathsf{R}}_{\mathsf{ZBox}}, \mathsf{ctxt}_{\mathsf{H}_2}, \mathsf{ctxt}_{\mathsf{Cipher}}, \mathsf{Meta}, m$):
    $\mathsf{nonce} \xleftarrow{\$} \{0,1\}^l$
    $K' \leftarrow sk^{\mathsf{S}}_{\mathsf{ZBox}} \cdot pk^{\mathsf{R}}_{\mathsf{ZBox}}$
    $K \leftarrow \mathsf{H}_2(K', \mathsf{ctxt}_{\mathsf{H}_2})$
    $h \leftarrow \mathsf{H}_1(\mathsf{ctxt}_{\mathsf{Cipher}}) \| \mathsf{H}_1(\mathsf{Meta})$
    $c' \leftarrow \mathsf{AEAD.Enc}(K, \mathsf{nonce}, h, m)$
    $c \leftarrow (c', \mathsf{nonce})$
    return $c$
```

```
ZBox.Dec($sk^{\mathsf{R}}_{\mathsf{ZBox}}, pk^{\mathsf{S}}_{\mathsf{ZBox}}, \mathsf{ctxt}_{\mathsf{H}_2}, \mathsf{ctxt}_{\mathsf{Cipher}}, \mathsf{Meta}, c$):
    Parse $(c', \mathsf{nonce}) \leftarrow c$
    $K' \leftarrow sk^{\mathsf{R}}_{\mathsf{ZBox}} \cdot pk^{\mathsf{S}}_{\mathsf{ZBox}}$
    $K \leftarrow \mathsf{H}_2(K', \mathsf{ctxt}_{\mathsf{H}_2})$
    $h \leftarrow \mathsf{H}_1(\mathsf{ctxt}_{\mathsf{Cipher}}) \| \mathsf{H}_1(\mathsf{Meta})$
    $m \leftarrow \mathsf{AEAD.Dec}(K, \mathsf{nonce}, h, c')$
    require $m \neq \perp$
    return $m$
```

Figure 6.2: The Zoom-Box algorithm ZBox. We have that $l = 192$, and the underlying function H_1 denotes SHA256. The function H_2 denotes HKDF (using an empty salt parameter). ECDH is performed on Curve25519. "$\cdot$" denotes scalar multiplication. The AEAD is instantiated with xchacha20poly1305.

- The signing algorithm ZSign.Sign inputs a secret key sk_{ZSign}, a context ctxt, and a message m. The ZSign.Sign first computes the hash function H_1 over respective context ctxt and message m, followed by concatenating them. Then, the ZSign.Sign computes and outputs the signature of the concatenation using DS upon the input secret key sk_{ZSign}.
- The verification ZSign.Vrfy algorithm inputs a public key pk_{ZSign}, a signature σ, a context ctxt, and a message m. This ZSign.Vrfy algorithm simply computes the concatenation as in the signing algorithm and outputs the DS verification result DS.Vrfy upon the public key pk_{ZSign}, the signature σ, and the concatenation.

Zoom Authenticated Public-Key Encryption (aka. Box) Algorithm: The Zoom Box algorithm ZBox = (ZBox.KGen, ZBox.Enc, ZBox.Dec) is depicted in Figure 6.2.

- The key generation algorithm ZBox.KGen samples and outputs a Diffie-Hellman key pair over an elliptic curve ECDH.
- The encryption algorithm ZBox.Enc takes as inputs a sender's secret key $sk^{\mathsf{S}}_{\mathsf{ZBox}}$, a receiver's public key $pk^{\mathsf{R}}_{\mathsf{ZBox}}$, two contexts $\mathsf{ctxt}_{\mathsf{H}_2}$ and $\mathsf{ctxt}_{\mathsf{Cipher}}$, a meta data Meta, and a message m. It first samples a random nonce of bit length l. Next, it computes the Diffie-Hellman exchange of $sk^{\mathsf{S}}_{\mathsf{ZBox}}$ and $pk^{\mathsf{R}}_{\mathsf{ZBox}}$, which is combined with the context $\mathsf{ctxt}_{\mathsf{H}_2}$

and used as input to the hash function H_2 for a key K. Then, it computes the header h by concatenating $\mathsf{H}_1(\mathsf{ctxt}_{\mathsf{Cipher}})$ and $\mathsf{H}_1(\mathsf{Meta})$. The output is the nonce nonce and an AEAD ciphertext produced from the key K, nonce nonce, data h, and message m.

- The decryption algorithm $\mathsf{ZBox.Dec}$ takes as input a receiver's secret key $sk^{\mathsf{R}}_{\mathsf{ZBox}}$, a sender's public key $pk^{\mathsf{S}}_{\mathsf{ZBox}}$, two contexts $\mathsf{ctxt}_{\mathsf{H}_2}$ and $\mathsf{ctxt}_{\mathsf{Cipher}}$, a meta data Meta, and a ciphertext c. This algorithm parses the nonce nonce from the ciphertext c, computes the key K and the header h as in the encryption algorithm, and executes the AEAD decryption. If the AEAD outputs a message $m \neq \perp$, this algorithm simply outputs this m, and aborts otherwise.

6.5.1.2 The Zoom End-to-End Protocol

The end-to-end secure Zoom library consists of six phases following the mGKD protocol syntax[2], of which we give an overview in Figure 6.3. For domain separation, Zoom uses hardcoded context strings (ctxt_1,ctxt_2,ctxt_3).

Sign Up $\mathsf{SignUp}(P)$***:*** During the Sign Up phase, every party P samples an identity public-private ZSign key pair and stores them into the long-term state st_P. The party P outputs the identity public key to the server as the sign-up message. This algorithm is executed only once for each party, i.e., each user on each hardware device.[3]

Group Schedule $\mathsf{Schedule}(P, \mathsf{gid}, gs)$***:*** During the Group Schedule phase, the leader P parses a passcode pc^{gid} from the input gs. The leader P sends pc^{gid} to not only the server as the group schedule message $m^{\mathsf{gid}}_{\mathsf{GSch}}$ for the access control management, but also to the authorized participants for joining the group over out-of-band channels, e.g., email.

Register $\mathsf{Register} = (\mathsf{Register\text{-}L}, \mathsf{Register\text{-}P})$***:*** The Register phase enables every party P to register for joining the meeting gid. We separate the description for $\mathsf{Register\text{-}P}(P', \mathsf{gid}, gs, m)$, where the P' is a participant of the group gid, and for $\mathsf{Register\text{-}L}(P, \mathsf{gid}, gs, m)$, where P the leader of the group gid.

- $\mathsf{Register\text{-}P}(P', \mathsf{gid}, gs, m)$: The input message m is given by the server and should be correctly parsed as a special mUUID string. The mUUID string is a server-generated per-group-instance random string that the individual parties cannot control. Moreover, the participant P' also inputs a group secret gs that can be correctly parsed as a passcode pc^{gid}. This algorithm first samples a public-private per-group ZBox key pair and stores them into the state $\pi^{\mathsf{gid}}_{P'}$. Next, it computes $\mathsf{Binding}^{\mathsf{gid}}_{P'}$, which is the concatenation of the group identifier gid, server-generated random string mUUID, as well as the party P''s

[2]The official Zoom white-paper [53] only sketches the re-joining mechanism informally, and does not specify any mechanism to change leaders. We leave the analysis both functionalities to future work.

[3]The Zoom library also supports anonymous log-in: people without a Zoom account can also join a group meeting as a "guest participant" (note that the guest cannot play the role of leader). Before a guest joins a group, the Sign Up algorithm is always executed. This prevents other parties from tracing them across meetings by noticing when a long-term key is reused [53].

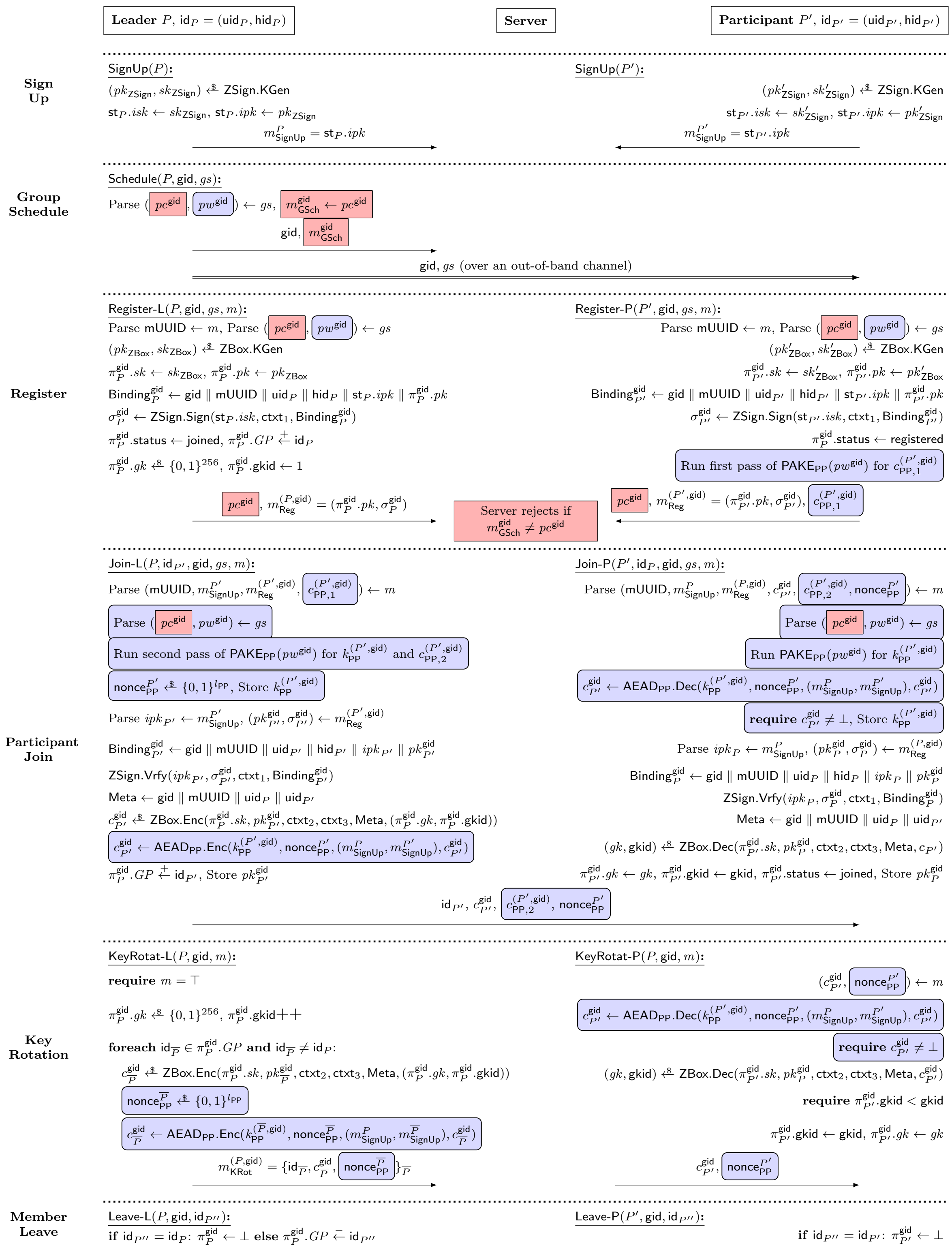

Leader P, $\mathsf{id}_P = (\mathsf{uid}_P, \mathsf{hid}_P)$
Server
Participant P', $\mathsf{id}_{P'} = (\mathsf{uid}_{P'}, \mathsf{hid}_{P'})$
Sign Up
SignUp(P):
$(pk_{\mathsf{ZSign}}, sk_{\mathsf{ZSign}}) \xleftarrow{\$} \mathsf{ZSign.KGen}$
$\mathsf{st}_P.isk \leftarrow sk_{\mathsf{ZSign}}$, $\mathsf{st}_P.ipk \leftarrow pk_{\mathsf{ZSign}}$
$m^P_{\mathsf{SignUp}} = \mathsf{st}_P.ipk$
SignUp(P'):
$(pk'_{\mathsf{ZSign}}, sk'_{\mathsf{ZSign}}) \xleftarrow{\$} \mathsf{ZSign.KGen}$
$\mathsf{st}_{P'}.isk \leftarrow sk'_{\mathsf{ZSign}}$, $\mathsf{st}_{P'}.ipk \leftarrow pk'_{\mathsf{ZSign}}$
$m^{P'}_{\mathsf{SignUp}} = \mathsf{st}_{P'}.ipk$
Group Schedule
Schedule(P, gid, gs):
Parse (pc^{gid}, pw^{gid}) $\leftarrow gs$, $m^{\mathsf{gid}}_{\mathsf{GSch}} \leftarrow pc^{\mathsf{gid}}$
gid, $m^{\mathsf{gid}}_{\mathsf{GSch}}$
gid, gs (over an out-of-band channel)
Register
Register-L(P, gid, gs, m):
Parse mUUID $\leftarrow m$, Parse (pc^{gid}, pw^{gid}) $\leftarrow gs$
$(pk_{\mathsf{ZBox}}, sk_{\mathsf{ZBox}}) \xleftarrow{\$} \mathsf{ZBox.KGen}$
$\pi^{\mathsf{gid}}_P.sk \leftarrow sk_{\mathsf{ZBox}}$, $\pi^{\mathsf{gid}}_P.pk \leftarrow pk_{\mathsf{ZBox}}$
$\mathsf{Binding}^{\mathsf{gid}}_P \leftarrow \mathsf{gid} \parallel \mathsf{mUUID} \parallel \mathsf{uid}_P \parallel \mathsf{hid}_P \parallel \mathsf{st}_P.ipk \parallel \pi^{\mathsf{gid}}_P.pk$
$\sigma^{\mathsf{gid}}_P \leftarrow \mathsf{ZSign.Sign}(\mathsf{st}_P.isk, \mathsf{ctxt}_1, \mathsf{Binding}^{\mathsf{gid}}_P)$
$\pi^{\mathsf{gid}}_P.\mathsf{status} \leftarrow \mathsf{joined}$, $\pi^{\mathsf{gid}}_P.GP \xleftarrow{+} \mathsf{id}_P$
$\pi^{\mathsf{gid}}_P.gk \xleftarrow{\$} \{0,1\}^{256}$, $\pi^{\mathsf{gid}}_P.\mathsf{gkid} \leftarrow 1$
pc^{gid}, $m^{(P,\mathsf{gid})}_{\mathsf{Reg}} = (\pi^{\mathsf{gid}}_P.pk, \sigma^{\mathsf{gid}}_P)$
Server rejects if $m^{\mathsf{gid}}_{\mathsf{GSch}} \neq pc^{\mathsf{gid}}$
Register-P(P', gid, gs, m):
Parse mUUID $\leftarrow m$, Parse (pc^{gid}, pw^{gid}) $\leftarrow gs$
$(pk'_{\mathsf{ZBox}}, sk'_{\mathsf{ZBox}}) \xleftarrow{\$} \mathsf{ZBox.KGen}$
$\pi^{\mathsf{gid}}_{P'}.sk \leftarrow sk'_{\mathsf{ZBox}}$, $\pi^{\mathsf{gid}}_{P'}.pk \leftarrow pk'_{\mathsf{ZBox}}$
$\mathsf{Binding}^{\mathsf{gid}}_{P'} \leftarrow \mathsf{gid} \parallel \mathsf{mUUID} \parallel \mathsf{uid}_{P'} \parallel \mathsf{hid}_{P'} \parallel \mathsf{st}_{P'}.ipk \parallel \pi^{\mathsf{gid}}_{P'}.pk$
$\sigma^{\mathsf{gid}}_{P'} \leftarrow \mathsf{ZSign.Sign}(\mathsf{st}_{P'}.isk, \mathsf{ctxt}_1, \mathsf{Binding}^{\mathsf{gid}}_{P'})$
$\pi^{\mathsf{gid}}_P.\mathsf{status} \leftarrow \mathsf{registered}$
Run first pass of $\mathsf{PAKE_{PP}}(pw^{\mathsf{gid}})$ for $c^{(P',\mathsf{gid})}_{\mathsf{PP},1}$
pc^{gid}, $m^{(P',\mathsf{gid})}_{\mathsf{Reg}} = (\pi^{\mathsf{gid}}_{P'}.pk, \sigma^{\mathsf{gid}}_{P'})$, $c^{(P',\mathsf{gid})}_{\mathsf{PP},1}$
Participant Join
Join-L(P, $\mathsf{id}_{P'}$, gid, gs, m):
Parse (mUUID, $m^{P'}_{\mathsf{SignUp}}$, $m^{(P',\mathsf{gid})}_{\mathsf{Reg}}$, $c^{(P',\mathsf{gid})}_{\mathsf{PP},1}$) $\leftarrow m$
Parse (pc^{gid}, pw^{gid}) $\leftarrow gs$
Run second pass of $\mathsf{PAKE_{PP}}(pw^{\mathsf{gid}})$ for $k^{(P',\mathsf{gid})}_{\mathsf{PP}}$ and $c^{(P',\mathsf{gid})}_{\mathsf{PP},2}$
$\mathsf{nonce}^{P'}_{\mathsf{PP}} \xleftarrow{\$} \{0,1\}^{l_{\mathsf{PP}}}$, Store $k^{(P',\mathsf{gid})}_{\mathsf{PP}}$
Parse $ipk_{P'} \leftarrow m^{P'}_{\mathsf{SignUp}}$, $(pk^{\mathsf{gid}}_{P'}, \sigma^{\mathsf{gid}}_{P'}) \leftarrow m^{(P',\mathsf{gid})}_{\mathsf{Reg}}$
$\mathsf{Binding}^{\mathsf{gid}}_{P'} \leftarrow \mathsf{gid} \parallel \mathsf{mUUID} \parallel \mathsf{uid}_{P'} \parallel \mathsf{hid}_{P'} \parallel ipk_{P'} \parallel pk^{\mathsf{gid}}_{P'}$
$\mathsf{ZSign.Vrfy}(ipk_{P'}, \sigma^{\mathsf{gid}}_{P'}, \mathsf{ctxt}_1, \mathsf{Binding}^{\mathsf{gid}}_{P'})$
$\mathsf{Meta} \leftarrow \mathsf{gid} \parallel \mathsf{mUUID} \parallel \mathsf{uid}_P \parallel \mathsf{uid}_{P'}$
$c^{\mathsf{gid}}_{P'} \xleftarrow{\$} \mathsf{ZBox.Enc}(\pi^{\mathsf{gid}}_P.sk, pk^{\mathsf{gid}}_{P'}, \mathsf{ctxt}_2, \mathsf{ctxt}_3, \mathsf{Meta}, (\pi^{\mathsf{gid}}_P.gk, \pi^{\mathsf{gid}}_P.\mathsf{gkid}))$
$c^{\mathsf{gid}}_{P'} \leftarrow \mathsf{AEAD_{PP}.Enc}(k^{(P',\mathsf{gid})}_{\mathsf{PP}}, \mathsf{nonce}^{P'}_{\mathsf{PP}}, (m^P_{\mathsf{SignUp}}, m^{P'}_{\mathsf{SignUp}}), c^{\mathsf{gid}}_{P'})$
$\pi^{\mathsf{gid}}_P.GP \xleftarrow{+} \mathsf{id}_{P'}$, Store $pk^{\mathsf{gid}}_{P'}$
$\mathsf{id}_{P'}$, $c^{\mathsf{gid}}_{P'}$, $c^{(P',\mathsf{gid})}_{\mathsf{PP},2}$, $\mathsf{nonce}^{P'}_{\mathsf{PP}}$
Join-P(P', id_P, gid, gs, m):
Parse (mUUID, m^P_{SignUp}, $m^{(P,\mathsf{gid})}_{\mathsf{Reg}}$, $c^{\mathsf{gid}}_{P'}$, $c^{(P',\mathsf{gid})}_{\mathsf{PP},2}$, $\mathsf{nonce}^{P'}_{\mathsf{PP}}$) $\leftarrow m$
Parse (pc^{gid}, pw^{gid}) $\leftarrow gs$
Run $\mathsf{PAKE_{PP}}(pw^{\mathsf{gid}})$ for $k^{(P',\mathsf{gid})}_{\mathsf{PP}}$
$c^{\mathsf{gid}}_{P'} \leftarrow \mathsf{AEAD_{PP}.Dec}(k^{(P',\mathsf{gid})}_{\mathsf{PP}}, \mathsf{nonce}^{P'}_{\mathsf{PP}}, (m^P_{\mathsf{SignUp}}, m^{P'}_{\mathsf{SignUp}}), c^{\mathsf{gid}}_{P'})$
require $c^{\mathsf{gid}}_{P'} \neq \perp$, Store $k^{(P',\mathsf{gid})}_{\mathsf{PP}}$
Parse $ipk_P \leftarrow m^P_{\mathsf{SignUp}}$, $(pk^{\mathsf{gid}}_P, \sigma^{\mathsf{gid}}_P) \leftarrow m^{(P,\mathsf{gid})}_{\mathsf{Reg}}$
$\mathsf{Binding}^{\mathsf{gid}}_P \leftarrow \mathsf{gid} \parallel \mathsf{mUUID} \parallel \mathsf{uid}_P \parallel \mathsf{hid}_P \parallel ipk_P \parallel pk^{\mathsf{gid}}_P$
$\mathsf{ZSign.Vrfy}(ipk_P, \sigma^{\mathsf{gid}}_P, \mathsf{ctxt}_1, \mathsf{Binding}^{\mathsf{gid}}_P)$
$\mathsf{Meta} \leftarrow \mathsf{gid} \parallel \mathsf{mUUID} \parallel \mathsf{uid}_P \parallel \mathsf{uid}_{P'}$
$(gk, \mathsf{gkid}) \xleftarrow{\$} \mathsf{ZBox.Dec}(\pi^{\mathsf{gid}}_{P'}.sk, pk^{\mathsf{gid}}_P, \mathsf{ctxt}_2, \mathsf{ctxt}_3, \mathsf{Meta}, c_{P'})$
$\pi^{\mathsf{gid}}_{P'}.gk \leftarrow gk$, $\pi^{\mathsf{gid}}_{P'}.\mathsf{gkid} \leftarrow \mathsf{gkid}$, $\pi^{\mathsf{gid}}_{P'}.\mathsf{status} \leftarrow \mathsf{joined}$, Store pk^{gid}_P
Key Rotation
KeyRotat-L(P, gid, m):
require $m = \top$
$\pi^{\mathsf{gid}}_P.gk \xleftarrow{\$} \{0,1\}^{256}$, $\pi^{\mathsf{gid}}_P.\mathsf{gkid}{+}{+}$
foreach $\mathsf{id}_{\overline{P}} \in \pi^{\mathsf{gid}}_P.GP$ and $\mathsf{id}_{\overline{P}} \neq \mathsf{id}_P$:
$c^{\mathsf{gid}}_{\overline{P}} \xleftarrow{\$} \mathsf{ZBox.Enc}(\pi^{\mathsf{gid}}_P.sk, pk^{\mathsf{gid}}_{\overline{P}}, \mathsf{ctxt}_2, \mathsf{ctxt}_3, \mathsf{Meta}, (\pi^{\mathsf{gid}}_P.gk, \pi^{\mathsf{gid}}_P.\mathsf{gkid}))$
$\mathsf{nonce}^{\overline{P}}_{\mathsf{PP}} \xleftarrow{\$} \{0,1\}^{l_{\mathsf{PP}}}$
$c^{\mathsf{gid}}_{\overline{P}} \leftarrow \mathsf{AEAD_{PP}.Enc}(k^{(\overline{P},\mathsf{gid})}_{\mathsf{PP}}, \mathsf{nonce}^{\overline{P}}_{\mathsf{PP}}, (m^P_{\mathsf{SignUp}}, m^{\overline{P}}_{\mathsf{SignUp}}), c^{\mathsf{gid}}_{\overline{P}})$
$m^{(P,\mathsf{gid})}_{\mathsf{KRot}} = \{\mathsf{id}_{\overline{P}}, c^{\mathsf{gid}}_{\overline{P}}, \mathsf{nonce}^{\overline{P}}_{\mathsf{PP}}\}_{\overline{P}}$
KeyRotat-P(P, gid, m):
$(c^{\mathsf{gid}}_{P'}, \mathsf{nonce}^{P'}_{\mathsf{PP}}) \leftarrow m$
$c^{\mathsf{gid}}_{P'} \leftarrow \mathsf{AEAD_{PP}.Dec}(k^{(P',\mathsf{gid})}_{\mathsf{PP}}, \mathsf{nonce}^{P'}_{\mathsf{PP}}, (m^P_{\mathsf{SignUp}}, m^{P'}_{\mathsf{SignUp}}), c^{\mathsf{gid}}_{P'})$
require $c^{\mathsf{gid}}_{P'} \neq \perp$
$(gk, \mathsf{gkid}) \xleftarrow{\$} \mathsf{ZBox.Dec}(\pi^{\mathsf{gid}}_{P'}.sk, pk^{\mathsf{gid}}_P, \mathsf{ctxt}_2, \mathsf{ctxt}_3, \mathsf{Meta}, c^{\mathsf{gid}}_{P'})$
require $\pi^{\mathsf{gid}}_{P'}.\mathsf{gkid} < \mathsf{gkid}$
$\pi^{\mathsf{gid}}_{P'}.\mathsf{gkid} \leftarrow \mathsf{gkid}$, $\pi^{\mathsf{gid}}_{P'}.gk \leftarrow gk$
$c^{\mathsf{gid}}_{P'}$, $\mathsf{nonce}^{P'}_{\mathsf{PP}}$
Member Leave
Leave-L(P, gid, $\mathsf{id}_{P''}$):
if $\mathsf{id}_{P''} = \mathsf{id}_P$: $\pi^{\mathsf{gid}}_P \leftarrow \perp$ else $\pi^{\mathsf{gid}}_P.GP \xleftarrow{-} \mathsf{id}_{P''}$
Leave-P(P', gid, $\mathsf{id}_{P''}$):
if $\mathsf{id}_{P''} = \mathsf{id}_{P'}$: $\pi^{\mathsf{gid}}_{P'} \leftarrow \perp$

Figure 6.3: Overview of the **Zoom** protocol and our modified **ZoomPAKE** protocol. The boxes of the form $\boxed{\text{added}}$ denote the additions from our transformation for ZoomPAKE, i.e., which are not in current Zoom. Note we do require any additional message flows. The boxes of the form $\boxed{\text{redundant}}$ denote the elements that become redundant in our PAKE-based design: thus, for ZoomPAKE, we can essentially set pc^{gid} to the empty string, and still obtain the same guarantees, effectively replacing the old passcode by the new PAKE password. We recall the cryptographic algorithms ZSign and ZBox from the Zoom library in Section 6.5.1.1.

identifier $\mathsf{id}_{P'} = (\mathsf{uid}, \mathsf{hid})$, identity public key $\mathsf{st}_{P'}.ipk$, and per-group public key $\pi_{P'}^{\mathsf{gid}}.pk$. Then, it signs the binding information $\mathsf{Binding}_{P'}^{\mathsf{gid}}$ for a signature $\sigma_{P'}^{\mathsf{gid}}$ using $\mathsf{ZSign.Sign}$ algorithm, the identity secret key $\mathsf{st}_{P'}.isk$, and a context string ctxt_1. The passcode pc^{gid} and the output register message $m_{\mathsf{Reg}}^{(P',\mathsf{gid})}$ consisting of the per-group public key $\pi_{P'}^{\mathsf{gid}}.pk$ and the signature $\sigma_{P'}^{\mathsf{gid}}$ is sent to the server. The server adds $m_{\mathsf{Reg}}^{(P',\mathsf{gid})}$ to the "bulletin board" of the group gid, if the passcode pc^{gid} matches the group schedule message $m_{\mathsf{GSch}}^{\mathsf{gid}}$ received from the group leader, and rejects it otherwise. The status $\pi_{P'}^{\mathsf{gid}}.\mathsf{status}$ of the participant P' in the group gid is set to $\mathsf{registered}$.

- $\mathsf{Register\text{-}L}(P, \mathsf{gid}, gs, m)$: If the party P is the leader of the group gid, P runs the same execution as a participant except for setting the status $\pi_{P}^{\mathsf{gid}}.\mathsf{status}$ to joined rather than $\mathsf{registered}$. Moreover, the leader P initializes the group by sampling the first group key $\pi_{P}^{\mathsf{gid}}.gk$ of bit length 256 and sets the group key index $\pi_{P}^{\mathsf{gid}}.\mathsf{gkid}$ to 1. The identifier id_P is added into the local party set $\pi_{P}^{\mathsf{gid}}.GP$.

Participants Join $\mathsf{Join} = (\mathsf{Join\text{-}L}, \mathsf{Join\text{-}P})$***:*** The Zoom library executes this interactive sub-protocol between the leader P and a participant P' only one-pass:

- $\mathsf{Join\text{-}L}(P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m)$: When the leader P notices the joining request of a new participant P', P retrieves an incoming message m from the server and the group gid's "bulletin board" followed by parsing it into: (1) a server-generated randomness mUUID, (2) the participant P''s sign-up message $m_{\mathsf{SignUp}}^{P'}$, and (3) the participant P''s register message $m_{\mathsf{Reg}}^{(P',\mathsf{gid})}$. Next, the leader P parses the participant P''s identity public key $ipk_{P'}$, per-group public key $pk_{P'}^{\mathsf{gid}}$, and per-group signature $\sigma_{P'}^{\mathsf{gid}}$ from the incoming message, followed by using them to produce the participant's binding information $\mathsf{Binding}_{P'}^{\mathsf{gid}}$. If the binding information cannot pass the verification $\mathsf{ZSign.Vrfy}$ upon the participant's identity public key $ipk_{P'}$, signature $\sigma_{P'}^{\mathsf{gid}}$, and the context ctxt_1, then the leader aborts and undoes the previous executions. Otherwise, the leader creates a meta data Meta by concatenating the group identifier gid, server-generated randomness mUUID, the leader's user identifier uid_P, and the participant's user identifier $\mathsf{uid}_{P'}$. Finally, the leader P encrypts the current group key $\pi_{P}^{\mathsf{gid}}.gk$ as well as the index $\pi_{P}^{\mathsf{gid}}.\mathsf{gkid}$ using the $\mathsf{ZBox.Enc}$ encryption algorithm and the leader P's per-group secret key $\pi_{P}^{\mathsf{gid}}.sk$, the participant P''s per-group public key $\pi_{P'}^{\mathsf{gid}}.pk$, and auxiliary information ctxt_2, ctxt_3, and Meta. The identifier of the participant P' and the ZBox ciphertext $c_{P'}^{\mathsf{gid}}$ are send to P' via the server. The leader P stores the identifier $\mathsf{id}_{P'}$ of the participant P' into the local party set $\pi_{P}^{\mathsf{gid}}.GP$ and stores the participant's per-group public key $pk_{P'}^{\mathsf{gid}}$.

- $\mathsf{Join\text{-}P}(P', \mathsf{id}_P, \mathsf{gid}, gs, m)$: When a participant P', who registered for a group gid, receives an incoming message m that includes (1) a server-generated randomness mUUID, (2) the leader P's sign-up message m_{SignUp}^{P}, (3) the leader P's register message $m_{\mathsf{Reg}}^{(P,\mathsf{gid})}$, and (4) the leader's reply $c_{P'}^{\mathsf{gid}}$, P' first parses the incoming message, creates the leader's binding information, and verifies it using $\mathsf{ZSign.Vrfy}$ algorithm, similar to the leader's execution. Then, P' also generates the meta data Meta and uses it together with its own per-group secret key $\pi_{P'}^{\mathsf{gid}}.sk$, the leader's per-group public key pk_{P}^{gid}, and the contexts ctxt_2 and ctxt_3, to decrypt the ciphertext $c_{P'}^{\mathsf{gid}}$. If any error occurs during above steps, then the participant P' aborts and undoes the previous executions. Otherwise, the participant P' stores the decrypted group key as well as the associated index into the per-group state $\pi_{P'}^{\mathsf{gid}}.gk$ and $\pi_{P'}^{\mathsf{gid}}.\mathsf{gkid}$, followed by setting the status $\pi_{P'}^{\mathsf{gid}}.\mathsf{status}$ to joined. Moreover, P' stores the leader P's per-group public key into the state. [4]

Key Rotation $\mathsf{KeyRotat} = (\mathsf{KeyRotat\text{-}L}, \mathsf{KeyRotat\text{-}P})$***:*** The execution of this algorithm is distinct according to the caller P's role: a leader or a participant. We separate the description for $\mathsf{KeyRotat\text{-}L}(P, \mathsf{gid}, m)$, where P is the leader of the group gid, and for $\mathsf{KeyRotat\text{-}P}(P', \mathsf{gid}, m)$, where P' is a participant of the group gid.

- $\mathsf{KeyRotat\text{-}L}(P, \mathsf{gid}, m)$: The leader P of the group gid executes this algorithm without any auxiliary incoming input, i.e., $m = \top$. The leader P samples a new group key $\pi_{P}^{\mathsf{gid}}.gk$ of bit length 256 and increments the corresponding index $\pi_{P}^{\mathsf{gid}}.\mathsf{gkid}$ by 1. Similar to the encryption during the Participant Join phase, the leader encrypts the new group key and index for each party in its local party set $\pi_{P}^{\mathsf{gid}}.GP$ except for himself. The output is a ciphertext bundle that includes the identifiers of each participant and the customized ciphertexts.

The server is expected to split the ciphertext bundles and to send each ciphertext to the specified participant.

- $\mathsf{KeyRotat\text{-}P}(P', \mathsf{gid}, m)$: The participant P' of the group gid first parses the incoming message m from the server to an ZBox ciphertext $ct_{P'}^{\mathsf{gid}}$. Then, P' decrypts the new group key gk and index gkid as during the Participant Join phase. If any error occurs in the above steps or the decrypted group key index is smaller than or equal to the local one, then the participant P' aborts and undoes the previous executions. Otherwise, P' simply overwrites the local group key as well as the index by the new ones.

Member Leave $\mathsf{Leave} = (\mathsf{Leave\text{-}L}, \mathsf{Leave\text{-}P})$***:*** The execution of this algorithm is distinct according to the caller P's role: a leader or a participant. We separate the description for $\mathsf{Leave\text{-}L}(P, \mathsf{gid}, \mathsf{id}_{P''})$, where the P is the leader of the group gid, and for $\mathsf{Leave\text{-}P}(P', \mathsf{gid}, \mathsf{id}_{P''})$, where P' is a participant of the group gid.

[4] In practice, Zoom has an independent mechanism for leader P to synchronize the party set $\pi_{P}^{\mathsf{gid}}.GP$ with every participant P'. We omit it here since, this does not impact Zoom security analysis in our models.

- $\mathsf{Leave\text{-}L}(P, \mathsf{gid}, \mathsf{id}_{P''})$: If $\mathsf{id}_P = \mathsf{id}_{P''}$, i.e., the leader P wants to leave the group gid, then P erases the per-group instance π_P^{gid}. Otherwise, the leader P notices a party P'' leaving the group gid, P simply removes the identifier of the participant $\mathsf{id}_{P''}$ from the local party set $\pi_P^{\mathsf{gid}}.GP$.
- $\mathsf{Leave\text{-}P}(P', \mathsf{gid}, \mathsf{id}_{P''})$: If $\mathsf{id}_{P'} = \mathsf{id}_{P''}$, i.e., the participant P' wants to leave the group gid, then P' erases the per-group instance $\pi_{P'}^{\mathsf{gid}}$. Otherwise, P' performs no action.

Instantiations. Underlying the ZSign and ZBox algorithms, the key derivation function H_1 is SHA256 and H_2 is HKDF algorithm (using an empty salt parameter). The length l underlying ZBox algorithm is 192. The elliptic curve ECDH underlying Diffie-Hellman key exchange is Curve25519. The ZSign algorithm relies DS on EdDSA over Ed25519. The AEAD algorithm is xchacha20poly1305.

6.5.2 Zoom is Sec-mGKD-pki secure

We omit the correctness analysis of the Zoom library. Below, we investigate the provable security of the Zoom library. The proof of the following theorem is give in Section 6.9.1.

Theorem 25. *Let Π denote the end-to-end Zoom protocol in Section 6.5.1. Assume the $\epsilon_{\mathsf{H}_1}^{\text{coll-res}}$-collision resistance of the underlying H_1, the $\epsilon_{\mathsf{DS}}^{\text{euf-cma}}$-EUF-CMA security of DS, the $\epsilon_{\mathsf{AEAD}}^{(n,m)\text{-frob}}$-$(n,m)$-FROB security, $\epsilon_{\mathsf{AEAD}}^{\text{ind\$-cca}}$-IND\$-CCA security, and $\epsilon_{\mathsf{AEAD}}^{\text{cti-cpa}}$-CTI-CPA security of the AEAD. Assume the $\epsilon_{\mathsf{ECDH},\mathsf{H}_2}^{\text{mn-prf-ODH}}$ hardness of the mn-prf-ODH problem over ECDH and function H_2. The advantage of any PPT attacker $\mathcal{A}$ that breaks the Sec-mGKD-pki security of Π is bounded by,*

$$\mathsf{Adv}_{\Pi}^{\text{Sec-mGKD-pki}}(\mathcal{A}) \leq \epsilon_{\mathsf{H}_1}^{\text{coll-res}} + q_{\textsc{NewParty}}\epsilon_{\mathsf{DS}}^{\text{euf-cma}} + c_{\mathsf{maxReg}}q_{\textsc{NewGroup}}\Big(\epsilon_{\mathsf{AEAD}}^{\text{cti-cpa}} + \epsilon_{\mathsf{AEAD}}^{(n,m)\text{-frob}} + c_{\mathsf{maxReg}}^{(n_{\mathsf{party}}-1)}(n_{\mathsf{party}}-1)(\epsilon_{\mathsf{ECDH},\mathsf{H}_2}^{\text{mn-prf-ODH}} + \epsilon_{\mathsf{AEAD}}^{\text{ind\$-cca}})\Big)$$

where c_{maxParty} denotes the maximal number of parties in every group, c_{maxReg} denotes the maximal number of register requests for every group, $n_{\mathsf{party}} \leq c_{\mathsf{maxParty}}$ denotes the number of parties in the set $GP^{(\mathsf{gid},\mathsf{gkid})}$ for tested group identifier gid and group key index gkid, and $q_{\mathcal{O}}$ denote the maximal number of the queries to any oracle $\mathcal{O}$.

Proof Sketch. By the collision resistance of H_1, the DS scheme underlying ZSign algorithm never signs two identical input. The euf-cma security of DS then ensures that all parties obtains other parties' honest sign-up messages in every group. Then, we consider two cases. If $\mathcal{A}$ can win via the event E_{KAuth}, then we can easily guess the group identifier $\widetilde{\mathsf{gid}}$ with some leader $P^{\widetilde{\mathsf{gid}}}$ and party $\widetilde{P}$ that trigger $\mathcal{A}$ to win with probability at least $1/c_{\mathsf{maxReg}}q_{\textsc{NewGroup}}$. By mn-prf-ODH security of ECDH and H_2 underlying ZBox algorithm, the key of AEAD between $P^{\widetilde{\mathsf{gid}}}$ and $\widetilde{P}$ is indistinguishable from random. If $\mathcal{A}$ can win via the event E_{KAuth}, then the attacker must be able to either forge a ciphertext or a nonce of AEAD, which breaks either the $\mathsf{CTI\text{-}CPA}$ security of (n, m)-FROB security of AEAD.

If $\mathcal{A}$ win via the event E_{KPriv}, then we can easily guess the group identifier $\widetilde{\mathsf{gid}}$ with some leader $P^{\widetilde{\mathsf{gid}}}$ and the identifier of all parties in the set $GP^{(\widetilde{\mathsf{gid}},\widetilde{\mathsf{gkid}})}$ for the tested group key index $\widetilde{\mathsf{gkid}}$ with probability at least $1/c_{\mathsf{maxParty}} c_{\mathsf{maxReg}}^{(n_{\mathsf{party}}-1)} q_{\textsc{NewGroup}}$. By a sequence of $(n_{\mathsf{party}} - 1)$ hybrid games, we know that all keys and ciphertext of AEAD between every participant in $\widetilde{\mathsf{gid}}$ and the leader $P^{\widetilde{\mathsf{gid}}}$ should be indistinguishable from random due to the $\mathsf{mn\text{-}prf\text{-}ODH}$ security of ECDH and H_2 and $\mathsf{ind\$\text{-}cca}$ security of AEAD. Thus, $\mathcal{A}$ cannot win via E_{KPriv}. $\square$

Note that H_1 = SHA256 provides an expected collision resistance of 128 bits [78]. The $\mathsf{EUF\text{-}CMA}$ security of Ed25519 was proven in [61]. The security of xchacha20poly1305 was discussed in [15]. The maximal number of parties per meeting is $c_{\mathsf{maxParty}} = 1000$ [53]. The above theorem shows that the end-to-end Zoom library provably provides $\mathsf{Sec\text{-}mGKD\text{-}pki}$ security and satisfies all properties listed in Section 6.4.3.

Remark 1. *Theorem 25 shows that Zoom achieves* $\mathsf{Sec\text{-}mGKD\text{-}pki}$ *independent of the passcode: the passcode is only used implicitly for access control by honest servers.*

6.5.3 Zoom is not $\mathsf{Sec\text{-}mGKD\text{-}pw}$ secure

Recall that the $\mathsf{Sec\text{-}mGKD\text{-}pki}$ model has two restrictions in Section 6.4.3. A natural question arises whether these restrictions apply to the $\mathsf{Sec\text{-}mGKD\text{-}pki}$ secure Zoom library.

Does Zoom Provide Trusted PKI? The PKI is expected to "*enable users of an insecure public network such as the Internet to securely and privately exchange data through the use of a public and a private cryptographic key pair that is obtained and shared through a trusted authority*" [174, Charpter 1]. As we mentioned in Section 6.5.1, all public keys of all parties in the Zoom library are uploaded to an infrastructure, called "identity management system", that is fully controlled by Zoom. The identity management system distributes the identity public keys. While Zoom claims the end-to-end security, the goal of which is to protect the secrecy and integrity of the exchanged content between every two parties against all third parties including the service providers, assuming Zoom-controlled PKI trusted is controversial and doubtful. Considering a malicious server, the server can easily perform the "machine-in-the-middle" attack by forging the sign-up messages and impersonate any party towards others.

Although there do exist other group meeting providers, such as Cisco WebEx and Skype, that employ a third-party PKI, such as Microsoft Certificate Authority (CA), we stress that the reliability of PKI is still imperfect. Eckersley and Burns [86] revealed that 14 CAs had been compromised. Moreover, a number of attacks that successfully break several CAs, including DigiNotar, Comodo, GlobalSign, StartSSL, and TurkTrust, have been publicly noticed [171]. It is prudent to consider the potential PKI compromise.

Does Zoom Provide (Implicit) Group Member Authentication? Unfortunately, this does not hold for Zoom. Although the end-to-end Zoom library asks every leader to create every new group together with an associated passcode, the leaders hand over the power of passcode verification to the untrusted server. By colluding with the untrusted server, an unauthorized (and malicious) party can join every group without the knowledge of any passcode. The consequence is that nobody in the group (including the leader) can distinguish authorized participants from the others, in particular, in the end-to-end setting.

6.6 A Generic Approach to Sec-mGKD-pw Security: Password-Protected Transformation

If we could assume that each group gid has a unique high-entropy group secret gs^{gid} only shared by the leader and authorized participants of the group gid, we could design a trivial construction that meets the Sec-mGKD-pw security. We can simply use a message authentication code MAC with the group secret gs^{gid} as key to sign and verify all outgoing and incoming messages.

However, in practice we use low-entropy passwords for usability, allowing the passwords to be shared over various out-of-band channels. For instance, real-world service providers often support only short passwords[5]. This restricts the upper bound of the password entropy and enables attackers to perform dictionary attacks on the password, e.g., by brute force guessing.

In this section, we introduce a generic Password-Protected (PP) transformation that provably transforms any Sec-mGKD-pki secure mGKD protocol Π to another Sec-mGKD-pw secure $\Pi' = \mathsf{PP}[\Pi, \mathsf{PAKE_{PP}}, \mathsf{AEAD_{PP}}]$ protocol by using a password-authenticated key exchange $\mathsf{PAKE_{PP}}$ and an authenticated encryption with associated data $\mathsf{AEAD_{PP}}$. We also prove that the PP transformation preserves Sec-mGKD-pki security, i.e., if Π is Sec-mGKD-pki secure, so is Π'. In this sense, Π' satisfies stronger security Sec-mGKD-pw+, due to Corollary 1. Finally, we illustrate how to apply our PP transformation to the Zoom library, and provide efficient instantiations for $\mathsf{PAKE_{PP}}$ and $\mathsf{AEAD_{PP}}$, without causing additional message flows.

[5]For instance, meeting passcodes in Zoom are 1-16 digit numeric lock codes; the default meeting password in Cisco WebEx has ≥ 11 characters.

6.6.1 The Generic Transformation

The goal of our PP transformation is to ensure that only the authorized parties that know the group secret can recover any group key, even if the server is malicious. The high-level overview of our PP transformation is to (1) let the leader and every participant run a $\mathsf{PAKE}_{\mathsf{PP}}$ protocol upon a new password (included in the group secret) to produce a symmetric key k_{PP} during the Participant Join phase, and (2) use the key k_{PP} and an $\mathsf{AEAD}_{\mathsf{PP}}$ scheme to encrypt/decrypt the original transcript of the mGKD protocol Π during the Participant Join and Key Rotation phases. Note that every participant has to first register for a group before joining it. To avoid introducing additional message flows, we design our PP transformation to shift the first pass of $\mathsf{PAKE}_{\mathsf{PP}}$ to the participant's Register phase. We give the formal definition of our PP transformation below.

Definition 59. *Let* $\Pi = (\mathsf{SignUp}, \mathsf{Schedule}, \mathsf{Register}, \mathsf{Join}, \mathsf{Leave}, \mathsf{KeyRotat})$ *denote a multi-stage group key distribution protocol. Let* $\mathsf{PAKE}_{\mathsf{PP}}$ *denote a password-authenticated key exchange scheme. Let* $\mathsf{AEAD}_{\mathsf{PP}}$ *denote an authenticated encryption with associated data. We define the password-protected (*PP*) transformation* $\mathsf{PP}[\Pi, \mathsf{PAKE}_{\mathsf{PP}}, \mathsf{AEAD}_{\mathsf{PP}}]$ *that outputs* $\Pi' = (\mathsf{SignUp}', \mathsf{Schedule}', \mathsf{Register}', \mathsf{Join}', \mathsf{Leave}', \mathsf{KeyRotat}')$ *as follows:*

Sign Up $\mathsf{SignUp}'(P)$**:** *Run* $m^{P}_{\mathsf{SignUp}} \xleftarrow{\$} \mathsf{SignUp}(P)$ *and stores* m^{P}_{SignUp} *locally into the long-term state* st_P*.*

Group Schedule $\mathsf{Schedule}'(P, \mathsf{gid}, gs)$**:** *Parse* $(gs^{\mathsf{gid}}_{\Pi}, pw^{\mathsf{gid}}) \leftarrow gs$*, run* $m^{\mathsf{gid}}_{\mathsf{GSch}} \xleftarrow{\$} \mathsf{Schedule}(P, \mathsf{gid}, gs^{\mathsf{gid}}_{\Pi})$*, and output* $m^{\mathsf{gid}}_{\mathsf{GSch}}$*. The full group secret* gs *is sent to authorized parties over out-of-band channels.*

Register $\mathsf{Register}' = (\mathsf{Register\text{-}L}', \mathsf{Register\text{-}P}')$**:** *We define the sub-algorithms as follows:*

- $\mathsf{Register\text{-}L}'(P, \mathsf{gid}, gs, m)$*: Parse* $(gs^{\mathsf{gid}}_{\Pi}, pw^{\mathsf{gid}}) \leftarrow gs$*, run* $m' \xleftarrow{\$} \mathsf{Register\text{-}L}(P, \mathsf{gid}, gs^{\mathsf{gid}}_{\Pi}, m)$*, and output* m'*.*
- $\mathsf{Register\text{-}P}'(P, \mathsf{gid}, gs, m)$*: First, parse* $(gs^{\mathsf{gid}}_{\Pi}, pw^{\mathsf{gid}}) \leftarrow gs$*. Next, execute* $m' \xleftarrow{\$} \mathsf{Register\text{-}P}(P, \mathsf{gid}, gs^{\mathsf{gid}}_{\Pi}, m)$*. Then, run the first pass of* $\mathsf{PAKE}_{\mathsf{PP}}$ *upon the password* pw^{gid} *for a ciphertext* $c^{(P,\mathsf{gid})}_{\mathsf{PP}}$*. Finally, output* $(m', c^{(P,\mathsf{gid})}_{\mathsf{PP}})$*.*

Participant Join $\mathsf{Join}' = (\mathsf{Join\text{-}L}', \mathsf{Join\text{-}P}')$**:** *This phase consists of two steps. In either step, if any error occurs during this algorithm, the caller* P *aborts and undoes the executions. In the first step, the leader and the participant run* $\mathsf{PAKE}_{\mathsf{PP}}$ *until* $\mathsf{PAKE}_{\mathsf{PP}}$ *outputs a key* k_{PP}*.*

- $\mathsf{Join\text{-}L}'(P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m)$ *or* $\mathsf{Join\text{-}P}'(P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m)$*: The caller* P *first parses* $(gs^{\mathsf{gid}}_{\Pi}, pw^{\mathsf{gid}}) \leftarrow gs$ *from the group secret and other necessary information for running* $\mathsf{PAKE}_{\mathsf{PP}}$ *from the input message* m*. Then,* P *runs the next pass of* $\mathsf{PAKE}_{\mathsf{PP}}$ *on* pw^{gid}*. If the key* k_{PP} *is still unavailable,* P *directly outputs the outgoing message of* $\mathsf{PAKE}_{\mathsf{PP}}$*. Otherwise, the key* k_{PP} *is stored into the per-group state* π^{gid}_{P}*.*

If the leader and the participant have computed the key k_{PP} before this algorithm invocation or in the first step in this invocation, they execute the following second step.

- $\mathsf{Join\text{-}L}'(P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m)$: *The leader P first parses $(gs_{\Pi}^{\mathsf{gid}}, pw^{\mathsf{gid}}) \leftarrow gs$ from the group secret. If the original* Join-L *algorithm needs any incoming information from the participant P', then the leader P extracts an* $\mathsf{AEAD}_{\mathsf{PP}}$ *ciphertext and an* $\mathsf{AEAD}_{\mathsf{PP}}$ *nonce from the input m. Next, the leader P decrypts the* $\mathsf{AEAD}_{\mathsf{PP}}$ *ciphertext using the key k_{PP}, the* $\mathsf{AEAD}_{\mathsf{PP}}$ *nonce, and an header consisting of both parties' sign-up messages, and obtains a message m_1. Then, the leader P extracts other necessary information m_2 from the input m for running $m' \xleftarrow{\$} \mathsf{Join\text{-}L}(P, \mathsf{id}_{P'}, \mathsf{gid}, gs_{\Pi}^{\mathsf{gid}}, m_1 \parallel m_2)$. After that, P encrypts m' using the* $\mathsf{AEAD}_{\mathsf{PP}}$ *key k_{PP}, a random nonce, and an header consisting of both parties' sign-up messages. Finally, the leader P outputs the* $\mathsf{AEAD}_{\mathsf{PP}}$ *ciphertext and nonce.*
- $\mathsf{Join\text{-}P}'(P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m)$: *The participant P first parses $(gs_{\Pi}^{\mathsf{gid}}, pw^{\mathsf{gid}}) \leftarrow gs$ from the group secret. If the original* Join-P *algorithm needs any incoming information from the leader P', then the participant P extracts an* $\mathsf{AEAD}_{\mathsf{PP}}$ *ciphertext and an* $\mathsf{AEAD}_{\mathsf{PP}}$ *nonce from the input m. Next, the participant P decrypts the* $\mathsf{AEAD}_{\mathsf{PP}}$ *ciphertext using the key k_{PP}, the* $\mathsf{AEAD}_{\mathsf{PP}}$ *nonce, and an header consisting of both parties' sign-up messages, and obtains a message m_1. Then, the participant P extracts other necessary information m_2 from the input m for running $m' \xleftarrow{\$} \mathsf{Join\text{-}P}(P, \mathsf{id}_{P'}, \mathsf{gid}, gs_{\Pi}^{\mathsf{gid}}, m_1 \parallel m_2)$. After that, P encrypts m' using the* $\mathsf{AEAD}_{\mathsf{PP}}$ *key k_{PP}, a random nonce, and an header consisting of both parties' sign-up messages. Finally, the participant P outputs the* $\mathsf{AEAD}_{\mathsf{PP}}$ *ciphertext and nonce.*

Member Leave $\mathsf{Leave}' = (\mathsf{Leave\text{-}L}', \mathsf{Leave\text{-}P}')$**:** *These algorithms are identical to the original* $\mathsf{Leave} = (\mathsf{Leave\text{-}L}, \mathsf{Leave\text{-}P})$. *Note that if a per-group state is erased, then the stored key k_{PP} must also be erased.*

Key Rotation $\mathsf{KeyRotat}' = (\mathsf{KeyRotat\text{-}L}', \mathsf{KeyRotat\text{-}P}')$: *We define the sub-algorithms as follows. If any error occurs during the above execution, then the caller aborts and undoes the executions in this invocation.*

- $\mathsf{KeyRotat\text{-}L}'(P, \mathsf{gid}, m)$: *The leader P first runs the original $m_{\mathsf{KRot}} \xleftarrow{\$} \mathsf{KeyRotat\text{-}L}(P, \mathsf{gid}, m)$. Then, the leader P extracts the portion $c_{\overline{P}}$ in m_{KRot} that is specific to every participant $\overline{P}$ in the group* gid, *followed by encrypting it using the stored corresponding* $\mathsf{AEAD}_{\mathsf{PP}}$ *key k_{PP}, a random nonce, and an header consisting of both parties' sign-up messages as in the Participant Join phase. Finally, the leader P outputs the* $\mathsf{AEAD}_{\mathsf{PP}}$ *ciphertext and nonce for every participant $\overline{P}$ in the group* gid.
- $\mathsf{KeyRotat\text{-}P}'(P, \mathsf{gid}, m)$: *The participant P first extracts an* $\mathsf{AEAD}_{\mathsf{PP}}$ *ciphertext and an* $\mathsf{AEAD}_{\mathsf{PP}}$ *nonce from the input m. Next, P recovers a message m_1 from the* $\mathsf{AEAD}_{\mathsf{PP}}$ *ciphertext using the stored corresponding* $\mathsf{AEAD}_{\mathsf{PP}}$ *key k_{PP}, the* $\mathsf{AEAD}_{\mathsf{PP}}$

nonce, and an header consisting of both parties' sign-up messages as in the Participant Join phase. Then, the participant P extracts other necessary information m_2 from the input m for running $m_{\mathsf{KRot}} \xleftarrow{\$} \mathsf{KeyRotat\text{-}P}(P, \mathsf{gid}, m_1 \parallel m_2)$ and outputting m_{KRot}.

For brevity we omit the correctness analysis. The theorem below shows that our PP transformation provably turns a Sec-mGKD-pki secure Π into a Sec-mGKD-pw secure $\Pi' = \mathsf{PP}[\Pi, \mathsf{PAKE}_{\mathsf{PP}}, \mathsf{AEAD}_{\mathsf{PP}}]$ protocol. We give the theorem's proof in Section 6.9.2.

Theorem 26. *Let Π denote a* mGKD *protocol. Let $\mathsf{PAKE}_{\mathsf{PP}}$ denote a password-authenticated key exchange scheme. Let $\mathsf{AEAD}_{\mathsf{PP}}$ denote an authenticated encryption with associated data scheme. Let $\Pi' = \mathsf{PP}[\Pi, \mathsf{PAKE}_{\mathsf{PP}}, \mathsf{AEAD}_{\mathsf{PP}}]$. Let $\mathcal{D} = \mathcal{D}_\Pi \times \mathcal{D}_{pw}$ denote the distribution of the group secrets. Assume the $\epsilon^{\mathsf{w\text{-}PAKE}}_{\mathsf{PAKE}_{\mathsf{PP}},\mathcal{D}_{pw}}$-w-PAKE security of the underlying $\mathsf{PAKE}_{\mathsf{PP}}$, the $\epsilon^{d\text{-}\mathsf{frob}}_{\mathsf{AEAD}_{\mathsf{PP}}}$-d-FROB security, $\epsilon^{\mathsf{ind\$\text{-}cca}}_{\mathsf{AEAD}_{\mathsf{PP}}}$-IND\$-CCA security, and $\epsilon^{\mathsf{cti\text{-}cpa}}_{\mathsf{AEAD}_{\mathsf{PP}}}$-CTI-CPA security of $\mathsf{AEAD}_{\mathsf{PP}}$. If there exists any PPT attacker $\mathcal{A}$ that breaks the Sec-mGKD-pw security of Π', then there must exist a PPT attacker $\mathcal{B}$ that breaks the Sec-mGKD-pki security of Π such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\Pi'}(\mathcal{A}) \leq & q_{\text{NewGroup}}\Big(\epsilon^{\mathsf{w\text{-}PAKE}}_{\mathsf{PAKE}_{\mathsf{PP}},\mathcal{D}_{pw}} + \epsilon^{d\text{-}\mathsf{frob}}_{\mathsf{AEAD}_{\mathsf{PP}}} \\ & + c_{\mathsf{maxReg}}(\epsilon^{\mathsf{cti\text{-}cpa}}_{\mathsf{AEAD}_{\mathsf{PP}}} + \epsilon^{\mathsf{ind\$\text{-}cca}}_{\mathsf{AEAD}_{\mathsf{PP}}}) + \mathsf{Adv}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\Pi}(\mathcal{B})\Big)\end{aligned}$$

where c_{maxReg} denotes the maximal number of register requests for every group and $q_{\mathcal{O}}$ denotes the maximal number of queries to any oracle $\mathcal{O}$.

Proof Sketch. We can easily guess the group $\widetilde{\mathsf{gid}}$ that enables $\mathcal{A}$ to win with probability at least q_{NewGroup}. Due to the w-PAKE security of the $\mathsf{PAKE}_{\mathsf{PP}}$ scheme with $\mathcal{D}_{pw}$, we can ensure that the keys $k_{\mathsf{PP}}^{(P',\widetilde{\mathsf{gid}})}$ of all authorized participants P' in the group $\widetilde{\mathsf{gid}}$, which are output by $\mathsf{PAKE}_{\mathsf{PP}}$ of and will be used for $\mathsf{AEAD}_{\mathsf{PP}}$, are random. Moreover, the key $k_{\mathsf{PP}}^{(P',\widetilde{\mathsf{gid}})}$ produced by the leader $P^{\widetilde{\mathsf{gid}}}$ of the group $\widetilde{\mathsf{gid}}$ is either same as the one produced by the corresponding authorized participant P' or independently random. By c_{maxReg} hybrid games on the CTI-CPA security of $\mathsf{AEAD}_{\mathsf{PP}}$, all authorized parties P' in the group $\widetilde{\mathsf{gid}}$ must agree on all $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertexts with the leader $P^{\widetilde{\mathsf{gid}}}$. By d-frob security of $\mathsf{AEAD}_{\mathsf{PP}}$, agreeing ciphertexts indicates that agreeing on the header, i.e., sign-up messages. Moreover, by the ind\$-cca security of AEAD $\mathsf{AEAD}_{\mathsf{PP}}$, all $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertexts produced by $P^{\widetilde{\mathsf{gid}}}$ for any unauthorized party are indistinguishable from random and therefore leaks no information about any group keys. Finally, if $\mathcal{A}$ win via the event E_{KAuth} or E_{KPriv} against Π', then $\mathcal{A}$ can also win the same event against Π's Sec-mGKD-pki security. □

Below, we further show that our PP transformation preserves the Sec-mGKD-pki security of the original mGKD protocol Π. We give the theorem's proof in Section 6.9.3.

Theorem 27. *Let* Π *denote a* mGKD *protocol. Let* $\mathsf{PAKE_{PP}}$ *denote a password-authenticated key exchange scheme. Let* $\mathsf{AEAD_{PP}}$ *denote an authenticated encryption with associated data. Let* $\Pi' = \mathsf{PP}[\Pi, \mathsf{PAKE_{PP}}, \mathsf{AEAD_{PP}}]$. *If there exists any* PPT *attacker* $\mathcal{A}$ *that breaks the* $\mathsf{Sec\text{-}mGKD\text{-}pki}$ *security of* Π'*, then there exists another* PPT *attacker* $\mathcal{B}$ *that breaks the* $\mathsf{Sec\text{-}mGKD\text{-}pki}$ *security of* Π *such that*

$$\mathsf{Adv}_{\Pi'}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}(\mathcal{A}) \leq \mathsf{Adv}_{\Pi}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}(\mathcal{B})$$

Proof Sketch. The proof can be easily given by a reduction. The attacker $\mathcal{B}$ can easily simulates the real $\mathsf{Sec\text{-}mGKD\text{-}pki}$ game against Π' to $\mathcal{A}$ by sampling all passwords for all groups and runs the PP transformation by himself. All information that $\mathcal{B}$ needs can be obtained by querying its challenger. Finally, $\mathcal{B}$ forwards the bit output by $\mathcal{A}$ and wins whenever $\mathcal{A}$ wins. □

Combining these two theorems, our PP transformation provably endows a $\mathsf{Sec\text{-}mGKD\text{-}pki}$ secure mGKD protocol Π with $\mathsf{Sec\text{-}mGKD\text{-}pw}$ security while preserving the original one, i.e., $\mathsf{Sec\text{-}mGKD\text{-}pw+}$ security due to Corollary 1.

6.6.2 Application to the Zoom Library

Then, we illustrate how to apply our PP transformation to the Zoom library in Section 6.5.1 by using a 2-pass $\mathsf{PAKE_{PP}}$ scheme and an $\mathsf{AEAD_{PP}}$ scheme. We call the transformed version ZoomPAKE and show the resulting protocol in Figure 6.3, where we use boxes to indicate the modifications. Note that Zoom achieves $\mathsf{Sec\text{-}mGKD\text{-}pki}$ security without relying on passcodes, as stated in Remark 1. The passcodes for the server's access control underlying Zoom are redundant in the stronger threat model, and can therefore be set to the empty string without impacting $\mathsf{Sec\text{-}mGKD\text{-}pw}$ security. In the following we assume that the passcode is set to the empty string, which amounts to replacing the passcode with the new PAKE password computations.

The Sign Up and Member Leave phases are unchanged.

ZoomPAKE Group Schedule Phase: This algorithm is nearly identical to the original one except that the leader sends the new password to authorized parties over an out-of-band channel instead of sending the passcode to the server.

ZoomPAKE Register Phase: Similar to the previous, parties no longer need to send the passcode to the server. Then, each participant P' runs the first pass of $\mathsf{PAKE_{PP}}$ on the password pw^{gid} for a ciphertext $c_{\mathsf{PP},1}^{(P',\mathsf{gid})}$ and outputs both the original outgoing messages and $c_{\mathsf{PP},1}^{(P',\mathsf{gid})}$.

ZoomPAKE Participant Join Phase: Our PP transformation modifies both leaders' and participants' execution.

From the leader P's side, P firsts parses an additional $\mathsf{PAKE_{PP}}$ ciphertext $c_{\mathsf{PP},1}^{(P',\mathsf{gid})}$ from the input, and uses the group secret as the password pw^{gid}. Next, the leader P runs the second pass of $\mathsf{PAKE_{PP}}$ with necessary input for a key $k_{\mathsf{PP}}^{(P',\mathsf{gid})}$ and a ciphertext $c_{\mathsf{PP},2}^{(P',\mathsf{gid})}$. Then, P stores $k_{\mathsf{PP}}^{(P',\mathsf{gid})}$ and samples a random nonce of length l_{PP} uniformly at random. After that, P executes the original computation. When the ZBox encryption $c_{P'}^{\mathsf{gid}}$ is derived, the leader P further re-encrypts it using the key $k_{\mathsf{PP}}^{(P',\mathsf{gid})}$, the random nonce $\mathsf{nonce}_{\mathsf{PP}}^{P'}$, and the header consisting both parties' sign-up messages. The $\mathsf{PAKE_{PP}}$ ciphertext $c_{\mathsf{PP},2}^{(P',\mathsf{gid})}$ and the $\mathsf{AEAD_{PP}}$ ciphertext are output.

From the participant P''s side, P' first parses two more components from the input messages: a $\mathsf{PAKE_{PP}}$ ciphertext $c_{\mathsf{PP},2}^{(P',\mathsf{gid})}$ and a nonce $\mathsf{nonce}_{\mathsf{PP}}^{P'}$. The participant P' also uses the group secret gs as the password pw^{gid}. Note that the original ciphertext $c_{P'}^{\mathsf{gid}}$ is not the one of ZBox anymore but the one of $\mathsf{AEAD_{PP}}$. Next, P' runs $\mathsf{PAKE_{PP}}$ for a key $k_{\mathsf{PP}}^{(P',\mathsf{gid})}$ and decrypts the $\mathsf{AEAD_{PP}}$ ciphertext $c_{P'}^{\mathsf{gid}}$ using the key $k_{\mathsf{PP}}^{(P',\mathsf{gid})}$, the nonce $\mathsf{nonce}_{\mathsf{PP}}^{P'}$, and the header consisting of the leader P's and the participant P''s sign-up messages, for the original ZBox ciphertext. If any error occurs during this step, the participant P' simply aborts. Otherwise, the key $k_{\mathsf{PP}}^{(P',\mathsf{gid})}$ is stored locally. The remaining computation of P' remains the same.

ZoomPAKE Key Rotation Join Phase: The key rotation phase is very similar to the original one. The only difference from the leader P's side is that P has to encrypt the ZBox ciphertext using $\mathsf{AEAD_{PP}}$ for every participant $\overline{P}$ in his local party set, i.e., $\mathsf{id}_{\overline{P}} \in \pi_P^{\mathsf{gid}}.GP$ and $\mathsf{id}_{\overline{P}} \neq \mathsf{id}_P$, using the stored key $k_{\mathsf{PP}}^{(\overline{P},\mathsf{gid})}$, output by $\mathsf{PAKE_{PP}}$, a independently random nonce $\mathsf{nonce}_{\mathsf{PP}}^{\overline{P}}$, the header consisting of the sign-up messages of P and $\overline{P}$. The output is a ciphertext bundle that includes $\mathsf{AEAD_{PP}}$ ciphertexts rather than the original ZBox ciphertexts.

When receiving the $\mathsf{AEAD_{PP}}$ ciphertext, a participant P' first decrypts it by using the stored key $k^{(P',\mathsf{gid})}$ for a ZBox ciphertext $c_{P'}^{\mathsf{gid}}$. Then, P' simply runs the original $\mathsf{KeyRotat\text{-}P}$ algorithm using the new ciphertext $c_{P'}^{\mathsf{gid}}$.

Instantiation Suggestions. We suggest to instantiate the underlying $\mathsf{PAKE_{PP}}$ with CPace [101] or SPAKE2 [9] for the $\mathsf{w\text{-}PAKE}$ security, see Section 6.3.2. The $\mathsf{AEAD_{PP}}$ can be instantiated with CAU-C4 or CAU-SIV-C4 [28] for the d-FROB security, see Section 2.2.4.

6.7 Comparison with Concurrent Work [83]

The concurrent work [83] and this paper both analyze the security of the end-to-end secure Zoom library while having different focuses. The analysis in this paper and [83] mainly has differences from three aspects.

1. **Different Protocol Abstraction**: [83] abstracts the Zoom library as a novel *leader-based continuous group key agreement with liveness* (LL-CGKA) scheme, that takes the

full "leader-participant-list" mechanism, the "heartbeat" mechanism, and liveness into account. By this, their analysis additionally captures that all members in a group agree on a group roster, i.e., a list including the identifiers of all group members. Moreover, their Zoom protocol considers the change of group leader in Zoom.

However, their Zoom protocol assumes a PKI that "provides to each long-term identity id their respective private signing key *isk*", while our protocol lets every party generate their own identity key pair, which is closer to the white-paper [53, Section 7.6]. Besides, their Zoom protocol omits the group secret generation and distribution, which is mentioned in [53, Section 2, Section 7.11], and therefore does not formally distinguish the roles "participants", i.e., the parties that are authorized for a group, from the "insiders", i.e., who develop and maintain Zoom's server infrastructure and its cloud providers. Our protocol captures the "group secret" mechanism. Moreover, their Zoom protocol does not include an independent key rotation phase but embed it into the member leaving phase. However, the member leaving in Zoom does not always trigger the key rotation, e.g., within 15 seconds of the previous member leaving [53, Section 7.6.6]. Our protocol has an independent Key Rotation phase to capture the group key update.

2. **Different Security Models**: The model in [83] and our Sec-mGKD-pki model respectively capture the characteristic features of their LL-CGKA scheme and our mGKD protocol. Apart from this, there are still following differences:

   On the one hand, their model considers the change of group leader and rejoining a group after leaving. Instead, our model only considers the unique leader of a group and prevent a party from rejoining a group after leaving.

   On the other hand, their model considers a globally trusted PKI that honestly distributes the identity public keys for all parties in the world. However, our model only assumes the honest sign-up messages distribution for any target group that will lead the attacker to win. Moreover, the attacker in their model has to corrupt each party's long-term private key and all alive per-group states at the same time. Our model partitions the oracle for state leakage into two: the corruption oracle that returns the long-term state of a party, and the compromise oracle that returns a per-group state of a party. By this, our model captures the fact that the leakage of a party's per-group state in one group does not influence the security of another alive group.

3. **Different Protocol Optimization**: [83] improves the security of Zoom library by importing a new "period" term. The leader in their optimization can opt not to sample a new group key but simply ask every participant derives the next group key from their local state. This improves the efficiency while preserving forward secrecy.

   Our optimization however has a totally different focus. Our PP transformation aims to provide the security against "insiders", i.e., the security for authorized parties holds even though no trusted PKI exists.

6.8 Technical Summary

In this paper, we propose a new mGKD protocol that captures the behaviors of the Zoom library and three associated security models: the basic Sec-mGKD-pki model considers restricted end-to-end encrypted security assuming the existence of a trusted PKI; the Sec-mGKD-pw model captures full end-to-end encrypted security without any trusted PKI; and the Sec-mGKD-pw+ that combines the Sec-mGKD-pki and Sec-mGKD-pw models. We proved that the Zoom library version 4.0 satisfies the basic Sec-mGKD-pki, but does not provide end-to-end Sec-mGKD-pw security.

To improve the Sec-mGKD-pki security of any mGKD protocol (including the Zoom library) to the Sec-mGKD-pw+ security, we propose a novel PP transformation that makes use of the group secret transmitted over out-of-band channels and cryptographic primitives PAKE and AEAD. Intuitively, to get the group keys that encrypts the real messages in the group chat, every participant must first additionally execute PAKE with the group leader for a shared key. This shared key is peer-wise independent: the group leader knows all shared keys and the participant only knows the one that it produces. Whenever the leader needs to rotate and distribute a new group key, the leader must additionally "wrap" the original ciphertext, i.e., encrypt it using the shared keys and AEAD, and every participant needs to first unwrap the original ciphertext in order to recover the real group keys. In particular, the application of our PP transform to the Zoom library is very efficient in terms of the communication rounds, as it does not cause any additional round trip time.

6.9 Full Proofs

6.9.1 Proof of Theorem 25

Proof. We give the proof of Zoom's security in Sec-mGKD-pki as a sequence of games. Let $\mathsf{Adv}_i(\mathcal{A})$ denote the advantage of an attacker $\mathcal{A}$ in winning **Game** i.

Game 0: This game is identical to the original Sec-mGKD-pki experiment. Thus, we have that

$$\mathsf{Adv}_0(\mathcal{A}) = \mathsf{Adv}_{\Pi}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}(\mathcal{A})$$

Game 1: This game is identical to **Game** 0, except that the challenger $\mathcal{C}$ let the attacker $\mathcal{A}$ immediately win if there exists collision on function H_1. That is, there exists two distinct inputs m_1 and m_2 such that $\mathsf{H}_1(m_1) = \mathsf{H}_1(m_2)$. By this, we ensure that there exists no collision on the function H_1 in the following games. Due to the collision resistance of H_1, we can easily have that:

$$\mathsf{Adv}_0(\mathcal{A}) \leq \mathsf{Adv}_1(\mathcal{A}) + \epsilon_{\mathsf{H}_1}^{\mathsf{coll\text{-}res}}$$

Game 2: This game is identical to the **Game** 1 except the following modification:

- The challenger $\mathcal{C}$ aborts the game if $\mathcal{A}$ can trigger the following event:

  Event E_1: There exists any party P^1, any party P^2, and any group identifier gid such that

  - the long-term state st_{P^1} is not corrupted before P^1 and P^2 both joined the group gid,
  - the sign-up messages, i.e., the identity public keys $\mathsf{st}_{P^1}.ipk = ipk_{P^1}$ of P^1 is honestly delivered to P^2 in the Participant Join phase between P^1 and P^2 for the group gid, and
  - P^1 and P^2 have disagreement on the binding information $\mathsf{Binding}_{P^1}^{\mathsf{gid}}$.

By this, we ensure that in the following games, if the sign-up message of a party P^1 is delivered to another party P^2 before the corruption of the long-term state st_{P^1} in any group gid, then P^2 and P^1 must agree on the Binding information $\mathsf{Binding}_{P^1}^{\mathsf{gid}}$.

Obviously, it holds that

$$\mathsf{Adv}_1 \leq \mathsf{Adv}_2(\mathcal{A}) + \Pr[E_1]$$

Below, we analyze the probability that $\mathcal{A}$ can trigger E_1 by reduction. If the attacker $\mathcal{A}$ can trigger the event E_1, then we construct an attacker $\mathcal{B}_1$ that breaks the $\mathsf{euf\text{-}cma}$ security of the underlying DS scheme. The attacker $\mathcal{B}_1$ receives a public verification key $vk^\star$ and honestly initializes **Game** 1. Moreover, $\mathcal{B}_1$ guesses the index $i^\star$of one NewParty query that will create the party P^1 in the event E_1. Note that there are at most NewParty queries in the game. $\mathcal{B}_1$ guesses correctly with probability at least $\frac{1}{q_{\text{NewParty}}}$. Then, $\mathcal{B}_1$ honestly answers $\mathcal{A}$'s queries except the following ones:

- NewParty(id_P): If this is the $i^\star$-th query, then $\mathcal{B}_1$ initializes a state st_P by setting $\mathsf{st}_P.\mathsf{id}_{\leftarrow}\mathsf{id}_P$. Then, $\mathcal{C}$ sets $\mathsf{st}_P.ipk$ to $vk^\star$ that is given by its challenger. Finally, $\mathcal{B}_1$ forwards $m_{\mathsf{SignUp}}^P = vk^\star$ to $\mathcal{A}$ and marks P as "created".

  For other queries to this oracle, $\mathcal{B}_1$ executes them honestly.

- RegisterAuth($\mathsf{id}_P, \mathsf{gid}, m$): If the input party P is created via the $i^\star$-th query to the NewParty oracle, then $\mathcal{B}_1$ honestly executes the checks. If no error occurs, $\mathcal{B}_1$ honestly produces its binding information $\mathsf{Binding}_P^{\widetilde{\mathsf{gid}}}$ and send $\mathsf{H}_1(\mathsf{ctxt}_1) \parallel \mathsf{H}_1(\mathsf{Binding}_P^{\widetilde{\mathsf{gid}}})$ to its DS signing oracle. Then, $\mathcal{B}_1$ receives a signature $\sigma_P^{\widetilde{\mathsf{gid}}}$ and use it as the output of the ZSign signature. The rest of this query is honestly executed.

  For other queries to this oracle, $\mathcal{B}_1$ executes them honestly.

- RegisterInject($\mathsf{id}_P, \mathsf{gid}, gs, m$): If the input party P is created via the $i^\star$-th query to the NewParty oracle and $\mathsf{gid} = \widetilde{\mathsf{gid}}$, then $\mathcal{B}_1$ honestly executes the checks. If no error occurs, $\mathcal{B}_1$ honestly produces its binding information $\mathsf{Binding}_P^{\widetilde{\mathsf{gid}}}$ and send $\mathsf{H}_1(\mathsf{ctxt}_1) \parallel \mathsf{H}_1(\mathsf{Binding}_P^{\widetilde{\mathsf{gid}}})$ to its DS signing oracle. Then, $\mathcal{B}_1$ receives a signature $\sigma_P^{\widetilde{\mathsf{gid}}}$ and use it as the output of the ZSign signature. The rest of this query is honestly executed.

For other queries to this oracle, $\mathcal{B}_1$ executes them honestly.

- $\textsc{SendJoinAuth}(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, m)$: If the input party P' is created via the $i^\star$-th query to the $\textsc{NewParty}$ oracle, then $\mathcal{B}_1$ extracts the binding information $\widetilde{\mathsf{Binding}_{P'}^{\mathsf{gid}}}$ and a signature $\widetilde{\sigma_{P'}^{\mathsf{gid}}}$. If $\widetilde{\sigma_{P'}^{\mathsf{gid}}}$ is not output by any $\textsc{RegisterAuth}$ or $\textsc{RegisterInject}$ oracle for the binding information $\widetilde{\mathsf{Binding}_{P'}^{\mathsf{gid}}}$ but the verification $\mathsf{DS.Vrfy}(vk^\star, \mathsf{H}_1(\mathsf{ctxt}_1) \parallel \mathsf{H}_1(\widetilde{\mathsf{Binding}_{P'}^{\mathsf{gid}}}), \widetilde{\sigma_{P'}^{\mathsf{gid}}}) = \mathsf{true}$ passes, then $\mathcal{B}_1$ immediately returns $(\mathsf{H}_1(\mathsf{ctxt}_1) \parallel \mathsf{H}_1(\widetilde{\mathsf{Binding}_{P'}^{\mathsf{gid}}}), \widetilde{\sigma_{P'}^{\mathsf{gid}}})$ to its challenger and aborts the experiment.

  In all other cases inside this query or for other queries to this oracle, $\mathcal{B}_1$ executes them honestly.

- $\textsc{SendJoinInject}(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m)$: If the input party P is created via the $i^\star$-th query to the $\textsc{NewParty}$ oracle, then $\mathcal{B}_1$ extracts the binding information $\widetilde{\mathsf{Binding}_{P'}^{\mathsf{gid}}}$ and a signature $\widetilde{\sigma_{P'}^{\mathsf{gid}}}$. If $\widetilde{\sigma_{P'}^{\mathsf{gid}}}$ is not output by any $\textsc{RegisterAuth}$ or $\textsc{RegisterInject}$ oracle for the binding information $\widetilde{\mathsf{Binding}_{P'}^{\mathsf{gid}}}$ but the verification $\mathsf{DS.Vrfy}(vk^\star, \mathsf{H}_1(\mathsf{ctxt}_1) \parallel \mathsf{H}_1(\widetilde{\mathsf{Binding}_{P'}^{\mathsf{gid}}}), \widetilde{\sigma_{P'}^{\mathsf{gid}}}) = \mathsf{true}$ passes, then $\mathcal{B}_1$ immediately returns $(\mathsf{H}_1(\mathsf{ctxt}_1) \parallel \mathsf{H}_1(\widetilde{\mathsf{Binding}_{P'}^{\mathsf{gid}}}), \widetilde{\sigma_{P'}^{\mathsf{gid}}})$ to its challenger and aborts the experiment.

  In all other cases inside this query or for other queries to this oracle, $\mathcal{B}_1$ executes them honestly.

- $\textsc{Corrupt}(\mathsf{id}_P)$: If the input party P is created via the $i^\star$-th query to the $\textsc{NewParty}$ oracle, $\mathcal{B}_1$ aborts. Otherwise, $\mathcal{B}_1$ honestly executes this oracle.

If the attacker $\mathcal{A}$ can trigger the event E_1 and $\mathcal{B}_1$ guesses the oracle that creates party P^1 correctly, then $\mathcal{A}$ must trigger the event E_1 before querying the $\textsc{Corrupt}$ that causes the abortion. Moreover, there must also exist a group identifier gid and a party P^2 such that

1. P^2 receives the honest sign-up message $\mathsf{st}_{P^1}.ipk = ipk_{P^1}$ of the the party P^1 in the group gid,
2. the long-term state st_{P^1} is not corrupted before P^1 and P^2 joined the group gid, and
3. the parties P^1 and P^2 have disagreement on P^1's binding information $\mathsf{Binding}_{P^1}^{\mathsf{gid}}$.

This means, $\mathcal{B}_1$ can always win in the $\textsc{SendJoinAuth}$ or $\textsc{SendJoinInject}$ oracle.

Note also that the event "the attacker $\mathcal{A}$ can trigger event E_1" and the event "$\mathcal{B}_1$ guesses correctly" are independent. Thus

$$\begin{aligned}\epsilon_{\mathsf{DS}}^{\mathsf{euf\text{-}cma}} &\geq \Pr[\mathcal{B}_1 \text{ wins}] \\ &\geq \Pr[\mathcal{A} \text{ can trigger event } E_1 \textbf{ and } \mathcal{B}_1 \text{ guesses correctly}] \\ &\geq \Pr[\mathcal{A} \text{ can trigger event }] \cdot \Pr[\mathcal{B}_1 \text{ guesses correctly}] \\ &\geq \Pr[E_1] \cdot \frac{1}{q_{\textsc{NewParty}}}\end{aligned}$$

The above equation can be rewritten as:

$$\Pr[E_1] \leq q_{\textsc{NewParty}} \epsilon_{\mathsf{DS}}^{\mathsf{euf\text{-}cma}}$$

Thus, we have that

$$\mathsf{Adv}_1(\mathcal{A}) \leq \mathsf{Adv}_2(\mathcal{A}) + q_{\textsc{NewParty}} \epsilon_{\mathsf{DS}}^{\mathsf{euf\text{-}cma}}$$

Game 3: This game is identical to **Game** 2 except the at the beginning of the experiment the challenger $\mathcal{C}$ guesses a group identifier $\widetilde{\mathsf{gid}}$, whose associated leader is denoted by $P^{\widetilde{\mathsf{gid}}}$, that will lead $\mathcal{A}$ to win, and lets $\mathcal{A}$ immediately lose if the guess is wrong. By this, we ensure that the attacker $\mathcal{A}$ can win only by triggering one of the events as follows:

- [In the case of event E_{KAuth}] there exists any party P' that is authorized for the group $\widetilde{\mathsf{gid}}$ and any group key index gkid such that $gk_{P'}^{(\widetilde{\mathsf{gid}},\mathsf{gkid})} \neq \perp$ but $gk_{P^{\widetilde{\mathsf{gid}}}}^{(\widetilde{\mathsf{gid}},\mathsf{gkid})} \neq gk_{P'}^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$, without the violation of the freshness condition $\mathsf{frsh}_{\mathsf{KAuth}}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$, or
- [In the case of event E_{KPriv}] $\mathcal{A}$ will query $\textsc{Test}$ oracle with input $(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$ for some party identifier P' and some group key index gkid and correctly guess the challenge bit $\mathsf{b} = \mathsf{b}'$, without violation of the freshness $\mathsf{frsh}_{\mathsf{KPriv}}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$.

Note that each group must be created via $\textsc{NewGroup}$ oracle. There are at most $q_{\textsc{NewGroup}}$ groups in the experiment. Thus, the guess is correct with probability at least $\frac{1}{q_{\textsc{NewGroup}}}$. Note that whether $\mathcal{A}$ wins in **Game** 2 and whether the challenger guesses correctly in **Game** 3 is independent. We have that

$$\mathsf{Adv}_2(\mathcal{A}) \leq q_{\textsc{NewGroup}} \mathsf{Adv}_3(\mathcal{A})$$

Below, we analyze the advantage that $\mathcal{A}$ wins in **Game** 3 by case distinction, i.e., whether $\mathcal{A}$ wins by triggering E_{KAuth} in Case 1, the advantage of which is denoted by Adv_3^{C1}, or by triggering E_{KPriv} in Case 2, the advantage of which is denoted by Adv_3^{C2}. Thus, we have that

$$\mathsf{Adv}_3(\mathcal{A}) := \max\left(\mathsf{Adv}_3^{C1}(\mathcal{A}), \mathsf{Adv}_3^{C2}(\mathcal{A})\right)$$

Case 1: $\mathcal{A}$ wins by triggering event E_{KAuth}.

In this case, due to the winning conditions and the freshness requirement, we know that for the group identifier $\widetilde{\mathsf{gid}}$ with a leader denoted by $P^{\widetilde{\mathsf{gid}}}$, there must exist an authorized party P' and a group key index gkid such that:

1. $gk_{P'}^{(\widetilde{\mathsf{gid}},\mathsf{gkid})} \neq \perp$ and $gk_{P^{\widetilde{\mathsf{gid}}}}^{(\widetilde{\mathsf{gid}},\mathsf{gkid})} \neq gk_{P'}^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$,
2. neither $\pi_{P'}^{\widetilde{\mathsf{gid}}}$ nor $\pi_{P^{\widetilde{\mathsf{gid}}}}^{\widetilde{\mathsf{gid}}}$ is compromised,
3. the long-term states $\mathsf{st}_{P'}$ and $\mathsf{st}_{P^{\widetilde{\mathsf{gid}}}}$ are not corrupted before P' and $P^{\widetilde{\mathsf{gid}}}$ both joined the group $\widetilde{\mathsf{gid}}$, and

4. the sign-up messages, i.e., the identity public keys $\mathsf{st}_{P'}.ipk = ipk_{P'}$ and $\mathsf{st}_{P^{\widetilde{\mathsf{gid}}}}.ipk = ipk_{P^{\widetilde{\mathsf{gid}}}}$, of P' and $P^{\widetilde{\mathsf{gid}}}$ both honestly arrive at the other.

By **Game** 2, it must further hold that P' and $P^{\widetilde{\mathsf{gid}}}$ agree on each other's binding information $\mathsf{Binding}^{\widetilde{\mathsf{gid}}}_{P'}$ and $\mathsf{Binding}^{\widetilde{\mathsf{gid}}}_{P^{\widetilde{\mathsf{gid}}}}$, which include:

- the group identifier $\widetilde{\mathsf{gid}}$,
- the server-controlled randomness mUUID,
- both parties' identifier $(\mathsf{uid}_{P^{\widetilde{\mathsf{gid}}}}, \mathsf{hid}_{P^{\widetilde{\mathsf{gid}}}})$ and $(\mathsf{uid}_{P'}, \mathsf{hid}_{P'})$,
- both parties' identity keys $ipk_{P^{\widetilde{\mathsf{gid}}}}$ and $ipk_{P'}$, and
- both parties' per-group public key $pk^{\widetilde{\mathsf{gid}}}_{P^{\widetilde{\mathsf{gid}}}}$ and $pk^{\widetilde{\mathsf{gid}}}_{P'}$.

Game $C1.4$: This game is identical to **Game** 3 except the following modification:

- The challenger guesses the index of query to the RegisterAuth or RegisterInject oracle, which creates the per-group state $\pi^{\widetilde{\mathsf{gid}}}_{P'}$ in the winning event E_{KAuth}, at the beginning of the experiment and aborts the game if the guess is wrong.

Note that there are at most c_{maxReg} register queries to the group $\widetilde{\mathsf{gid}}$ via the RegisterAuth or RegisterInject oracles in the game. The probability that $\mathcal{C}$ guesses correctly is at least $\frac{1}{c_{\mathsf{maxReg}}}$. Thus, we have that

$$\mathsf{Adv}^{C1}_{3}(\mathcal{A}) \leq c_{\mathsf{maxReg}} \mathsf{Adv}^{C1}_{4}(\mathcal{A})$$

Note that the challenger $\mathcal{C}$ will know the identifier of the party P' in the winning event E_{KAuth} at the time of receiving the guessed query to RegisterAuth or RegisterInject oracle. In the following games, we denote the party P' in the winning event E_{KAuth} with $\widetilde{P}$.

Game $C1.5$: This game is identical to the **Game** $C1.4$ except the following modifications:

- At the beginning of the experiment, the challenger $\mathcal{C}$ samples a random $\widetilde{K}$ of bit length l_{H_2}.
- When the leader $P^{\widetilde{\mathsf{gid}}}$ of the the group $\widetilde{\mathsf{gid}}$ needs to compute the output K of the Hash function H_2 over a Diffie-Hellman exchange key K', which is computed by the leader's identity private key $\pi^{\widetilde{\mathsf{gid}}}_{P^{\widetilde{\mathsf{gid}}}}.sk$ and the party $\widetilde{P}$'s public key $pk^{\widetilde{\mathsf{gid}}}_{\widetilde{P}}$, and a constant $\mathsf{ctxt}_{\mathsf{H}_2}$ in Line 5 in Figure 6.2 during the Participant Join phase and the Key Rotation phase of $\widetilde{P}$, $\mathcal{C}$ replaces K with $\widetilde{K}$.
- When $\widetilde{P}$ needs to compute the output K of the Hash function H_2 over a Diffie-Hellman exchange key K', which is computed by the party $\widetilde{P}$'s identity private key $\pi^{\widetilde{\mathsf{gid}}}_{\widetilde{P}}.sk$ and the leader $P^{\widetilde{\mathsf{gid}}}$'s public key $pk^{\widetilde{\mathsf{gid}}}_{P^{\widetilde{\mathsf{gid}}}}$, and a constant $\mathsf{ctxt}_{\mathsf{H}_2}$ in Line 12 in Figure 6.2 during the Participant Join phase and the Key Rotation phase with the leader of the group $\widetilde{\mathsf{gid}}$, $\mathcal{C}$ replaces K with $\widetilde{K}$.

We analyze the gap between **Game** $C1.4$ and **Game** $C1.5$ by reduction to the hardness of $\mathsf{mn\text{-}prf\text{-}ODH}$ problem over ECDH and H_2. If the attacker $\mathcal{A}$ can distinguish **Game** $C1.4$ and **Game** $C1.5$, then we can construct another attacker $\mathcal{B}_2$ that breaks $\mathsf{mn\text{-}prf\text{-}ODH}$ assumption over ECDH and H_2.

The attacker $\mathcal{B}_2$ receives the ECDH parameters and $U = g^u$ for some unknown u. Next, $\mathcal{B}_2$ immediately issues a challenger query $x^\star = \mathsf{ctxt}_{\mathsf{H}_2}$ to its challenger and receives a tuple $(V = g^v, y^\star)$ for some unknown v. Then $\mathcal{B}_2$ invokes $\mathcal{A}$ and simulates **Game** $C1.4$ honestly, except for answering the queries to the following oracles.

- $\textsc{RegisterAuth}(\mathsf{id}_P, \mathsf{gid}, m)$: If the input party P is the leader of the group $\widetilde{\mathsf{gid}}$, i.e., $P = P^{\widetilde{\mathsf{gid}}}$, and the group identifier $\mathsf{gid} = \widetilde{\mathsf{gid}}$, then $\mathcal{B}_2$ does not sample the ZBox key pair uniformly at random. Instead, $\mathcal{B}_2$ replaces $\pi^{\widetilde{\mathsf{gid}}}_{P^{\widetilde{\mathsf{gid}}}}.pk$ with $U = g^u$ that is given by challenger. The rest of this query is executed honestly.

  If the input party P is the participant that is guessed in **Game** $C1.4$, i.e., $P = \widetilde{P}$, and the group identifier $\mathsf{gid} = \widetilde{\mathsf{gid}}$, then $\mathcal{B}_2$ does not sample the ZBox key pair uniformly at random. Instead, $\mathcal{B}_2$ replaces $\pi^{\widetilde{\mathsf{gid}}}_{\widetilde{P}}.pk$ with $V = g^v$ that is given by challenger. The rest of this query is executed honestly.

  For the other queries to this oracle, $\mathcal{B}_2$ executes them honestly.

- $\textsc{RegisterInject}(\mathsf{id}_P, \mathsf{gid}, gs, m)$: If the input party P is the participant that is guessed in **Game** $C1.4$, i.e., $P = \widetilde{P}$, and the group identifier $\mathsf{gid} = \widetilde{\mathsf{gid}}$, then $\mathcal{B}_2$ does not sample the ZBox key pair uniformly at random. Instead, $\mathcal{B}_2$ replaces $\pi^{\widetilde{\mathsf{gid}}}_{\widetilde{P}}.pk$ with $V = g^v$ that is given by challenger. The rest of this query is executed honestly.

  For the other queries to this oracle, $\mathcal{B}_2$ executes them honestly.

- $\textsc{SendJoinAuth}(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, m)$: If the input party $P = P^{\widetilde{\mathsf{gid}}}$, the group identifier $\mathsf{gid} = \widetilde{\mathsf{gid}}$, and ZBox public key pk included in the fourth input m equals $V = g^v$ that is given by the challenger, then $\mathcal{B}_2$ does not compute the the Diffie-Hellman exchange in Line 4 and the computation of H_2 in Line 5 in Figure 6.2. Instead, $\mathcal{B}_2$ replaces the output of H_2 with $y^\star$ that is given by the challenger. The rest of this query is executed honestly.

  If the input party $P = P^{\widetilde{\mathsf{gid}}}$, the group identifier $\mathsf{gid} = \widetilde{\mathsf{gid}}$, and ZBox public key pk included in the fourth input m does *not* equal to $V = g^v$ that is given by the challenger, then $\mathcal{B}_2$ does not compute the the Diffie-Hellman exchange in Line 4 and the computation of H_2 in Line 5 in Figure 6.2. Instead, $\mathcal{B}_2$ queries its ODH_u oracle with input $(pk, \mathsf{ctxt}_{\mathsf{H}_2})$ for a reply y, followed by replacing the output of H_2 with the reply y. The rest of this query is executed honestly.

  If the input party $P = \widetilde{P}$, the group identifier $\mathsf{gid} = \widetilde{\mathsf{gid}}$, and ZBox public key pk included in the third input m equals $U = g^u$ that is given by the challenger, then $\mathcal{B}_2$ does not compute the the Diffie-Hellman exchange in Line 11 and the computation of H_2 in Line 12 in Figure 6.2. Instead, $\mathcal{B}_2$ replaces the output of H_2 with $y^\star$ that is given by the challenger. The rest of this query is executed honestly.

For the other queries to this oracle, $\mathcal{B}_2$ executes them honestly.

- $\textsc{SendJoinInject}(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m)$: If the input party $P = \widetilde{P}$, the group identifier $\mathsf{gid} = \widetilde{\mathsf{gid}}$, and ZBox public key pk included in the fifth input m equals $U = g^u$ that is given by the challenger, then $\mathcal{B}_2$ does not compute the the Diffie-Hellman exchange in Line 11 and the computation of H_2 in Line 12 in Figure 6.2. Instead, $\mathcal{B}_2$ replaces the output of H_2 with $y^\star$ that is given by the challenger. The rest of this query is executed honestly.

  For the other queries to this oracle, $\mathcal{B}_2$ executes them honestly.

- $\textsc{SendKeyRotat}(\mathsf{id}_P, \mathsf{gid}, m)$: Note that this oracle requires that the P must have already joined the group gid. In particular, $\mathcal{B}_2$ must have already computed the output of H_2 for every communication between the leader of the group gid and the participants. In this oracle, $\mathcal{B}_2$ does not re-compute the Diffie-Hellman exchange and the computation of H_2. Instead, $\mathcal{B}_2$ simply reuses the corresponding values derived in the $\textsc{SendJoinAuth}$ or $\textsc{SendJoinInject}$ oracles.

  The rest of this oracle is executed honestly.

- $\textsc{Compromise}(\mathsf{id}_P, \mathsf{gid})$: If the first input party $P = P^{\widetilde{\mathsf{gid}}}$ or $P = \widetilde{P}$ and the second input $\mathsf{gid} = \widetilde{\mathsf{gid}}$, then $\mathcal{B}_2$ aborts.

  The rest of this oracle is executed honestly.

Note that if the attacker $\mathcal{A}$ can trigger the winning event E_{KAuth} without violating the freshness condition, then neither $\pi^{\widetilde{\mathsf{gid}}}_{P^{\widetilde{\mathsf{gid}}}}$ nor $\pi^{\widetilde{\mathsf{gid}}}_{\widetilde{P}}$ is allowed to be compromised due to the freshness condition $\mathsf{frsh}_{\mathsf{KAuth}}$. This game abortion in the $\textsc{Compromise}$ oracle will not happen.

If the attacker $\mathcal{A}$ is able to distinguish **Game** $C1.4$ and **Game** $C1.5$, then the attacker $\mathcal{B}_2$ returns to 0 to its challenger if the $\mathcal{A}$ thinks this is **Game** $C1.4$ and 1 to its challenger if the $\mathcal{A}$ thinks this is **Game** $C1.5$.

Note that $\mathcal{B}_2$ perfectly simulates **Game** $C1.4$ if $y^\star = \mathsf{H}_2(g^{uv}, \mathsf{ctxt}_{\mathsf{H}_2})$ and **Game** $C1.5$ if $y^\star$ is sampled uniformly at random. $\mathcal{B}_2$ wins whenever $\mathcal{A}$ can distinguish the games. Thus, we have that

$$\mathsf{Adv}_4^{C1}(\mathcal{A}) \leq \mathsf{Adv}_5^{C1}(\mathcal{A}) + \epsilon^{\mathsf{mn\text{-}prf\text{-}ODH}}_{\mathsf{ECDH},\mathsf{H}_2}$$

Final Analysis for Case 1: Finally, we analyze the advantage that $\mathcal{A}$ can win by triggering E_{KAuth}. This means, there exists any group key index gkid such that

- $gk_{\widetilde{P}}^{(\widetilde{\mathsf{gid}},\mathsf{gkid})} \neq \perp$ and $gk_{P^{\widetilde{\mathsf{gid}}}}^{(\widetilde{\mathsf{gid}},\mathsf{gkid})} \neq gk_{\widetilde{P}}^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$,

- $\widetilde{P}$ and $P^{\widetilde{\mathsf{gid}}}$ agree on each other's binding information $\mathsf{Binding}_{\widetilde{P}}^{\widetilde{\mathsf{gid}}}$ and $\mathsf{Binding}_{P^{\widetilde{\mathsf{gid}}}}^{\widetilde{\mathsf{gid}}}$ in the Participant Join phase, and
- the per-group states $\pi_{\widetilde{P}}^{\widetilde{\mathsf{gid}}}$ and $\pi_{P^{\widetilde{\mathsf{gid}}}}^{\widetilde{\mathsf{gid}}}$ are not compromised.

From **Game** $C1.5$, we know that $\widetilde{P}$ and $P^{\widetilde{\mathsf{gid}}}$ shares the same random key $\widetilde{K}$ for computing AEAD. From $gk_{\widetilde{P}}^{(\widetilde{\mathsf{gid}},\mathsf{gkid})} \neq \perp$, we know that $\widetilde{P}$ must receives a AEAD nonce and ciphertext tuple (nonce, c), which is decrypted to $(gk_{\widetilde{P}}^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}, \mathsf{gkid}) \neq (gk_{P^{\widetilde{\mathsf{gid}}}}^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}, \mathsf{gkid})$ for some gkid. We consider the following four cases:

- Case 1: $gk_{P^{\widetilde{\mathsf{gid}}}}^{(\widetilde{\mathsf{gid}},\mathsf{gkid})} = \perp$. In this case, the leader $P^{\widetilde{\mathsf{gid}}}$ has not generated the gkid-th group key. This mean, the tuple (nonce, c) must be forged by $\mathcal{A}$. Note that the AEAD header h is known by $\mathcal{A}$. If $\mathcal{A}$ can trigger this case, then we can easily construct an attacker $\mathcal{B}_3$ that breaks the $\mathsf{CTI\text{-}CPA}$ security of the underlying AEAD.
- Case 2: $gk_{P^{\widetilde{\mathsf{gid}}}}^{(\widetilde{\mathsf{gid}},\mathsf{gkid})} \neq \perp$ and c is not produced by $P^{\widetilde{\mathsf{gid}}}$. Similar to the above, if $\mathcal{A}$ can trigger this case, then we can easily construct an attacker $\mathcal{B}_3$ that breaks the $\mathsf{CTI\text{-}CPA}$ security of the underlying AEAD.
- Case 3: $gk_{P^{\widetilde{\mathsf{gid}}}}^{(\widetilde{\mathsf{gid}},\mathsf{gkid})} \neq \perp$ and c is produced by $P^{\widetilde{\mathsf{gid}}}$ but the nonce nonce is not produced by $P^{\widetilde{\mathsf{gid}}}$ for this $gk_{P^{\widetilde{\mathsf{gid}}}}^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$. Let $\mathsf{nonce}^{\widetilde{\mathsf{gid}}}$ denote the nonce produced by $P^{\widetilde{\mathsf{gid}}}$ for this $gk_{P^{\widetilde{\mathsf{gid}}}}^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$. If $\mathcal{A}$ can trigger this case, we can easily construct an attacker $\mathcal{B}_3$ that breaks the customized (n,m)-FROB security of AEAD by outputting $\left(c, (\widetilde{K}, \mathsf{nonce}, h), (\widetilde{K}, \mathsf{nonce}^{\widetilde{\mathsf{gid}}}, h)\right)$.
- Case 4: $gk_{P^{\widetilde{\mathsf{gid}}}}^{(\widetilde{\mathsf{gid}},\mathsf{gkid})} \neq \perp$ and (nonce, c) is produced by $P^{\widetilde{\mathsf{gid}}}$ for this $gk_{P^{\widetilde{\mathsf{gid}}}}^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$. This case is impossible due to the perfect correctness.

Merging the cases analysis above, it holds that

$$\mathsf{Adv}_5^{C1}(\mathcal{A}) \leq \max\left(\epsilon_{\mathsf{AEAD}}^{\mathsf{cti\text{-}cpa}}, \epsilon_{\mathsf{AEAD}}^{(n,m)\text{-}\mathsf{frob}}\right) \leq \epsilon_{\mathsf{AEAD}}^{\mathsf{cti\text{-}cpa}} + \epsilon_{\mathsf{AEAD}}^{(n,m)\text{-}\mathsf{frob}}$$

We further have that

$$\mathsf{Adv}_3^{C1} \leq c_{\mathsf{maxReg}}(\epsilon_{\mathsf{ECDH},\mathsf{H}_2}^{\mathsf{mn\text{-}prf\text{-}ODH}} + \epsilon_{\mathsf{AEAD}}^{\mathsf{cti\text{-}cpa}} + \epsilon_{\mathsf{AEAD}}^{(n,m)\text{-}\mathsf{frob}})$$

Case 2: $\mathcal{A}$ wins by triggering event E_{KPriv}.

In this case, due to the winning event E_{KPriv} and the freshness requirement $\mathsf{frsh}_{\mathsf{KPriv}}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}$, it must hold for the guessed the group identifier $\widetilde{\mathsf{gid}}$ with some leader $P^{\widetilde{\mathsf{gid}}}$ and some group key index gkid that

- $\mathsf{b} = \mathsf{b}'$,
- the group key $gk_P^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$ is not leaked for all P such that $\mathsf{id}_P \in GP^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$,
- the short-term state $\pi_P^{\widetilde{\mathsf{gid}}}$ is not compromised for all P such that $\mathsf{id}_P \in GP^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$,

- the long-term state $\mathsf{st}_{P^{\widetilde{\mathsf{gid}}}}$ of the leader $P^{\widetilde{\mathsf{gid}}}$ is not corrupted before all other participants P such that $\mathsf{id}_P \in GP^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$ and $\mathsf{id}_P \neq \mathsf{id}_{P^{\widetilde{\mathsf{gid}}}}$ joined the group $\widetilde{\mathsf{gid}}$,
- the long-term state st_P of all participants P such that $\mathsf{id}_P \in GP^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$ is not corrupted before P joined the group $\widetilde{\mathsf{gid}}$, and
- the sign-up messages of all parties P such that $\mathsf{id}_P \in GP^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$ are honestly distributed inside the group $\widetilde{\mathsf{gid}}$.

Game $C2.4$**:** This game is identical the **Game** 3 except for the following modification:

- At the beginning of the experiment, the challenger $\mathcal{C}$ guesses the number n_{party} of parties in the set $GP^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$, where gkid is the tested group key identifier. The challenger $\mathcal{C}$ aborts if the guess is wrong.

Note that there are at most c_{maxParty} parties in a group simutanously. The probability that $\mathcal{C}$ guesses correctly is bounded by $\frac{1}{c_{\mathsf{maxParty}}}$. Thus, it holds that

$$\mathsf{Adv}_3^{C2}(\mathcal{A}) \leq c_{\mathsf{maxParty}} \mathsf{Adv}_4^{C2}(\mathcal{A})$$

Game $C2.5$**:** This game is identical the **Game** $C2.4$ except for the following modification:

- At the beginning of the experiment, the challenger $\mathcal{C}$ guesses $(n_{\mathsf{party}} - 1)$ indices of the queries RegisterAuth or RegisterInject that create the per-group states $\pi_{P^i}^{\widetilde{\mathsf{gid}}}$ for some parties P^i, where $1 \leq i \leq (n_{\mathsf{party}} - 1)$. The challenger aborts if $\{P^i\}_i$ are not participants in the set $GP^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$, where gkid is the tested group key identifier.

Note that there are $(n_{\mathsf{party}} - 1)$ participants in the set $GP^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$ and each per-group state can be created in at most c_{maxReg} queries to the RegisterAuth or RegisterInject oracles. Thus, the challenger guesses correctly except for the probability $\frac{1}{c_{\mathsf{maxReg}}^{(n_{\mathsf{party}}-1)}}$.

$$\mathsf{Adv}_4^{C2}(\mathcal{A}) \leq c_{\mathsf{maxReg}}^{(n_{\mathsf{party}}-1)} \mathsf{Adv}_5^{C2}(\mathcal{A})$$

Note that the challenger $\mathcal{C}$ will know the identifier of the participants $P \in GP^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$ in the winning event E_{KPriv} at the time of receiving the $(n_{\mathsf{party}} - 1)$ guessed queries to RegisterAuth or RegisterInject oracle. In the following games, we denote the n_{party} participants in the winning event E_{KPriv} with P^1, ..., $P^(n_{\mathsf{party}} - 1)$. This means, $GP^{(\widetilde{\mathsf{gid}},\mathsf{gkid})} = \{\mathsf{id}_{P^i}\}_i \cup \{\mathsf{id}_{P^{\widetilde{\mathsf{gid}}}}\}$, where gkid is the tested group key identifier.

Game $C2.6$**:** This game is identical the **Game** $C2.5$ except for the following modification:

- At the beginning of the experiment, the challenger $\mathcal{C}$ samples $(n_{\mathsf{party}} - 1)$ random string $\widetilde{K}^1$, ..., $\widetilde{K}^{(n_{\mathsf{party}}-1)}$ of bit length l_{H_2}.

- When the leader $P^{\widetilde{\mathsf{gid}}}$ of the the group $\widetilde{\mathsf{gid}}$ needs to compute the output K of the Hash function H_2 over a Diffie-Hellman exchange key K', which is computed by the leader's identity private key $\pi^{\widetilde{\mathsf{gid}}}_{P^{\widetilde{\mathsf{gid}}}}.sk$ and the party P^i's public key $pk^{\widetilde{\mathsf{gid}}}_{P^i}$ for any $1 \leq i \leq (n_{\mathsf{party}}-1)$, and a constant $\mathsf{ctxt}_{\mathsf{H}_2}$ in Line 5 in Figure 6.2 during the Participant Join phase and the Key Rotation phase of $\widetilde{P}$, $\mathcal{C}$ replaces K with $\widetilde{K}^i$.
- When P^i for any $1 \leq i \leq (n_{\mathsf{party}}-1)$ needs to compute the output K of the Hash function H_2 over a Diffie-Hellman exchange key K', which is computed by the party P^i's identity private key $\pi^{\widetilde{\mathsf{gid}}}_{P^i}.sk$ and the leader $P^{\widetilde{\mathsf{gid}}}$'s public key $pk^{\widetilde{\mathsf{gid}}}_{P^{\widetilde{\mathsf{gid}}}}$, and a constant $\mathsf{ctxt}_{\mathsf{H}_2}$ in Line 12 in Figure 6.2 during the Participant Join phase and the Key Rotation phase with the leader of the group $\widetilde{\mathsf{gid}}$, $\mathcal{C}$ replaces K with $\widetilde{K}^i$.

The gap between **Game** $C2.5$ and **Game** $C2.6$ can be given by a sequence of hybrid games.

Hybrid Game 0. This game is identical to **Game** $C2.5$. Thus, we have that

$$\mathsf{Adv}_{\mathsf{hy}.0}(\mathcal{A}) = \mathsf{Adv}^{C2}_5(\mathcal{A})$$

Hybrid Game i, where $1 \leq i \leq (n_{\mathsf{party}}-1)$. This game is identical to **Hybrid Game** $(i-1)$ except the following modification

- At the beginning of the experiment, the challenger $\mathcal{C}$ samples a random string $\widetilde{K}^i$ of bit length l_{H_2}.
- When the leader $P^{\widetilde{\mathsf{gid}}}$ of the the group $\widetilde{\mathsf{gid}}$ needs to compute the output K of the Hash function H_2 over a Diffie-Hellman exchange key K', which is computed by the leader's identity private key $\pi^{\widetilde{\mathsf{gid}}}_{P^{\widetilde{\mathsf{gid}}}}.sk$ and the party P^i's public key $pk^{\widetilde{\mathsf{gid}}}_{P^i}$, and a constant $\mathsf{ctxt}_{\mathsf{H}_2}$ in Line 5 in Figure 6.2 during the Participant Join phase and the Key Rotation phase of $\widetilde{P}$, $\mathcal{C}$ replaces K with $\widetilde{K}^i$.
- When P^i needs to compute the output K of the Hash function H_2 over a Diffie-Hellman exchange key K', which is computed by the party P^i's identity private key $\pi^{\widetilde{\mathsf{gid}}}_{P^i}.sk$ and the leader $P^{\widetilde{\mathsf{gid}}}$'s public key $pk^{\widetilde{\mathsf{gid}}}_{P^{\widetilde{\mathsf{gid}}}}$, and a constant $\mathsf{ctxt}_{\mathsf{H}_2}$ in Line 12 in Figure 6.2 during the Participant Join phase and the Key Rotation phase with the leader of the group $\widetilde{\mathsf{gid}}$, $\mathcal{C}$ replaces K with $\widetilde{K}^i$.

If the attacker $\mathcal{A}$ can distinguish **Hybrid Game** $(i-1)$ and **Hybrid Game** i, then we can easily construct an attacker $\mathcal{B}_4$ that breaks the $\mathsf{mn\text{-}prf\text{-}ODH}$ security of the underlying ECDH and H_2, similar to the reduction in **Game** $C1.5$. Thus, we can easily have that

$$\mathsf{Adv}_{\mathsf{hy}.(i-1)}(\mathcal{A}) = \mathsf{Adv}_{\mathsf{hy}.i}(\mathcal{A}) + \epsilon^{\mathsf{mn\text{-}prf\text{-}ODH}}_{\mathsf{ECDH},\mathsf{H}_2}$$

Hybrid Game $(n_{\mathsf{party}}-1)$. This game is identical to **Game** $C2.6$. Thus, we have that

$$\mathsf{Adv}_{\mathsf{hy}.(n_{\mathsf{party}}-1)}(\mathcal{A}) = \mathsf{Adv}^{C2}_6(\mathcal{A})$$

To sum up, it holds that

$$\mathsf{Adv}_5^{C2}(\mathcal{A}) \leq \mathsf{Adv}_6^{C2}(\mathcal{A}) + (n_{\mathsf{party}} - 1)\epsilon_{\mathsf{ECDH},\mathsf{H}_2}^{\mathsf{mn\text{-}prf\text{-}ODH}}$$

Game $C2.7$**:** This game is identical to **Game** $C2.6$ except the following modification:

- All ciphertexts between the leader $P^{\widetilde{\mathsf{gid}}}$ and P^i, for every $1 \leq i \leq (n_{\mathsf{party}} - 1)$, are replaced by random strings of the same length.

The gap between **Game** $C2.6$ and **Game** $C2.7$ can be given by $(n_{\mathsf{party}} - 1)$ hybrid games, where the i-th hybrid game replaces all ciphertexts between P^{gid} and P^i in the group $\widetilde{\mathsf{gid}}$ encrypted using $\widetilde{K}^i$ for every $1 \leq i \leq (n_{\mathsf{party}} - 1)$. It is easy to know that the gap between every adjacent hybrid games can be reduced to the IND\$-CCA security of the underlying AEAD. Thus, we have that

$$\mathsf{Adv}_6(\mathcal{A}) \leq \mathsf{Adv}_7(\mathcal{A}) + (n_{\mathsf{party}} - 1)\epsilon_{\mathsf{AEAD}}^{\mathsf{ind\$\text{-}cca}}$$

Now, the attacker $\mathcal{A}$ obtains no information about the challenge bit b and can only randomly guess. The probability that $\mathcal{A}$ wins is $\frac{1}{2}$, i.e.,

$$\mathsf{Adv}_7(\mathcal{A}) = 0$$

We further have that

$$\mathsf{Adv}_3^{C2} \leq c_{\mathsf{maxParty}} c_{\mathsf{maxReg}}^{(n_{\mathsf{party}}-1)} (n_{\mathsf{party}} - 1)(\epsilon_{\mathsf{ECDH},\mathsf{H}_2}^{\mathsf{mn\text{-}prf\text{-}ODH}} + \epsilon_{\mathsf{AEAD}}^{\mathsf{ind\$\text{-}cca}})$$

Final Analysis. By merging the statements above, the proof is concluded by,

$$\mathsf{Adv}_{\Pi}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}(\mathcal{A}) \leq \epsilon_{\mathsf{H}_1}^{\mathsf{coll\text{-}res}} + q_{\textsc{NewParty}}\epsilon_{\mathsf{DS}}^{\mathsf{euf\text{-}cma}} + c_{\mathsf{maxReg}} q_{\textsc{NewGroup}} \Big(\epsilon_{\mathsf{AEAD}}^{\mathsf{cti\text{-}cpa}} + \epsilon_{\mathsf{AEAD}}^{(n,m)\text{-}\mathsf{frob}} + c_{\mathsf{maxReg}}^{(n_{\mathsf{party}}-1)} (n_{\mathsf{party}} - 1)(\epsilon_{\mathsf{ECDH},\mathsf{H}_2}^{\mathsf{mn\text{-}prf\text{-}ODH}} + \epsilon_{\mathsf{AEAD}}^{\mathsf{ind\$\text{-}cca}}) \Big)$$

where c_{maxParty} denotes the maximal number of parties per meeting, $l = 192$ denote the length of random nonce in ZBox algorithm, and $q_{\mathcal{O}}$ denote the maximal number of the queries to any oracle $\mathcal{O}$. □

6.9.2 Proof of Theorem 26

Proof. The proof is given by a sequence of games. Let $\mathsf{Adv}_i(\mathcal{A})$ denote the advantage of an attacker $\mathcal{A}$ in winning **Game** i.

Game 0: This game is identical to the original Sec-mGKD-pw experiment. Thus, we have that

$$\mathsf{Adv}_0(\mathcal{A}) = \mathsf{Adv}_{\Pi'}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}(\mathcal{A})$$

Game 1: This game is identical to the **Game** 0 except the following modification:

- The challenger $\mathcal{C}$ guesses an group identifier $\widetilde{\mathsf{gid}}$ with some leader $P^{\widetilde{\mathsf{gid}}}$ that makes $\mathcal{A}$ win, and aborts the game if the guess is wrong. Namely,

  1. there exists any party P' that is authorized for the group $\widetilde{\mathsf{gid}}$ and any group key index gkid, such that $gk_{P'}^{(\widetilde{\mathsf{gid}},\mathsf{gkid})} \neq \bot$ but $gk_{P^{\widetilde{\mathsf{gid}}}}^{(\widetilde{\mathsf{gid}},\mathsf{gkid})} \neq gk_{P'}^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$, without violating the freshness condition $\mathsf{frsh}_{\mathsf{KAuth}}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$.
  2. $\mathsf{b} = \mathsf{b}'$ without violating the freshness condition $\mathsf{frsh}_{\mathsf{KPriv}}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$, where P', $\widetilde{\mathsf{gid}}$, and gkid are respectively the tested party, group identifier, and group key index.

Note that each group must be created via the NEWGROUP and that the NEWGROUP can be queried at most NEWGROUP times. The challenger guesses correctly with probability at least $\frac{1}{q_{\text{NewGroup}}}$. Thus, we have that

$$\mathsf{Adv}_0 \leq q_{\text{NewGroup}} \mathsf{Adv}_1(\mathcal{A})$$

Note that the freshness conditions $\mathsf{frsh}_{\mathsf{KAuth}}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$ and $\mathsf{frsh}_{\mathsf{KPriv}}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$ both require that the group $\widetilde{\mathsf{gid}}$ is not revealed. In particular, this means that if the challenger guesses correctly, then the attacker $\mathcal{A}$ cannot query the REVEAL oracle with input $\widetilde{\mathsf{gid}}$.

Game 2: This game is identical to **Game** 1 except the following modifications:

- Whenever the attacker $\mathcal{A}$ sends queries to the SENDJOINAUTH$(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, m)$ oracle for some parties P and P' and group $\mathsf{gid} = \widetilde{\mathsf{gid}}$ and the party P is expected to derive a key k_{PP} of the PAKE, the challenger does the following:

  – If there exists no authorized party $P'' \neq P$ in the group $\widetilde{\mathsf{gid}}$ such that P and P'' have the same transcript for the PAKE execution, then the challenger samples the key k_{PP} for the conversation between P and P' uniformly at random instead of computing it from $\mathsf{PAKE}_{\mathsf{PP}}$.

  – If there exists any other authorized party $P'' \neq P$ in the group $\widetilde{\mathsf{gid}}$ such that P and P'' have the same transcript for the PAKE execution, then P'' must have already sampled the key k_{PP} (for a conversation with some party P'''). In this case, the challenger replaces the key k_{PP} of P with the one of P''.

We analyze the gap between **Game** 1 and **Game** 2 by reduction to the $\mathsf{w\text{-}PAKE}$ security of the PAKE scheme $\mathsf{PAKE}_{\mathsf{PP}}$. Namely, if the attacker $\mathcal{A}$ can distinguish **Game** 1 and **Game** 2, then we can construct an attacker $\mathcal{B}$ that breaks the $\mathsf{w\text{-}PAKE}$ security of the PAKE scheme $\mathsf{PAKE}_{\mathsf{PP}}$. The attacker $\mathcal{B}_1$ simulates **Game** 1 honestly except for answering the following oracles:

- $\textsc{NewGroup}(\mathsf{id}_P, \mathsf{gid})$ if $\mathsf{gid} = \widetilde{\mathsf{gid}}$: $\mathcal{B}_1$ samples gs_Π from the distribution $\mathcal{D}_\Pi$ and runs the $m^{\mathsf{gid}}_{\mathsf{GSch}} \xleftarrow{\$} \mathsf{Schedule}(P, \widetilde{\mathsf{gid}}, gs^{\widetilde{\mathsf{gid}}}_\Pi)$ of Π rather than $\mathsf{Schedule}'$ of Π' for an associated outgoing message $m^{\widetilde{\mathsf{gid}}}_{\mathsf{GSch}}$. Then, $\mathcal{B}_1$ marks the group gid as "created" and "valid" and marks P as the leader of the group gid and "authorized". Finally, $\mathcal{B}_1$ returns $m^{\mathsf{gid}}_{\mathsf{GSch}}$ to $\mathcal{A}$.
- $\textsc{RegisterAuth}(\mathsf{id}_P, \mathsf{gid}, m)$ if $\mathsf{gid} = \widetilde{\mathsf{gid}}$ and $\mathsf{id}_P \neq \mathsf{id}_{P_{\widetilde{\mathsf{gid}}}}$: $\mathcal{B}_1$ aborts if this oracle has been queried on the same tuple $(\mathsf{id}_P, \widetilde{\mathsf{gid}})$, or $\widetilde{\mathsf{gid}}$ is not marked as created and valid, or the party P is not authorized for the group $\widetilde{\mathsf{gid}}$. Otherwise, $\mathcal{B}_1$ first simply runs $m' \xleftarrow{\$} \mathsf{Register\text{-}P}(P, \mathsf{gid}, gs^{\widetilde{\mathsf{gid}}}_\Pi, m)$, where $gs^{\widetilde{\mathsf{gid}}}_\Pi$ is the group secret of protocol Π. Then, $\mathcal{B}_1$ sends queries $\textsc{SendPAKE}(U^{(\mathsf{id}_P, \widetilde{\mathsf{gid}})}, \epsilon)$ to its challenger and receives a reply $c^{(P, \widetilde{\mathsf{gid}})}_{\mathsf{PP}}$. Finally, $\mathcal{B}_1$ forwards the tuple $(m', c^{(P, \widetilde{\mathsf{gid}})}_{\mathsf{PP}})$ to $\mathcal{A}$.
- $\textsc{SendJoinAuth}(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, m)$ if $\mathsf{gid} = \widetilde{\mathsf{gid}}$: $\mathcal{B}_1$ first checks
  - whether $\widetilde{\mathsf{gid}}$ is marked as created and valid,
  - whether P is authorized for this group $\widetilde{\mathsf{gid}}$,
  - whether both parties P and P' have been created and registered for this group,
  - whether either P or P' is the leader of the group $\widetilde{\mathsf{gid}}$,
  - whether the leader of the group, either P or P', has joined the group, and that the other party hasn't joined the group yet.

  If any of the check fails, $\mathcal{B}_1$ directly returns $\perp$ to $\mathcal{A}$. Otherwise, $\mathcal{B}_1$ behaviors differently depending on whether the P in the group gid has produced a key k_{PP} of $\mathsf{PAKE}_{\mathsf{PP}}$ during the communication with P'.

  - If P has not produced the $\mathsf{PAKE}_{\mathsf{PP}}$ key in the communication with P' in the group $\widetilde{\mathsf{gid}}$, then $\mathcal{B}_1$ first extracts a valid input message m_1 for running $\mathsf{PAKE}_{\mathsf{PP}}$ from the input message m. Then, $\mathcal{B}_1$ sends queries $\textsc{SendPAKE}(U^{(\mathsf{id}_P, \widetilde{\mathsf{gid}})}, m_1)$ to its challenger and receives a reply c_{PP}. Finally, $\mathcal{B}_1$ checks whether $\mathsf{PAKE}_{\mathsf{PP}}$ is expected to output a key. If so, $\mathcal{B}_1$ further queries $\textsc{TestPAKE}(U^{(\mathsf{id}_P, \widetilde{\mathsf{gid}})})$ to its challenger and uses the reply k_{PP} as the output key of $\mathsf{PAKE}_{\mathsf{PP}}$.
  - If the key k_{PP} of $\mathsf{PAKE}_{\mathsf{PP}}$ is produced, then $\mathcal{B}_1$ first extracts an input message m_2 for running $\mathsf{Join\text{-}L}$ or $\mathsf{Join\text{-}P}$ algorithm of Π (depending whether P is the leader or a participant of the group $\widetilde{\mathsf{gid}}$) from the input message m. Next, $\mathcal{B}_1$ runs $c \xleftarrow{\$} \mathsf{Join\text{-}L}(P, \mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, gs^{\mathsf{gid}}_\Pi, m_2)$ if P is the leader of the group $\widetilde{\mathsf{gid}}$ or $c \xleftarrow{\$} \mathsf{Join\text{-}P}(P, \mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, gs^{\mathsf{gid}}_\Pi, m_2)$ if P is a participant, where $gs^{\widetilde{\mathsf{gid}}}_\Pi$ is the group secret of the protocol Π. Then, $\mathcal{B}_1$ encrypts c using $\mathsf{AEAD}_{\mathsf{PP}}$ under the key k_{PP}, a random nonce $\mathsf{nonce}_{\mathsf{PP}}$, an header consisting the sign-up messages of both P and P' for a ciphertext c'.

  Finally, $\mathcal{B}_1$ returns c_{PP} and c' that are computed from above steps.
- $\textsc{Reveal}(\mathsf{gid})$ if $\mathsf{gid} = \widetilde{\mathsf{gid}}$: $\mathcal{B}_1$ simply aborts the game.

Note that if $\mathcal{A}$ wins, then $\mathcal{A}$ cannot violate the freshness condition and therefore never queries the Reveal oracle upon $\widetilde{\mathsf{gid}}$. Thus, the game abortion never happens. $\mathcal{B}_1$ perfectly simulates **Game** 1 if the challenge bit of the $\mathsf{w\text{-}PAKE}$ game is 0 and **Game** 2 if the challenge bit of the $\mathsf{w\text{-}PAKE}$ game is 1. If $\mathcal{A}$ can distinguish **Game** 1 and **Game** 2, then $\mathcal{B}$ can also distinguish the challenge bit of the $\mathsf{w\text{-}PAKE}$ security game. Thus, we have that

$$\mathsf{Adv}_1(\mathcal{A}) \leq \mathsf{Adv}_2(\mathcal{A}) + \epsilon^{\mathsf{w\text{-}PAKE}}_{\mathsf{PAKE}_{\mathsf{PP}},\mathcal{D}_{pw}}$$

Game 3: This game is identical to **Game** 2 except the following modifications:

- The challenger $\mathcal{C}$ aborts the game if there exists a participant P' such that
  - P' is authorized for the group $\widetilde{\mathsf{gid}}$ and successfully completed the $\mathsf{PAKE}_{\mathsf{PP}}$ execution when joining the group $\widetilde{\mathsf{gid}}$,
  - P' successfully decrypts an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext that is not output by $P^{\widetilde{\mathsf{gid}}}$ during the Participant Join phase or Key Rotation phase, and
  - the per-group state of P' and $P^{\widetilde{\mathsf{gid}}}$ in the group $\widetilde{\mathsf{gid}}$ are not compromised.

Recall that the key k_{PP} of the authorized party P' for the group $\widetilde{\mathsf{gid}}$ is sampled random uniformly at random. If P' can successfully decrypt an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext that is not output by $P^{\widetilde{\mathsf{gid}}}$ during the Participant Join phase or Key Rotation phase, then this means $\mathcal{A}$ can forge an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext for the key k_{PP} of P' and $P^{\widetilde{\mathsf{gid}}}$ and further breaks the $\mathsf{CTI\text{-}CPA}$ security of the underlying $\mathsf{AEAD}_{\mathsf{PP}}$ scheme. Thus, we can easily construct another attacker $\mathcal{B}_2$ that breaks the $\mathsf{CTI\text{-}CPA}$ security of $\mathsf{AEAD}_{\mathsf{PP}}$ by invoking $\mathcal{A}$. Note that there are at most c_{maxReg} participants in the group $\widetilde{\mathsf{gid}}$. The reduction $\mathcal{B}_2$ can simply guesses the index of the register request of P', which is correct with probability at least $\frac{1}{c_{\mathsf{maxReg}}}$, and honestly simulates **Game** 2 to $\mathcal{A}$. Note that $\mathcal{A}$ cannot query Compromise oracle upon $(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}})$ or $(\mathsf{id}_{P^{\widetilde{\mathsf{gid}}}}, \widetilde{\mathsf{gid}})$. $\mathcal{B}_2$ can perfectly simulates **Game** 2 to $\mathcal{A}$ and win whenever $\mathcal{A}$ can make the forgery. Thus, it holds that

$$\mathsf{Adv}_2(\mathcal{A}) \leq \mathsf{Adv}_3(\mathcal{A}) + c_{\mathsf{maxReg}}\epsilon^{\mathsf{cti\text{-}cpa}}_{\mathsf{AEAD}_{\mathsf{PP}}}$$

Game 4: This game is identical to **Game** 3 except the following modifications:

- The challenger $\mathcal{C}$ aborts the game if there exists a participant P' such that:
  - P' is authorized for the group $\widetilde{\mathsf{gid}}$ and successfully completed the Participant Join phase in the group $\widetilde{\mathsf{gid}}$,
  - P' and $P^{\widetilde{\mathsf{gid}}}$ have disagreement on sign-up messages of P' and $P^{\widetilde{\mathsf{gid}}}$, and
  - the per-group state of P' and $P^{\widetilde{\mathsf{gid}}}$ in the group $\widetilde{\mathsf{gid}}$ are not compromised.

Recall in **Game** 3 that we ensure that the authorized participant P' must agree on all received $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext with the leader $P^{\widetilde{\mathsf{gid}}}$, in particular, during the Participan Join phase in the group $\widetilde{\mathsf{gid}}$. If P' and $P^{\widetilde{\mathsf{gid}}}$ have disagreement on sign-up messages of P' and $P^{\widetilde{\mathsf{gid}}}$, which are the header for encrypting and decrypting the $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext, then we can easily construct an $\mathcal{B}_2$ that breaks the d-frob security of the underlying $\mathsf{AEAD}_{\mathsf{PP}}$ scheme. Thus, we can easily have that

$$\mathsf{Adv}_3(\mathcal{A}) \leq \mathsf{Adv}_4(\mathcal{A}) + \epsilon^{d\text{-}\mathsf{frob}}_{\mathsf{AEAD}_{\mathsf{PP}}}$$

In particular, this game ensures that for all participant P' that is authorized for the group $\widetilde{\mathsf{gid}}$ and successfully completed the Participant Join phase in the group $\widetilde{\mathsf{gid}}$, if the per-group state of P' and $P^{\widetilde{\mathsf{gid}}}$ in the group $\widetilde{\mathsf{gid}}$ are not compromised, then the participant P' and the leader $P^{\widetilde{\mathsf{gid}}}$ must agree on the each other's sign-up messages. In other words, the sign-up messages $m^{P^{\widetilde{\mathsf{gid}}}}_{\mathsf{SignUp}}$ and $m^{P'}_{\mathsf{SignUp}}$ of $P^{\widetilde{\mathsf{gid}}}$ and P' must be honestly delivered to the other.

Below, we analyze the advantage that $\mathcal{A}$ wins in **Game** 4 by case distinction, i.e., whether $\mathcal{A}$ wins by triggering E_{KAuth} in Case 1, the advantage of which is denoted by Adv^{C1}_4, or by triggering E_{KPriv} in Case 2, the advantage of which is denoted by Adv^{C2}_4. Thus, we have that

$$\mathsf{Adv}_4(\mathcal{A}) := \max\Big(\mathsf{Adv}^{C1}_4(\mathcal{A}), \mathsf{Adv}^{C2}_4(\mathcal{A})\Big)$$

Case 1: $\mathcal{A}$ wins by triggering event E_{KAuth}.

Final Analysis of Case 1. In this case, $\mathcal{A}$ wins by triggering event E_{KAuth}. This means,there exists any party P' that is authorized for the group $\widetilde{\mathsf{gid}}$ and any group key index gkid, such that $gk^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}_{P'} \neq \perp$ but $gk^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}_{P^{\widetilde{\mathsf{gid}}}} \neq gk^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}_{P'}$, without violating the freshness condition $\mathsf{frsh}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\mathsf{KAuth}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$.

Recall that $\mathsf{frsh}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\mathsf{KAuth}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$ holds if and only if

1. the per-group states $\pi^{\widetilde{\mathsf{gid}}}_{P^{\widetilde{\mathsf{gid}}}}$ and $\pi^{\widetilde{\mathsf{gid}}}_{P'}$ are not compromised,
2. the long-term states $\mathsf{st}_{P^{\widetilde{\mathsf{gid}}}}$ and $\mathsf{st}_{P'}$ are not corrupted before $P^{\widetilde{\mathsf{gid}}}$ and P' joined the group $\widetilde{\mathsf{gid}}$, and
3. the group $\widetilde{\mathsf{gid}}$ is not revealed.

Recall also that the freshness condition $\mathsf{frsh}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\mathsf{KAuth}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$ holds if and only if

1. the per-group states $\pi^{\widetilde{\mathsf{gid}}}_{P^{\widetilde{\mathsf{gid}}}}$ and $\pi^{\widetilde{\mathsf{gid}}}_{P'}$ are not compromised,
2. the long-term states $\mathsf{st}_{P^{\widetilde{\mathsf{gid}}}}$ and $\mathsf{st}_{P'}$ are not corrupted before $P^{\widetilde{\mathsf{gid}}}$ and P' joined the group $\widetilde{\mathsf{gid}}$, and

3. the sign-up messages $m_{\mathsf{SignUp}}^{P^{\widetilde{\mathsf{gid}}}}$ and $m_{\mathsf{SignUp}}^{P'}$ of $P^{\widetilde{\mathsf{gid}}}$ and P' are honestly delivered to the other.

In **Game** 4, we ensure that the the sign-up messages $m_{\mathsf{SignUp}}^{P^{\widetilde{\mathsf{gid}}}}$ and $m_{\mathsf{SignUp}}^{P'}$ of $P^{\widetilde{\mathsf{gid}}}$ and P' are honestly delivered to the other if

1. the party P' is authorized for the group $\widetilde{\mathsf{gid}}$ completed the Participant Join phase in the group $\widetilde{\mathsf{gid}}$,
2. the per-group states $\pi_{P^{\widetilde{\mathsf{gid}}}}^{\widetilde{\mathsf{gid}}}$ and $\pi_{P'}^{\widetilde{\mathsf{gid}}}$ are not compromised,

Thus, if the freshness condition $\mathsf{frsh}_{\mathsf{KAuth}}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$ holds, then the freshness condition $\mathsf{frsh}_{\mathsf{KAuth}}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$ must also hold.

Below, we prove that if $\mathcal{A}$ can win via the event E_{KAuth} against Π', then we can construct another attacker $\mathcal{B}_3$ that breaks the $\mathsf{Sec\text{-}mGKD\text{-}pki}$ security of Π via the event E_{KAuth} by invoking $\mathcal{A}$. $\mathcal{B}_3$ answers the queries from $\mathcal{A}$ as follows:

- NewParty(id_P): $\mathcal{B}_3$ simply forwards this query to its challenger and the reply to $\mathcal{A}$.
- NewGroup($\mathsf{id}_P, \mathsf{gid}$): $\mathcal{B}_3$ simply forwards this query to its challenger and the reply to $\mathcal{A}$. Moreover, $\mathcal{B}_3$ samples a random pw^{gid} from the distribution $\mathcal{D}_{pw}$ for the group gid.
- Auth($\mathsf{gid}, \mathsf{id}_P$): $\mathcal{B}_3$ simply forwards this query to its challenger.
- RegisterAuth($\mathsf{id}_P, \mathsf{gid}, m$): $\mathcal{B}_3$ simply forwards this query to its challenger for a reply c. If P is not the leader of the group gid, then $\mathcal{B}_3$ additionally runs the first pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon pw^{gid} and returns c together with the outgoing message of $\mathsf{PAKE}_{\mathsf{PP}}$ to $\mathcal{A}$.
- RegisterInject($\mathsf{id}_P, \mathsf{gid}, gs, m$): $\mathcal{B}_3$ parses gs into two portions (gs_Π, pw). Next, $\mathcal{B}_3$ simply forwards the query RegisterInject($\mathsf{id}_P, \mathsf{gid}, gs_\Pi, m$) to its challenger for a reply c. If P is not the leader of the group gid, then $\mathcal{B}_3$ additionally runs the first pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon pw and returns c together with the outgoing message of $\mathsf{PAKE}_{\mathsf{PP}}$ to $\mathcal{A}$.
- SendJoinAuth($\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, m$): We consider two cases: For the first case that $\mathsf{gid} = \widetilde{\mathsf{gid}}$, we consider the following two steps:
  - If the party P of the group gid has not derived the key k_{PP}, then $\mathcal{B}_3$ runs the next pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon necessary information from the input message m and the password pw^{gid}. If the party P of the group gid now is expected to derive the key k_{PP} of $\mathsf{PAKE}_{\mathsf{PP}}$ and there is no other party P'' in the group $\widetilde{\mathsf{gid}}$ that has the same transcript of $\mathsf{PAKE}_{\mathsf{PP}}$ as P, then $\mathcal{B}_3$ replaces it by a random key of the same length.

    If the party P of the group gid now is expected to derive the key k_{PP} of $\mathsf{PAKE}_{\mathsf{PP}}$ and there is a party P'' in the group $\widetilde{\mathsf{gid}}$ that has the same transcript of $\mathsf{PAKE}_{\mathsf{PP}}$ as P, then $\mathcal{B}_3$ replaces the key of P with the one of P''.

- If the party P of the group gid now has derived the key k_{PP}, then $\mathcal{B}_3$ first checks P agrees on the party P''s sign-up message $m^{P'}_{\mathsf{SignUp}}$. If not, then $\mathcal{B}_3$ simply aborts. Otherwise, $\mathcal{B}_3$ checks whether m should include an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext. If so, $\mathcal{B}_3$ decrypts the $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext using the random key k_{PP} and other necessary information from the input m for a message m_1. Then, $\mathcal{B}_3$ sends the query $\textsc{SendJoinAuth}(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, m_1)$ to its challenger for a reply m_2. After that, if m_2 is not an empty string, then $\mathcal{B}_3$ encrypts m_2 using $\mathsf{AEAD}_{\mathsf{PP}}$ upon the random key k_{PP}, a random nonce, an header consisting the sign-up messages of P and P'.

Finally, $\mathcal{B}_3$ forwards all outgoing messages in this algorithm to $\mathcal{A}$.

For the second case that $\mathsf{gid} \neq \widetilde{\mathsf{gid}}$, we consider the following two steps:

- If the party P of the group gid has not derived the key k_{PP}, then $\mathcal{B}_3$ runs the next pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon necessary information from the input message m and the password pw^{gid}.
- If the party P of the group gid now has derived the key k_{PP}, then $\mathcal{B}_3$ checks whether m should include an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext. If so, $\mathcal{B}_3$ decrypts the $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext using the key k_{PP} and other necessary information from the input m for a message m_1. Then, $\mathcal{B}_3$ sends the query $\textsc{SendJoinAuth}(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, m_1)$ to its challenger for a reply m_2. After that, if m_2 is not an empty string, then $\mathcal{B}_3$ encrypts m_2 using $\mathsf{AEAD}_{\mathsf{PP}}$ upon the key k_{PP}, a random nonce, an header consisting the sign-up messages of P and P'.

Finally, $\mathcal{B}_3$ forwards all outgoing messages in this algorithm to $\mathcal{A}$.

- $\textsc{SendJoinInject}(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m)$: $\mathcal{B}_3$ parses gs into two portion gs_Π and pw. We consider the following two steps:

  - If the party P of the group gid has not derived the key k_{PP}, then $\mathcal{B}_3$ runs the next pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon necessary information from the input message m and the password pw.
  - If the party P of the group gid now has derived the key k_{PP}, then $\mathcal{B}_3$ checks whether m should include an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext. If so, $\mathcal{B}_3$ decrypts the $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext using the key k_{PP} and other necessary information from the input m for a message m_1. Then, $\mathcal{B}_3$ sends the query $\textsc{SendJoinInject}(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, gs_\Pi, m_1)$ to its challenger for a reply m_2. After that, if m_2 is not an empty string, then $\mathcal{B}_3$ encrypts m_2 using $\mathsf{AEAD}_{\mathsf{PP}}$ upon the key k_{PP}, a random nonce, an header consisting the sign-up messages included in m.

  Finally, $\mathcal{B}_3$ forwards all outgoing messages in this algorithm to $\mathcal{A}$.

- $\textsc{SendLeave}(\mathsf{id}_P, \mathsf{gid}, \mathsf{id}_{P'})$: $\mathcal{B}_3$ simply forwards this query to its challenger. If a per-group state π needs to be erased, then $\mathcal{B}_3$ also erases the corresponding $\mathsf{PAKE}_{\mathsf{PP}}$ secrets of π.
- $\textsc{EndGroup}(\mathsf{gid})$: $\mathcal{B}_3$ simply forwards this query to its challenger. If the leader's per-group state π needs to be erased, then $\mathcal{B}_3$ also erases the corresponding $\mathsf{PAKE}_{\mathsf{PP}}$ secrets of π.
- $\textsc{SendKeyRotat}(\mathsf{id}_P, \mathsf{gid}, m)$: We consider two cases:
  - If P is the leader in the group gid, then $\mathcal{B}_3$ first queries $\textsc{SendKeyRotat}(\mathsf{id}_P, \mathsf{gid}, m)$ to its challenger for a reply c. Then, $\mathcal{B}_3$ extracts the portion $c_{\overline{P}}$ in c that is specific to every participant $\overline{P}$ in the group gid, followed by encrypting it using the corresponding $\mathsf{AEAD}_{\mathsf{PP}}$ key k_{PP}, a random nonce, and an header that is same as the one in the corresponding Participant Join phase. Finally, $\mathcal{B}_3$ outputs the $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext and nonce for every participant $\overline{P}$ in the group gid. If any error occurs during the above execution, then $\mathcal{B}_3$ aborts and undoes the above executions.
  - If P is a participant in the group gid, then $\mathcal{B}_3$ first extracts an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext and an $\mathsf{AEAD}_{\mathsf{PP}}$ nonce from the input m. Next, $\mathcal{B}_3$ recovers a message m_1 from the $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext using the corresponding $\mathsf{AEAD}_{\mathsf{PP}}$ key k_{PP}, the $\mathsf{AEAD}_{\mathsf{PP}}$ nonce, and an header that is same as the one in the corresponding Participant Join phase. Then, $\mathcal{B}_3$ extracts other necessary information m_2 from the input m for querying $\textsc{SendKeyRotat}(\mathsf{id}_P, \mathsf{gid}, m_1 \parallel m_2)$ returns the reply m_{KRot} to $\mathcal{A}$. If any error occurs during the above execution, then $\mathcal{B}_3$ aborts and undoes the above executions.
- $\textsc{Corrupt}(\mathsf{id}_P)$: $\mathcal{B}_3$ simply forwards this query to its challenger and the reply to $\mathcal{A}$.
- $\textsc{Compromise}(\mathsf{id}_P, \mathsf{gid})$: $\mathcal{B}_3$ forwards this query to its challenger. If the reply is not $\perp$, then $\mathcal{B}_3$ forwards the reply together with the key k_{PP} of P in the group gid to $\mathcal{A}$. Otherwise, $\mathcal{B}_3$ simply returns $\perp$ to $\mathcal{A}$.
- $\textsc{Leak}(\mathsf{id}_P, \mathsf{gid}, \mathsf{gkid})$: $\mathcal{B}_3$ simply forwards this query to its challenger and the reply to $\mathcal{A}$.
- $\textsc{Reveal}(\mathsf{gid})$: $\mathcal{B}_3$ forwards this query to its challenger. Then, $\mathcal{B}_3$ forwards the reply together with the password pw^{gid} to $\mathcal{A}$.
- $\textsc{Test}(\mathsf{id}_P, \mathsf{gid}, \mathsf{gkid})$: $\mathcal{B}_3$ simply forwards this query to its challenger and the reply to $\mathcal{A}$.

Note that $\mathcal{B}_3$ perfectly simulates **Game** 4 to $\mathcal{A}$. If $\mathcal{A}$ can trigger the event E_{KAuth} against Π', then $\mathcal{B}_3$ can also trigger the event E_{KAuth} against Π. Recall that if the freshness condition $\mathsf{frsh}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\mathsf{KAuth}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$ holds, then the freshness condition $\mathsf{frsh}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\mathsf{KAuth}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$ must also hold. Thus, if $\mathcal{A}$ can win the game against Π' by triggering the event E_{KAuth}, then $\mathcal{B}_3$ can also win the game against Π by triggering E_{KAuth}. We have that

$$\mathsf{Adv}^{C1}_4(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\Pi}(\mathcal{B}_3)$$

Case 2: $\mathcal{A}$ wins by triggering event E_{KPriv}.

Game $C2.5$**:** This game is identical to **Game** 4 except the following modifications:

- Whenever the challenger needs to let the leader $P^{\widetilde{\mathsf{gid}}}$ compute a $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext for a participant P', where $\mathsf{id}_{P'} \in \pi^{\widetilde{\mathsf{gid}}}_{P^{\widetilde{\mathsf{gid}}}}.GP$ but P' is unauthorized for the group $\widetilde{\mathsf{gid}}$, the challenger replaces this $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext by a random ciphertext of the same length.

Recall in **Game** 2 that we have the key k_{PP} computed by all authorized parties P in the group $\widetilde{\mathsf{gid}}$ be uniformly at random. This in particular means that all keys k_{PP} generated by the leader $P^{\widetilde{\mathsf{gid}}}$ for all participants P', which are unauthorized for the group $\widetilde{\mathsf{gid}}$, are random. Note also that the leader must be in the party set $\mathsf{id}_{P^{\widetilde{\mathsf{gid}}}} \in GP^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$ for all gkid unless the group ends. Thus for all party P' and group key index gkid, the short-term state $\pi^{\widetilde{\mathsf{gid}}}_{P^{\widetilde{\mathsf{gid}}}}$ must not be compromised, if $\mathsf{frsh}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\mathsf{KPriv}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$ holds. Then, we analyze the gap between **Game** 4 and **Game** $C2.5$ by n hybrid games, where n denotes the number of register requests sent by unauthorized party. Obviously, it holds that $n \leq c_{\mathsf{maxReg}}$.

Hybrid Game 0. This game is identical to **Game** 4. Thus, we have that

$$\mathsf{Adv}_{\mathsf{hy}.0}(\mathcal{A}) = \mathsf{Adv}_4^{C2}(\mathcal{A})$$

Hybrid Game i, where $1 \leq i \leq n$. This game is identical to **Hybrid Game** $(i-1)$ except the following modifications:

- Whenever the challenger needs to sample a random key k_{PP} in a query $\textsc{SendJoinAuth}(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, m)$, where $P = P^{\widetilde{\mathsf{gid}}}$, P' is the unauthorized party that sends the i-th register request, and $\mathsf{gid} = \widetilde{\mathsf{gid}}$, the challenger do not sample this key but mark this key as k^i_{PP}.
- Whenever the challenger needs to compute an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext that is encrypted upon the key k^i_{PP}, the challenger first checks whether a ciphertext has been produced upon the same input. If such ciphertext exists, then the challenger simply reuses this ciphertext. If not, then the challenger samples a ciphertext of the same length uniformly at random.

If the attacker $\mathcal{A}$ can distinguish **Hybrid Game** $(i-1)$ and **Hybrid Game** i, then we can easily construct an attacker $\mathcal{B}_4$ that breaks the $\mathsf{IND\$\text{-}CCA}$ security of the underlying $\mathsf{AEAD}_{\mathsf{PP}}$. Thus, we can easily have that

$$\mathsf{Adv}_{\mathsf{hy}.(i-1)}(\mathcal{A}) = \mathsf{Adv}_{\mathsf{hy}.i}(\mathcal{A}) + \epsilon^{\mathsf{ind\$\text{-}cca}}_{\mathsf{AEAD}_{\mathsf{PP}}}$$

Hybrid Game n. This game is identical to **Game** $C2.5$. Thus, we have that

$$\mathsf{Adv}_{\mathsf{hy}.n}(\mathcal{A}) = \mathsf{Adv}_5^{C2}(\mathcal{A})$$

To sum up, it holds that

$$\begin{aligned}\mathsf{Adv}_4^{C2}(\mathcal{A}) &\leq \mathsf{Adv}_5^{C2}(\mathcal{A}) + n\epsilon^{\mathsf{ind\$\text{-}cca}}_{\mathsf{AEAD}_{\mathsf{PP}}} \\ &\leq \mathsf{Adv}_5^{C2}(\mathcal{A}) + c_{\mathsf{maxReg}}\epsilon^{\mathsf{ind\$\text{-}cca}}_{\mathsf{AEAD}_{\mathsf{PP}}}\end{aligned}$$

Final Analysis Of Case 2. Now, we prove that $\mathcal{A}$ cannot win by triggering the events E_{KPriv} by reduction. If the attacker $\mathcal{A}$ can win against Π' by triggering E_{KPriv}, the we can construct another attacker $\mathcal{B}_5$ that breaks the Sec-mGKD-pki security of Π by triggering the event E_{KPriv}. The attacker $\mathcal{B}_5$ honestly simulates **Game** $C2.5$ to $\mathcal{A}$ except for the following modifications:

- $\textsc{NewParty}(\mathsf{id}_P)$: $\mathcal{B}_5$ simply forwards this query to its challenger and the reply to $\mathcal{A}$.
- $\textsc{NewGroup}(\mathsf{id}_P, \mathsf{gid})$: $\mathcal{B}_5$ simply forwards this query to its challenger and the reply to $\mathcal{A}$. Moreover, $\mathcal{B}_5$ samples a random pw^{gid} from the distribution $\mathcal{D}_{pw}$ for the group gid.
- $\textsc{Auth}(\mathsf{gid}, \mathsf{id}_P)$: $\mathcal{B}_5$ simply forwards this query to its challenger.
- $\textsc{RegisterAuth}(\mathsf{id}_P, \mathsf{gid}, m)$: $\mathcal{B}_5$ simply forwards this query to its challenger for a reply c. If P is not the leader of the group gid, then $\mathcal{B}_5$ additionally runs the first pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon pw^{gid} and returns c together with the outgoing message of $\mathsf{PAKE}_{\mathsf{PP}}$ to $\mathcal{A}$.
- $\textsc{RegisterInject}(\mathsf{id}_P, \mathsf{gid}, gs, m)$: We consider two case:
  - If $\mathsf{gid} = \widetilde{\mathsf{gid}}$, then $\mathcal{B}_5$ first sends a query $\textsc{Corrupt}(\mathsf{id}_P)$ to its challenger. Afterwards, $\mathcal{B}_5$ honestly runs $\mathsf{Register}\text{-}\mathsf{P}'(P, \mathsf{gid}, gs, m)$ by himself.
  - If $\mathsf{gid} \neq \widetilde{\mathsf{gid}}$, then $\mathcal{B}_5$ parses gs into two portions (gs_Π, pw). Next, $\mathcal{B}_5$ simply forwards the query $\textsc{RegisterInject}(\mathsf{id}_P, \mathsf{gid}, gs_\Pi, m)$ to its challenger for a reply c. Then $\mathcal{B}_5$ runs the first pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon pw and returns c together with the outgoing message of $\mathsf{PAKE}_{\mathsf{PP}}$ to $\mathcal{A}$.

- $\textsc{SendJoinAuth}(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, m)$: We consider two cases: For the first case that $\mathsf{gid} = \widetilde{\mathsf{gid}}$, we consider the following two steps:
  - If the party P of the group gid has not derived the key k_{PP}, then $\mathcal{B}_5$ runs the next pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon necessary information from the input message m and the password pw^{gid}. If the party P of the group gid now is expected to derive the key k_{PP} of $\mathsf{PAKE}_{\mathsf{PP}}$ and there is no other party P'' in the group $\widetilde{\mathsf{gid}}$ that has the same transcript of $\mathsf{PAKE}_{\mathsf{PP}}$ as P, then $\mathcal{B}_5$ replaces it by a random key of the same length.

    If the party P of the group gid now is expected to derive the key k_{PP} of $\mathsf{PAKE}_{\mathsf{PP}}$ and there is a party P'' in the group $\widetilde{\mathsf{gid}}$ that has the same transcript of $\mathsf{PAKE}_{\mathsf{PP}}$ as P, then $\mathcal{B}_5$ replaces the key of P with the one of P''.
  - If the party P of the group gid now has derived the key k_{PP}, then $\mathcal{B}_5$ first checks whether P' is authorized for the group gid or not. If P' is authorized for the group gid, then $\mathcal{B}_5$ further checks whether P agrees on the party P''s sign-up message $m^{P'}_{\mathsf{SignUp}}$. If not, then $\mathcal{B}_5$ simply aborts. Otherwise, $\mathcal{B}_5$ checks whether m should include an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext. If so, this $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext must be encrypted by the party P' from some message m_1. Then, $\mathcal{B}_5$ simply sends the query

$\textsc{SendJoinAuth}(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, m_1)$ to its challenger for a reply m_2. After that, if m_2 is not an empty string, then $\mathcal{B}_5$ encrypts m_2 using $\mathsf{AEAD}_{\mathsf{PP}}$ upon the random key k_{PP}, a random nonce, an header consisting the sign-up messages of P and P'.

If P' is unauthorized for the group gid and this invocation needs to outputs some $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext, $\mathcal{B}_5$ simply samples the $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext randomly. If $\mathsf{id}_{P'}$ is expected to be added in to the set $\pi^{\widetilde{\mathsf{gid}}}_{P^{\widetilde{\mathsf{gid}}}}.GP$, $\mathcal{B}_5$ does not add $\mathsf{id}_{P'}$ into this set. Instead, $\mathcal{B}_5$ marks it as "fake".

Finally, $\mathcal{B}_5$ forwards all outgoing messages in this algorithm to $\mathcal{A}$.

For the second case that $\mathsf{gid} \neq \widetilde{\mathsf{gid}}$, we consider the following two steps:

– If the party P of the group gid has not derived the key k_{PP}, then $\mathcal{B}_5$ runs the next pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon necessary information from the input message m and the password pw^{gid}.

– If the party P of the group gid now has derived the key k_{PP}, then $\mathcal{B}_5$ checks whether m should include an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext. If so, $\mathcal{B}_5$ decrypts the $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext using the key k_{PP} and other necessary information from the input m for a message m_1. Then, $\mathcal{B}_5$ sends the query $\textsc{SendJoinAuth}(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, m_1)$ to its challenger for a reply m_2. After that, if m_2 is not an empty string, then $\mathcal{B}_5$ encrypts m_2 using $\mathsf{AEAD}_{\mathsf{PP}}$ upon the key k_{PP}, a random nonce, an header consisting the sign-up messages of P and P'.

Finally, $\mathcal{B}_5$ forwards all outgoing messages in this algorithm to $\mathcal{A}$.

- $\textsc{SendJoinInject}(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m)$: We consider two cases. For the first case that $\mathsf{gid} = \widetilde{\mathsf{gid}}$, $\mathcal{B}_5$ simply computes $\mathsf{Join\text{-}P}(P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m)$ by himself.

For the second case that $\mathsf{gid} \neq \widetilde{\mathsf{gid}}$, $\mathcal{B}_5$ parses gs into two portion gs_{Π} and pw. Then, we consider the following two steps:

– If the party P of the group gid has not derived the key k_{PP}, then $\mathcal{B}_5$ runs the next pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon necessary information from the input message m and the password pw.

– If the party P of the group gid now has derived the key k_{PP}, then $\mathcal{B}_5$ checks whether m should include an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext. If so, $\mathcal{B}_5$ decrypts the $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext using the key k_{PP} and other necessary information from the input m for a message m_1. Then, $\mathcal{B}_5$ sends the query $\textsc{SendJoinInject}(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, gs_{\Pi}, m_1)$ to its challenger for a reply m_2. After that, if m_2 is not an empty string, then $\mathcal{B}_5$ encrypts m_2 using $\mathsf{AEAD}_{\mathsf{PP}}$ upon the key k_{PP}, a random nonce, an header consisting the sign-up messages included in m.

Finally, $\mathcal{B}_5$ forwards all outgoing messages in this algorithm to $\mathcal{A}$.

- $\textsc{SendLeave}(\mathsf{id}_P, \mathsf{gid}, \mathsf{id}_{P'})$: We consider two cases: For the first case that $\mathsf{gid} = \widetilde{\mathsf{gid}}$, we further consider the following three sub-cases:

  - If $\mathsf{id}_P = \mathsf{id}_{P_{\widetilde{\mathsf{gid}}}}$ and P' is unauthorized for the group $\widetilde{\mathsf{gid}}$ and marked as "fake", then $\mathcal{B}$ removes the mark "fake" on P'.
  - If P is marked as "fake", then $\mathcal{B}_5$ executes $\mathsf{Leave\text{-}P}(P, \mathsf{gid}, \mathsf{id}_{P'})$ by himself.
  - For all other queries, $\mathcal{B}_5$ forwards them to its challenger and the reply to $\mathcal{A}$.

  For the second case that $\mathsf{gid} \neq \widetilde{\mathsf{gid}}$, $\mathcal{B}_5$ simply forwards this query to its challenger. If a per-group state π needs to be erased, then $\mathcal{B}_5$ also erases the corresponding $\mathsf{PAKE}_{\mathsf{PP}}$ secrets of π.

- $\textsc{EndGroup}(\mathsf{gid})$: We consider the following two case: For the first case that $\mathsf{gid} = \widetilde{\mathsf{gid}}$, if there exists no party P' that is still marked as "fake", then $\mathcal{B}_5$ forwards this query to its challenger. Otherwise, $\mathcal{B}_5$ aborts.

  For the second case that $\mathsf{gid} \neq \widetilde{\mathsf{gid}}$, $\mathcal{B}_5$ simply forwards this query to its challenger. If the leader's per-group state π needs to be erased, then $\mathcal{B}_5$ also erases the corresponding $\mathsf{PAKE}_{\mathsf{PP}}$ secrets of π.

- $\textsc{SendKeyRotat}(\mathsf{id}_P, \mathsf{gid}, m)$ We consider two cases: For the first case that $\mathsf{gid} = \widetilde{\mathsf{gid}}$, we further consider following three cases:

  - If P is the leader in the group gid, then $\mathcal{B}_5$ first queries $\textsc{SendKeyRotat}(\mathsf{id}_P, \mathsf{gid}, m)$ to its challenger for a reply c. Then, $\mathcal{B}_5$ extracts the portion $c_{\overline{P}}$ in c that is specific to every participant $\overline{P}$ in the group gid, followed by encrypting it using the corresponding $\mathsf{AEAD}_{\mathsf{PP}}$ key k_{PP}, a random nonce, and an header that is same as the one in the corresponding Participant Join phase. Moreover, for every party P' that is marked as "fake", $\mathcal{B}_5$ also samples a random $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext and a random nonce. Finally, $\mathcal{B}_5$ outputs all above $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertexts and nonces. If any error occurs during the above execution, then $\mathcal{B}_5$ aborts and undoes the above executions.
  - If P is an authorized participant in the group gid, then $\mathcal{B}_5$ first extracts an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext and an $\mathsf{AEAD}_{\mathsf{PP}}$ nonce from the input m. If this $\mathsf{AEAD}_{\mathsf{PP}}$ is not produced by the leader for some message m_1, then $\mathcal{B}_5$ aborts. Otherwise, $\mathcal{B}_5$ extracts other necessary information m_2 from the input m for querying $\textsc{SendKeyRotat}(\mathsf{id}_P, \mathsf{gid}, m_1 \parallel m_2)$ returns the reply m_{KRot} to $\mathcal{A}$. If any error occurs during the above execution, then $\mathcal{B}_5$ aborts and undoes the above executions.
  - If P is an unauthorized participant in the group gid, then $\mathcal{B}_5$ executes $\mathsf{KeyRotat\text{-}P}(P, \mathsf{gid}, m)$ by himself.

  For the second case that $\mathsf{gid} \neq \widetilde{\mathsf{gid}}$, we further consider following two sub-cases:

- If P is the leader in the group gid, then $\mathcal{B}_5$ first queries $\textsc{SendKeyRotat}(\mathsf{id}_P, \mathsf{gid}, m)$ to its challenger for a reply c. Then, $\mathcal{B}_5$ extracts the portion $c_{\overline{P}}$ in c that is specific to every participant $\overline{P}$ in the group gid, followed by encrypting it using the corresponding $\mathsf{AEAD}_{\mathsf{PP}}$ key k_{PP}, a random nonce, and an header that is same as the one in the corresponding Participant Join phase. Finally, $\mathcal{B}_5$ outputs the $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext and nonce for every participant $\overline{P}$ in the group gid. If any error occurs during the above execution, then $\mathcal{B}_5$ aborts and undoes the above executions.
- If P is a participant in the group gid, then $\mathcal{B}_5$ first extracts an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext and an $\mathsf{AEAD}_{\mathsf{PP}}$ nonce from the input m. Next, $\mathcal{B}_5$ recovers a message m_1 from the $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext using the corresponding $\mathsf{AEAD}_{\mathsf{PP}}$ key k_{PP}, the $\mathsf{AEAD}_{\mathsf{PP}}$ nonce, and an header that is same as the one in the corresponding Participant Join phase. Then, $\mathcal{B}_5$ extracts other necessary information m_2 from the input m for querying $\textsc{SendKeyRotat}(\mathsf{id}_P, \mathsf{gid}, m_1 \parallel m_2)$ returns the reply m_{KRot} to $\mathcal{A}$. If any error occurs during the above execution, then $\mathcal{B}_5$ aborts and undoes the above executions.

- $\textsc{Corrupt}(\mathsf{id}_P)$: $\mathcal{B}_5$ simply forwards this query to its challenger and the reply to $\mathcal{A}$.
- $\textsc{Compromise}(\mathsf{id}_P, \mathsf{gid})$: We consider the following two cases. For the first case that $\mathsf{gid} = \widetilde{\mathsf{gid}}$, we further consider the following two sub-cases:

  - If P is an authorized party for the group gid, then $\mathcal{B}_5$ forwards this query to its challenger. If the reply is not $\perp$, then $\mathcal{B}_5$ forwards the reply together with the key k_{PP} of P in the group gid to $\mathcal{A}$. Otherwise, $\mathcal{B}_5$ simply returns $\perp$ to $\mathcal{A}$.
  - If P is an unauthorized party for the group gid, then $\mathcal{B}_5$ must create π_P^{gid} by himself. In this case, $\mathcal{B}_5$ simply returns π_P^{gid} to $\mathcal{A}$.

  For the second case that $\mathsf{gid} \neq \widetilde{\mathsf{gid}}$, $\mathcal{B}_5$ forwards this query to its challenger. If the reply is not $\perp$, then $\mathcal{B}_5$ forwards the reply together with the key k_{PP} of P in the group gid to $\mathcal{A}$. Otherwise, $\mathcal{B}_5$ simply returns $\perp$ to $\mathcal{A}$.
- $\textsc{Leak}(\mathsf{id}_P, \mathsf{gid}, \mathsf{gkid})$: We consider the following two cases. For the first case that $\mathsf{gid} = \widetilde{\mathsf{gid}}$, we further consider the following two cases:

  - If P is an authorized party for the group gid, then $\mathcal{B}_5$ simply forwards this query to its challenger and the reply to $\mathcal{A}$.
  - If P is an unauthorized party for the group gid, then $\mathcal{B}_5$ must create $gk_P^{(\mathsf{gid},\mathsf{gkid})}$ by himself. In this case, $\mathcal{B}_5$ simply returns $gk_P^{(\mathsf{gid},\mathsf{gkid})}$ to $\mathcal{A}$.

  For the second case that $\mathsf{gid} \neq \widetilde{\mathsf{gid}}$, $\mathcal{B}_5$ simply forwards this query to its challenger and the reply to $\mathcal{A}$.
- $\textsc{Reveal}(\mathsf{gid})$: $\mathcal{B}_5$ forwards this query to its challenger. Then, $\mathcal{B}_5$ forwards the reply together with the password pw^{gid} to $\mathcal{A}$.

- $\textsc{Test}(\mathsf{id}_P, \mathsf{gid}, \mathsf{gkid})$: $\mathcal{B}_5$ simply forwards this query to its challenger and the reply to $\mathcal{A}$.

Note that $\mathcal{B}_5$ perfectly simulates **Game** $C2.5$ to $\mathcal{A}$. Besides, $\mathcal{B}_5$ simulates the behaviors of all unauthorized parties in the group $\widetilde{\mathsf{gid}}$ by himself. Thus, the parties in the group $\widetilde{\mathsf{gid}}$ in the $\mathsf{Sec\text{-}mGKD\text{-}pw}$ experiment between $\mathcal{B}_5$ and its challenger are identical to the authorized parties in the group $\widetilde{\mathsf{gid}}$ in the $\mathsf{Sec\text{-}mGKD\text{-}pki}$ experiment between $\mathcal{A}$ and $\mathcal{B}_5$.

Note in **Game** 4 that we ensure that the sign-up messages are honestly distributed between the leader and authorized participants $P \in GP^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$ for the tested group $\widetilde{\mathsf{gid}}$ and group key index gkid. Thus, for the tested party P', the tested group $\widetilde{\mathsf{gid}}$, and the tested group key index gkid, if $\mathcal{A}$ can trigger E_{KPriv} without violating the freshness condition $\mathsf{frsh}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\mathsf{KPriv}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$, then $\mathcal{B}$ can also trigger E_{KPriv} by outputting the same b' as $\mathcal{A}$, without violating the freshness condition $\mathsf{frsh}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\mathsf{KPriv}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$. It holds that

$$\mathsf{Adv}^{C2}_{5}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\Pi}(\mathcal{B}_5)$$

Final Analysis Of The Full Proof. By merging the statements above, if $\mathcal{A}$ can break the $\mathsf{Sec\text{-}mGKD\text{-}pw}$ security of Π', then there exists an attacker $\mathcal{B}$ that breaks the $\mathsf{Sec\text{-}mGKD\text{-}pki}$ security of Π, such that

$$\begin{aligned}\mathsf{Adv}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\Pi'}(\mathcal{A}) \leq & q_{\textsc{NewGroup}}\Big(\epsilon^{\mathsf{w\text{-}PAKE}}_{\mathsf{PAKE}_{\mathsf{PP}},\mathcal{D}_{pw}} + \epsilon^{d\text{-}\mathsf{frob}}_{\mathsf{AEAD}_{\mathsf{PP}}} \\ & + c_{\mathsf{maxReg}}(\epsilon^{\mathsf{cti\text{-}cpa}}_{\mathsf{AEAD}_{\mathsf{PP}}} + \epsilon^{\mathsf{ind\$\text{-}cca}}_{\mathsf{AEAD}_{\mathsf{PP}}}) + \mathsf{Adv}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\Pi}(\mathcal{B})\Big)\end{aligned}$$

□

6.9.3 Proof of Theorem 27

Proof. The proof is given by reduction. If there exists any PPT attacker $\mathcal{A}$ that breaks the $\mathsf{Sec\text{-}mGKD\text{-}pki}$ of Π', then we construct an attacker $\mathcal{B}$ that breaks the $\mathsf{Sec\text{-}mGKD\text{-}pki}$ of Π. The attacker $\mathcal{B}$ initializes the $\mathsf{Sec\text{-}mGKD\text{-}pki}$ experiment and answers $\mathcal{A}$'s oracle queries as follows:

- $\textsc{NewParty}(\mathsf{id}_P)$: $\mathcal{B}$ simply forwards this query to its challenger and then forwards the reply to $\mathcal{A}$.
- $\textsc{NewGroup}(\mathsf{id}_P, \mathsf{gid})$: $\mathcal{B}$ forwards this query to its challenger. Then, $\mathcal{B}$ samples a password pw^{gid} from the distribution $\mathcal{D}_{pw}$ and associate the password with the group gid.
- $\textsc{Auth}(\mathsf{gid}, \mathsf{id}_P)$: $\mathcal{B}$ simply forwards this query to its challenger.
- $\textsc{RegisterAuth}(\mathsf{id}_P, \mathsf{gid}, m)$: $\mathcal{B}$ forwards this query to its challenger for a reply c_1. If P is the leader of the group gid, $\mathcal{B}$ simply returns c_1. Otherwise, $\mathcal{B}$ runs the first pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon the password pw^{gid} for a ciphertext c_2. Finally, $\mathcal{B}$ returns (c_1, c_2) to $\mathcal{A}$.

- $\textsc{RegisterInject}(\mathsf{id}_P, \mathsf{gid}, gs, m)$: $\mathcal{B}$ first parses gs into two portions gs_Π and pw. Next, $\mathcal{B}$ sends query $\textsc{RegisterInject}(\mathsf{id}_P, \mathsf{gid}, gs_\Pi, m)$ to its challenger for a reply c_1. Then, $\mathcal{B}$ runs the first pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon the password pw for a ciphertext c_2. Finally, $\mathcal{B}$ returns (c_1, c_2) to $\mathcal{A}$.

- $\textsc{SendJoinAuth}(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, m)$: We consider two steps.

  The first step is executed if the party P has not output the key k_{PP} of the $\mathsf{PAKE}_{\mathsf{PP}}$ with P' in the group gid. $\mathcal{B}$ first extracts necessary information from the input m and runs the next pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon pw^{gid}. If the key k_{PP} is available, $\mathcal{B}$ remembers this key for both parties P and P' in the group gid. Otherwise, $\mathcal{B}$ simply outputs the outgoing message of $\mathsf{PAKE}_{\mathsf{PP}}$.

  The second step is executed if the party P has already output the key k_{PP} of the $\mathsf{PAKE}_{\mathsf{PP}}$ with P' in the group gid. $\mathcal{B}$ first checks whether the input m includes any $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext and the $\mathsf{AEAD}_{\mathsf{PP}}$ nonce. If so, $\mathcal{B}$ first decrypts it using the key k_{PP}, the $\mathsf{AEAD}_{\mathsf{PP}}$ nonce, the header consisting both P and P''s sign-up messages, for a message m_1. Otherwise, the message m_1 is set to empty string $\top$. Then, $\mathcal{B}$ sends the query $\textsc{SendJoinAuth}(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, m_1)$ to its challenger for a reply m_2. Finally, $\mathcal{B}$ encrypts the message m_2 using the key k_{PP}, a random nonce, the header consisting both P and P''s sign-up messages, for a ciphertext.

  The ciphertexts of $\mathsf{AEAD}_{\mathsf{PP}}$ and optionally the one of $\mathsf{PAKE}_{\mathsf{PP}}$ (if available) are returned to $\mathcal{A}$.

- $\textsc{SendJoinInject}(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m)$: $\mathcal{B}$ first parses gs into two portions gs_Π and pw. Then, we consider two steps.

  The first step is executed if the party P has not output the key k_{PP} of the $\mathsf{PAKE}_{\mathsf{PP}}$ with P' in the group gid. $\mathcal{B}$ first extracts necessary information from the input m and runs the next pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon pw. If the key k_{PP} is available, $\mathcal{B}$ remembers this key for both parties P and P' in the group gid. Otherwise, $\mathcal{B}$ simply outputs the outgoing message of $\mathsf{PAKE}_{\mathsf{PP}}$.

  The second step is executed if the party P has already output the key k_{PP} of the $\mathsf{PAKE}_{\mathsf{PP}}$ with P' in the group gid. $\mathcal{B}$ first checks whether the input m includes any $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext and the $\mathsf{AEAD}_{\mathsf{PP}}$ nonce. If so, $\mathcal{B}$ first decrypts it using the key k_{PP}, the $\mathsf{AEAD}_{\mathsf{PP}}$ nonce, the header consisting both P and P''s sign-up messages, for a message m_1. Otherwise, the message m_1 is set to empty string $\top$. Then, $\mathcal{B}$ sends the query $\textsc{SendJoinInject}(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, gs_\Pi, m_1)$ to its challenger for a reply m_2. Finally, $\mathcal{B}$ encrypts the message m_2 using the key k_{PP}, a random nonce, the header consisting both P and P''s sign-up messages, for a ciphertext.

  The ciphertexts of $\mathsf{AEAD}_{\mathsf{PP}}$ and optionally the one of $\mathsf{PAKE}_{\mathsf{PP}}$ (if available) are returned to $\mathcal{A}$.

- $\textsc{SendLeave}(\mathsf{id}_P, \mathsf{gid}, \mathsf{id}_{P'})$: $\mathcal{B}$ forwards this query to its challenger. If $\mathsf{id}_P = \mathsf{id}_{P'}$, then $\mathcal{B}$ also removes the key k_{PP} of the party P generated in the Participant Join phase for Pjoining the group gid. If P is the leader of the group gid, then $\mathcal{B}$ removes the key k_{PP} of the party P generated in the Participant Join phase for P' the group gid.
- $\textsc{EndGroup}(\mathsf{gid})$: $\mathcal{B}$ forwards this query to its challenger. Then $\mathcal{B}$ also removes all remaining keys k_{PP} of the leader of the group gid.
- $\textsc{SendKeyRotat}(\mathsf{id}_P, \mathsf{gid}, m)$: If P is the leader of the group gid, then $\mathcal{B}$ forwards this query to its challenger for a reply c. Then, the $\mathcal{B}$ extracts the portion $c_{\overline{P}}$ in c that is specific to every participant $\overline{P}$ in the group gid, followed by encrypting it using the stored corresponding $\mathsf{AEAD}_{\mathsf{PP}}$ key k_{PP}, a random nonce, and an header consisting of the leader P and the participant $\overline{P}$' sign-up messages as in the Participant Join phase. Finally, $\mathcal{B}$ outputs the $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext and nonce for every participant $\overline{P}$ in the group gid. If any error occurs during the above execution, then the leader P aborts and undoes the above executions.

  Otherwise, P is the participant of the group gid. $\mathcal{B}$ first extracts an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext and an $\mathsf{AEAD}_{\mathsf{PP}}$ nonce from the input m. Next, $\mathcal{B}$ recovers a message m_1 from the $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext using the stored corresponding $\mathsf{AEAD}_{\mathsf{PP}}$ key k_{PP}, the $\mathsf{AEAD}_{\mathsf{PP}}$ nonce, and an header consisting of the participant P's and the leader's sign-up messages as in the Participant Join phase. Then, $\mathcal{B}$ extracts other necessary information m_2 from the input m for querying $\textsc{SendKeyRotat}(P, \mathsf{gid}, m_1 \parallel m_2)$ to its challenger. Finally, $\mathcal{B}$ forwards the reply from the challenger to $\mathcal{A}$. If any error occurs during the above execution, then the participant P aborts and undoes the above executions.
- $\textsc{Corrupt}(\mathsf{id}_P)$: $\mathcal{B}$ simply forwards this query to its challenger and then forwards the reply to $\mathcal{A}$.
- $\textsc{Compromise}(\mathsf{id}_P, \mathsf{gid})$: $\mathcal{B}$ simply forwards this query to its challenger for a state π. Then, $\mathcal{B}$ forwards π together with all keys k_{PP} of party P for the group gid to $\mathcal{A}$.
- $\textsc{Leak}(\mathsf{id}_P, \mathsf{gid}, \mathsf{gkid})$: $\mathcal{B}$ simply forwards this query to its challenger and then forwards the reply to $\mathcal{A}$.
- $\textsc{Reveal}(\mathsf{gid})$: $\mathcal{B}$ simply forwards this query to its challenger for a group secret gs_Π. Then, $\mathcal{B}$ forwards the group secret gs_Π together with the password pw^{gid} to $\mathcal{A}$.
- $\textsc{Test}(\mathsf{id}_P, \mathsf{gid}, \mathsf{gkid})$: $\mathcal{B}$ simply forwards this query to its challenger and then forwards the reply to $\mathcal{A}$.

It is easy to know that $\mathcal{B}$ perfectly simulates $\mathsf{Sec\text{-}mGKD\text{-}pki}$ experiment to $\mathcal{A}$ and wins if $\mathcal{A}$ wins. The proof is concluded by

$$\mathsf{Adv}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\mathsf{mGKD}'}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\mathsf{mGKD}}(\mathcal{B})$$

□

Chapter 7

Trade-off Areas in Secure Messaging Design

This chapter is based on the paper:

Cas Cremers and Mang Zhao, "Secure Messaging with Strong Compromise Resilience, Temporal Privacy, and Immediate Decryption", in 2024 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, US, 2024.

This paper was joint work with my supervisor Cas Cremers. I lead the research on this paper and the substantial contributions in this chapter are my own. My co-author principally contributed to the initial conception of the work and the final write up of the paper.

7.1 Introduction

Driven by the global uptake of the Signal protocol, which has been widely deployed in many messaging applications worldwide by virtue of its high efficiency and strong security guarantees, there have been many advances in the theory and design of messaging protocols with desirable efficiency and security properties during the last decade. We highlight three of these properties.

(i) Immediate Decryption with Constant-Size Overhead: This property, which is essential for practical messaging apps and was formally studied by Alwen et al. [12], requires that the recipients can decrypt every message at the time of arrival, irrespective of the arrival of prior messages. Conventional messaging solutions reuse a static encryption/decryption key pair during every two-party conversation (aka. session). However, the leakage of the private decryption keys indicates the loss of privacy of all messages in the past and/or future. Two basic security properties are formalized for modern messaging protocols: forward secrecy (FS) and post-compromise security (PCS) [73]. While FS requires the privacy of past messages prior to the state expose, PCS enables the parties to recover from state exposure. Common modern messaging solutions obtain strong security guarantees by making their encryption keys dependent in some way on all previously sent messages. However, in realistic messaging settings, messages can arrive out-of-order or may be lost forever. If message n arrives before message $n-1$, it cannot be decrypted until message $n-1$ arrives; and if it never arrives, communications become stuck. In theory, this can be naively solved by appending all previous ciphertexts to the next message sent. In practice, this naive solution is unusable, as practical applications require constant-size overhead for messages. The Signal protocol is a pioneering example in the domain of messaging with relatively strong security and immediate decryption with constant overhead.

(ii) Temporal Privacy: State compromise does not cause loss of privacy of messages sent prior to a time interval and can be healed after every time interval. Pijnenburg and Pöttering [153] first observe that the immediate decryption restricts FS by definition: an attacker that intercepts a message and corrupts the receiver in the future can always compromise this message. To solve this, [153] proposes a time-based BOOM protocol that expires old keys and updates new keys after a specific time interval. Intuitively, this solves the restricted FS problem as attackers cannot corrupt the expired keys that have been erased from the state. However, every party in BOOM obtains the partner's latest public key only when receiving the partner's latest message. If two parties do not frequently exchange messages, the restricted FS problem remains. A trivial fix is to force every party to frequently send "empty messages" for key updates. However, due to the key-updatable framework underlying BOOM, this solution potentially yields linearly growing bandwidth.

The original Signal protocol satisfies a similar temporal privacy property but only for new conversations. Conceptually, the Signal protocol defines the initial *Extended Triple-Diffie-Hellman* (X3DH) asynchronous key exchange [138] and the *Double Ratchet* (DR) [152] for the subsequent message exchanges. Note that the X3DH key establishment uses the combination of public/private keys with different lifetimes, i.e., long-term, medium-term, and one-time. Even if all previous keys are compromised, the privacy of new conversations can still be recovered if the honest recipients upload their new medium-term keys. Conversely, the privacy of all past conversations under a certain medium-term key holds if that key is not leaked, even if other keys are leaked.

(iii) Resilience against Fine-Grained State Compromise: The compromise of senders' and recipients' state does not cause loss of privacy and authenticity, respectively. Modern secure messaging protocols like Signal [75] have been fundamentally designed to be resilient against a weak form of state compromise: The state is healed from compromise after a back-and-forth interaction, i.e., PCS. However, Alwen et al. [12] notice that such state compromise resilience of Signal is very coarse rather than "fine-grained": corruption of the state of either party in a conversation will cause the loss of both privacy and authenticity, since the privacy and authenticity of messages depend on a symmetric secret that is present in both parties' states. It is however possible to achieve the stronger notion of resilience against fine-grained compromise by breaking this symmetry: in the literature, a number of "optimal-secure" protocols [85, 118, 122, 153, 154] provably achieve such resilience against fine-grained compromise.

Challenges: Perhaps surprisingly, while each of the above properties have been studied in isolation, there currently exists no provably secure protocol that simultaneously offers the above three desirable properties.

Alwen et al. [12] generalize DR of Signal to a new SM protocol, based on which another TR protocol [50] is proposed with slightly stronger security. However, the original Signal, SM, and TR all satisfy immediate decryption with constant-size overhead but lack the resilience against fine-grained state compromise. To the best of our knowledge, the BOOM protocol [153] is the only known protocol that provides the temporal privacy. Moreover, similar to other "optimal-secure" protocols [85, 118, 122, 154] in the literature, the BOOM protocol also provides a flavor of very strong security guarantee (we call it "ID-optimal") that includes the resilience against fine-grained state compromise. However, all these optimally secure protocols lack immediate decryption with constant-size overhead. We summarize the situation for related provably secure protocols in Figure 7.1.

Contributions: Our main contribution is the first provably secure messaging protocol with immediate decryption and constant-size overhead, temporal privacy, and resilience against fine-grained state compromise. To this end, we introduce a related new strong

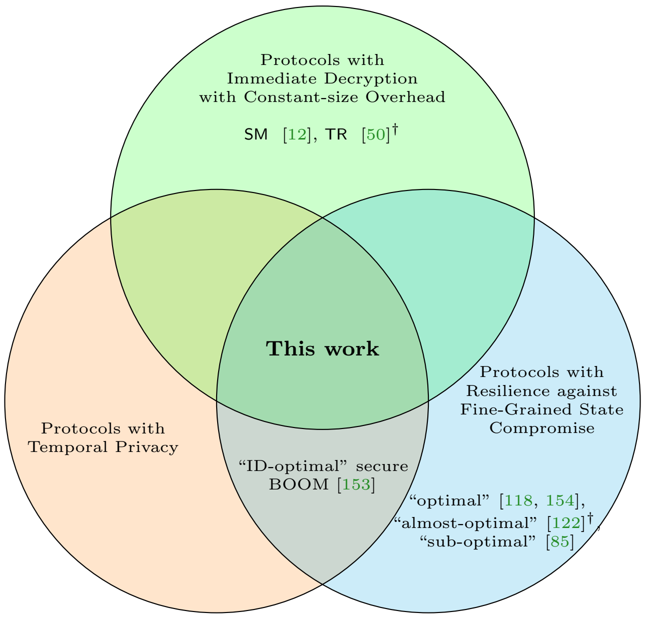


Figure 7.1: Comparison between this work and other existing protocols with provable security properties w.r.t. (i) immediate decryption with constant-size overhead, (ii) temporal privacy, and (iii) resilience against fine-grained state compromise. All constructions in the diagram (including this work) are PQ-compatible except for the ones marked with †.

security notion called Extended-Secure-Messaging (eSM). We show that the eSM notion covers above strong properties and prove that our protocol meets it, in particular, in the PQ setting.

Furthermore, to show that our protocol is a suitable PQ-secure candidate for the DR in Signal, which is provably offline deniable, we extend the offline deniability definition for SPQR [63] (currently the only provably secure PQ-asynchronous key establishment) to the multi-stage setting. We prove that the combination of our eSM-secure protocol and SPQR is offline deniable, making it the first full messaging protocol that is provably offline deniable in the PQ setting.

Overview: We give background and related work in Section 7.2. We recall related cryptographic primitives in Section 7.3. We propose our new eSM syntax and security notion in Section 7.4. We propose our concrete protocol that is provably eSM-secure in Section 7.5, and show its offline-deniability when combined with SPQR in Section 7.6. For interested readers, we review related designs ACD19 and TR protocols in Section 7.7 and messaging protocols with various optimal security in Section 7.8. We compare our eSM security model with SM model in Section 7.9. We compare our eSM construction with ACD19 and TR protocols in Section 7.10. We provide the full proofs of our theorems in Section 7.11.

7.2 Background and Related Work

7.2.1 Instant Messaging Protocols and Immediate Decryption with Constant-Size Overhead

The Signal protocol provably offers strong security guarantees, such as *forward secrecy* and *post-compromise security* [72, 75], and *offline deniability* [175]. Moreover, Signal has several features that are critical for large-scale real-world deployment, such as *message-loss resilience* and *immediate decryption*. Roughly speaking, message-loss resilience and immediate decryption enable the receiver to decrypt a legitimate message immediately after it is received, even when some messages arrive out-of-order or are permanently lost by the network. Notably, the Signal protocol provides the above properties with constant-size overhead.

The core Signal protocol consists of two components: the *Extended Triple-Diffie-Hellman* (X3DH) initial key exchange and the *Double Ratchet* (DR) for subsequent message transmissions. Alwen et al. [12] introduce the notion of *Secure Messaging* (SM), which is a syntax and associated security notion that generalizes the security of Signal's DR. Alwen et al. also provide a concrete construction and prove that it is SM-secure. This construction is not explicitly named in [12]: in this work, we will refer to it as ACD19.

To the best of our knowledge, in addition to ACD19, the only known provably secure protocol that provides immediate decryption with constant-size overhead is the *Triple Ratchet* (TR) protocol [50]. However, the TR protocol is neither PQ-secure nor resilient against the fine-grained state compromise. We review the ACD19 and TR in details in Section 7.7. For the interested readers, we also compare ACD19 and TR with our protocol in Section 7.10.

7.2.2 Secure Messaging Protocols and Strong Security Guarantees

Alwen et al. [12] observe that the ACD19 protocol lacks resilience against fine-grained state compromise, because both encryption and decryption of a message in ACD19 uses the shared state of both parties in a conversation. The corruption of the shared state of either party immediately compromises the subsequent messages, no matter whether the corrupted party is the sender or receiver. To reduce the impact of state exposure, the authors also describe a second security notion for secure messaging, called PKSM, and a corresponding construction, which we call ACD19-PK. At a very high level, ACD19-PK extends ACD19 by encrypt-then-signing the output of the original SM protocol using a public key encryption (PKE) and a digital signature (DS). Intuitively, ACD19-PK reduces the impact of state compromise, since the attacker can neither recover the output of SM protocol (and further the real message) from the PKE ciphertext without knowing the recipient's decryption key, nor forge a valid ciphertext without knowing the sender's signing key. However, the main focus of [12] are SM and ACD19: for ACD19-PK, neither a formal security model nor a concrete proof is given; thus, its security is essentially conjectured.

In a parallel line of research, several messaging protocols have been proposed to meet various strong or even "optimal" security [19, 36, 85, 118, 122, 153, 154]. They follow different ratcheting frameworks aiming at various flavors of security, notably, all of which capture resilience against fine-grained compromise. Unfortunately, none of them provides immediate decryption with constant-size overhead, due to their key-update or state-update structures.

In particular, [153] observes that a protocol satisfying immediate decryption can only achieve a weak form of forward secrecy: an attacker that intercepts a message and corrupts the receiver in the future can always compromise this message. To solve this, [153] proposes a novel strong security model, which we call "ID-optimal", and a time-based BOOM protocol that periodically expires old keys and updates new keys. By this, neither the receiver nor an attacker who corrupts the receiver's state can decrypt a message that was encrypted under an expired key. The efficiency and security can be balanced by picking a reasonable time interval for key update and expiration. However, we find that the BOOM protocol has two constraints: On the one hand, every party in BOOM obtains the partner's latest public key only at the time of receiving the partner's latest message. If the message exchange between two parties are not frequent, then the restricted forward secrecy problem remains. On the other hand, the BOOM protocol also makes use of a complicated key-update mechanism and therefore provides immediate decryption with linearly growing bandwidth.

We review protocols that meet various "optimal" security in Section 7.8.

7.2.3 Offline Deniability and Post-Quantum Security

The property of *offline deniability* prevents a judge from deciding whether an honest user has participated in a conversation even when other participants try to frame them. The formal definition of offline deniability originates from [80] and [175] in the simulation-based models respectively for the authenticated key exchange (AKE) and full messaging protocols. These works also prove that several well-known classical AKE constructions, such as MQV, HMQV, 3DH, and X3DH, and the full Signal protocol are offline deniable.

Constructing PQ secure asynchronous key establishments is surprisingly complicated. There are a number of key establishment protocols [65, 81, 172, 173] that are potential candidates for PQ security. However, all of their security proofs rely on either the random oracle model or novel tailored assumptions, which are still not well-studied in the PQ setting. Hashimoto et al. [106] propose the first PQ secure key establishment but unfortunately have to assume that every party can pre-upload inexhaustible one-time keys for full asynchronicity. A subsequent work by Brendel et al. [63] proposed a new

PQ asynchronous deniable authenticated key exchange (DAKE) protocol, called SPQR, and a new game-based offline deniability notion. Brendel et al. prove that SPQR is offline deniable in the game-based paradigm against quantum (semi-honest) attackers.

To the best of our knowledge, SPQR is the only known PQ secure key establishment with full asynchronicity. Although it is straightforward that the combination of SPQR and ACD19 can form a PQ-secure full messaging protocol with promising privacy and authenticity, it is still an open question which flavors of offline deniability can be provably obtained for the combined protocols in the PQ setting.

7.3 Additional Preliminaries

We recall the DAKE scheme and its offline deniability notion from [63].

7.3.1 The DAKE scheme

Definition 60. *An asynchronous deniable authenticated key exchange (*DAKE*) protocol* Σ *is a tuple of algorithms* $\Sigma = (\Sigma.\mathsf{IdKGen}, \Sigma.\mathsf{PreKGen}, \Sigma.\mathsf{EpKGen}, \Sigma.\mathsf{Run}, \Sigma.\mathsf{Fake})$ *as defined below.*

- ***(Long-term) identity key generation*** $(\overline{ipk}_u, \overline{ik}_u) \xleftarrow{\$} \Sigma.\mathsf{IdKGen}()$*: outputs the identity public/private key pair of a party* u*.*
- ***(Medium-term) pre-key generation*** $(\overline{prepk}_u^{\mathsf{ind}}, \overline{prek}_u^{\mathsf{ind}}) \xleftarrow{\$} \Sigma.\mathsf{PreKGen}()$*: outputs the* ind*-th public/private key pair of a party* u*.*
- ***(Ephemeral) key generation*** $(\overline{epk}_u^{\mathsf{ind}}, \overline{ek}_u^{\mathsf{ind}}) \xleftarrow{\$} \Sigma.\mathsf{EpKGen}()$*: outputs the* ind*-th public/private key pair of user* u
- ***Session execution*** $(\pi', m') \xleftarrow{\$} \Sigma.\mathsf{Run}(\overline{ik}_u, \mathcal{L}_u^{\overline{prek}}, \mathcal{L}_{\mathsf{all}}^{\overline{ipk}}, \mathcal{L}_{\mathsf{all}}^{\overline{prepk}}, \pi, m)$*: inputs a party* u*'s long-term private key* $\overline{ik}_u$*, a list of* u*'s private pre-keys* $\mathcal{L}_u^{\overline{prek}}$*, lists of long-term and medium-term public keys for all honest parties* $\mathcal{L}_{\mathsf{all}}^{\overline{ipk}}$ *and* $\mathcal{L}_{\mathsf{all}}^{\overline{prepk}}$*, a session state* π*, and an incoming message* m*, and outputs an updated session state* π' *and a (possibly empty) outgoing message* m'*. To set up the session sending the first message,* $\Sigma.\mathsf{Run}$ *is called with a distinguished message* $m = \mathsf{create}$*.*
- ***Fake algorithm*** $(K, T) \xleftarrow{\$} \Sigma.\mathsf{Fake}(\overline{ipk}_u, \overline{ik}_v, \mathcal{L}_v^{\overline{prek}}, \mathsf{ind})$*: inputs one party* u*'s long-term identity public key* $\overline{ipk}_u$*, the other party* v*'s long-term identity private key* $\overline{ik}_v$*, a list of* v*'s private pre-keys* $\mathcal{L}_v^{\overline{prek}}$*, and an index of party* v*'s pre-key* ind *and generates a session key* K *and a transcript* T *of a protocol interaction between them.*

The session state π includes following variables (we only recall the ones related to the offline deniability):

```
$\mathrm{Expr}^{\mathsf{deni}}_{\Sigma,q_{\mathsf{P}},q_{\mathsf{M}},q_{\mathsf{S}}}(\mathcal{A})$:
$\mathcal{L}_{\mathsf{all}}, \mathcal{L}^{\overline{ipk}}_{\mathsf{all}}, \mathcal{L}^{\overline{prepk}}_{\mathsf{all}} \leftarrow \emptyset$
for $u \in [q_{\mathsf{P}}]$
    $\mathcal{L}^{\overline{prek}}_{u} \leftarrow \emptyset$
    $(\overline{ipk}_u, \overline{ik}_u) \xleftarrow{\$} \Sigma.\mathsf{IdKGen}()$
    $\mathcal{L}^{\overline{ipk}}_{\mathsf{all}} \xleftarrow{+} \overline{ipk}_u$
    $\mathcal{L}_{\mathsf{all}} \xleftarrow{+} (\overline{ipk}_u, \overline{ik}_u)$
    for $\mathsf{ind} \in [q_{\mathsf{M}}]$
        $(\overline{prepk}^{\mathsf{ind}}_u, \overline{prek}^{\mathsf{ind}}_u) \xleftarrow{\$} \Sigma.\mathsf{PreKGen}()$
        $\mathcal{L}^{\overline{prek}}_{u} \xleftarrow{+} \overline{prek}^{\mathsf{ind}}_u$, $\mathcal{L}^{\overline{prepk}}_{\mathsf{all}} \xleftarrow{+} \overline{prepk}^{\mathsf{ind}}_u$
        $\mathcal{L}_{\mathsf{all}} \xleftarrow{+} (\overline{prepk}_u, \overline{prek}_u)$
$\mathsf{b} \xleftarrow{\$} \{0,1\}$
$\mathsf{b}' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\mathcal{L}_{\mathsf{all}})$
return $[\![\mathsf{b} = \mathsf{b}']\!]$

$\mathsf{Session\text{-}Start}(\mathsf{sid}, \mathsf{rid}, \mathsf{ind})$:
if $\mathsf{b} = 0$
    $\pi_{\mathsf{rid}}.\mathsf{role} \leftarrow \mathtt{resp}$, $\pi_{\mathsf{rid}}.\mathsf{st}_{\mathsf{exec}} \leftarrow \mathsf{running}$
    $\pi_{\mathsf{sid}}.\mathsf{role} \leftarrow \mathtt{init}$, $\pi_{\mathsf{sid}}.\mathsf{st}_{\mathsf{exec}} \leftarrow \mathsf{running}$
    $(\pi'_{\mathsf{rid}}, m) \xleftarrow{\$} \Sigma.\mathsf{Run}(\overline{ik}_{\mathsf{rid}}, \mathcal{L}^{\overline{prek}}_{\mathsf{rid}}, \mathcal{L}^{\overline{ipk}}_{\mathsf{all}}, \mathcal{L}^{\overline{prepk}}_{\mathsf{all}}, \pi_{\mathsf{rid}}, (\mathsf{create}, \mathsf{ind}))$
    $(\pi'_{\mathsf{sid}}, m') \xleftarrow{\$} \Sigma.\mathsf{Run}(\overline{ik}_{\mathsf{sid}}, \mathcal{L}^{\overline{prek}}_{\mathsf{sid}}, \mathcal{L}^{\overline{ipk}}_{\mathsf{all}}, \mathcal{L}^{\overline{prepk}}_{\mathsf{all}}, \pi_{\mathsf{sid}}, m)$
    $(K, T) \xleftarrow{\$} (\pi'_{\mathsf{sid}}.K, (m, m'))$
else
    $(K, T) \xleftarrow{\$} \Sigma.\mathsf{Fake}(\overline{ipk}_{\mathsf{sid}}, \overline{ik}_{\mathsf{rid}}, \mathcal{L}^{\overline{prek}}_{\mathsf{rid}}, \mathsf{ind})$
return $(K, T)$
```

Figure 7.2: The offline deniability experiment for an attacker $\mathcal{A}$ against a DAKE scheme Σ. The oracle $\mathcal{O} := \{\mathsf{Session\text{-}Start}\}$.

- $\mathsf{role} \in \{\mathtt{init}, \mathtt{resp}\}$: the role of the party. The initiator $\mathtt{init}$ and the responder $\mathtt{resp}$ indicate the message sender and receiver in the DAKE, respectively.
- $\mathsf{st}_{\mathsf{exec}} \in \{\perp, \mathsf{running}, \mathsf{accepted}, \mathsf{reject}\}$: The status of this session's execution. The status is initialized with $\perp$ and turns to $\mathsf{running}$ when the session starts. The status is set to accept if the DAKE is executed without errors and reject otherwise.

7.3.2 The game-based offline deniability experiment

The game-based offline deniability experiment $\mathrm{Expr}^{\mathsf{deni}}_{\Sigma,q_{\mathsf{P}},q_{\mathsf{M}},q_{\mathsf{S}}}(\mathcal{A})$ for a DAKE protocol Σ is depicted in Figure 7.2, where q_{P}, q_{M}, and q_{S} respectively denotes the maximal number of parties, of (medium-term) pre-keys per party, and of total sessions. At the start of this experiment, long-term identity and medium-term pre- public/private key pairs are generated for all q_{P} honest parties and provided to the attacker[1]. A random challenge bit b is fixed for the duration of the experiment. The attacker is given repeated access to a $\mathsf{Session\text{-}Start}$ oracle which takes as input two party identifiers sid and rid and a pre-key index ind. If b is 0, then the $\mathsf{Session\text{-}Start}$ oracle will generate an honest transcript of an interaction between sid and rid using the $\Sigma.\mathsf{Run}$ algorithm and each party's secret keys. If b is 1, then the $\mathsf{Session\text{-}Start}$ oracle will generate a simulated transcript of an interaction between sid and rid using the $\Sigma.\mathsf{Fake}$ algorithm. At the end of the experiment, the attacker outputs a guess b' of b. The experiment outputs 1 if $\mathsf{b}' = \mathsf{b}$ and 0 otherwise. The attacker's advantage in the deniability game is the absolute value of the difference between $\frac{1}{2}$ and the probability the experiment outputs 1.

[1]The attacker here can be considered as a judge in reality.

Definition 61. *An asynchronous* DAKE *protocol* Σ *is* $(t, \epsilon, q_{\mathsf{S}})$*-deniable (with respect to maximal number of parties* q_{P} *and pre-keys per party* q_{M}*) if for any attacker* $\mathcal{A}$ *with running time at most* t *and making at most* q_{S} *many queries (to its* Session-Start *oracle), we have that*

$$\mathsf{Adv}^{\mathsf{deni}}_{\Sigma}(\mathcal{A}) := \left|\Pr[\mathrm{Expr}^{\mathsf{deni}}_{\Sigma,q_{\mathsf{P}},q_{\mathsf{M}},q_{\mathsf{S}}}(\mathcal{A}) = 1] - \frac{1}{2}\right| \leq \epsilon$$

where $\mathrm{Expr}^{\mathsf{deni}}_{\Sigma,q_{\mathsf{P}},q_{\mathsf{M}},q_{\mathsf{S}}}(\mathcal{A})$ *is defined in Figure 7.2.*

7.4 Extended Secure Messaging

In this section, we first define our new *extended secure messaging* (eSM) scheme in Section 7.4.1, followed by the expected security properties in Section 7.4.2. Then, we define an associated strong security model (eSM) in Section 7.4.3. Finally in Section 7.4.4, we modularize the eSM security into three simplified security models, which can ease our final security proof.

7.4.1 Syntax

Definition 62. *Let* $\mathcal{ISS}$ *denote the space of the initial shared secrets between two parties. An* extended secure messaging *(*eSM*) scheme consists of six algorithms* eSM = (IdKGen, PreKGen, eInit-A, eInit-B, eSend, eRcv)*, where*

- $(ipk, ik) \xleftarrow{\$} \mathsf{IdKGen}()$ *outputs an long-term identity public-private key pair,*
- $(prepk, prek) \xleftarrow{\$} \mathsf{PreKGen}()$ *outputs a medium-term public-private pre-key pair,*
- $\mathsf{st}_{\mathtt{A}} \leftarrow \mathsf{eInit\text{-}A}(iss)$ *(resp.* $\mathsf{st}_{\mathtt{B}} \leftarrow \mathsf{eInit\text{-}B}(iss)$*) inputs an initial shared secret* $iss \in \mathcal{ISS}$ *and outputs a session state,*
- $(\mathsf{st}', c) \xleftarrow{\$} \mathsf{eSend}(\mathsf{st}, ipk, prepk, m)$ *inputs a state* st*, a long-term identity public key* ipk*, a medium-term public prekey* $prepk$*, and a message* m*, and outputs a new state and a ciphertext, and*
- $(\mathsf{st}', t, i, m) \leftarrow \mathsf{eRcv}(\mathsf{st}, ik, prek, c)$ *inputs a state* st*, a long-term identity private key* ik*, a medium-term private key* $prek$*, and a ciphertext* c*, and outputs a new state, an epoch number, a message index, and a message.*

Our eSM re-uses two important concepts *epoch* and *message index* that originate in [12].

Epoch. The epoch t is used to describe how many back-and-forth interactions in a two-party communication channel (aka. session) have been processed. Let $t_{\mathtt{A}}$ and $t_{\mathtt{B}}$ respectively denote the epoch counters of parties $\mathtt{A}$ and $\mathtt{B}$ in a session. Both epoch counters start from 0. If either party $\mathtt{P} \in \{\mathtt{A}, \mathtt{B}\}$ switches the actions, i.e., from sending to receiving or from receiving to sending messages, then the counter $t_{\mathtt{P}}$ is incremented by 1. In this paper, we use even epochs ($t_{\mathtt{A}}, t_{\mathtt{B}} = 0, 2, 4, ...$) to denote the scenario where B acts as the sender and A acts as the receiver, and odd epochs in reverse. In each epoch, the sender can send arbitrarily many messages in a sequence. The difference between the two counters $t_{\mathtt{A}}$ and $t_{\mathtt{B}}$ is never greater than 1, i.e., $|t_{\mathtt{A}} - t_{\mathtt{B}}| \leq 1$.

Message Indices. The message index i identifies the index of a message in each epoch. Notably, the epoch number t and message index i output by eRcv indicate the position of the decrypted message m during the communication. The receiver is expected to recover the position of each decrypted message even if it is delivered out of order.

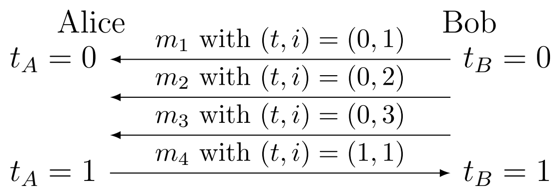

Alice
Bob
$t_A = 0$
$t_B = 0$
m_1 with $(t, i) = (0, 1)$
m_2 with $(t, i) = (0, 2)$
m_3 with $(t, i) = (0, 3)$
m_4 with $(t, i) = (1, 1)$
$t_A = 1$
$t_B = 1$

Figure 7.3: An example session between Alice and Bob. The session starts with $t_{\mathtt{A}} = t_{\mathtt{B}} = 0$, i.e., Bob is the sender. When Bob continuously sends messages, the message index grows from 1 for m_1 to 3 for m_3. When Alice switches the role from receiver to sender, the epoch increases to $t_{\mathtt{A}} = t_{\mathtt{B}} = 1$.

7.4.2 Strong Security Properties

The eSM schemes aim at following strong security properties. First, we expect our eSM to meet well-studied basic properties below:

1. **Correctness:** The messages exchanged between two parties are recovered in the correct order, if no attacker manipulates the underlying transmissions.
2. **Immediate decryption (ID) and message-loss resilience (MLR):** Messages must be decrypted to the correct position as soon as they arrive; the loss of some messages does not prevent subsequent interaction.
3. **Forward secrecy (FS)**: All messages that have been sent and received prior to a session state compromise of either party (or both) remain secure to an attacker.
4. **Post-compromise security (PCS):** The parties can recover from session state compromise (assuming the access to fresh randomness) when the attacker is passive.

Second, our eSM targets the following advanced security against fine-grained compromise.

5. ***Strong*** **authenticity:** The attacker cannot modify the messages in transmission or inject new ones, unless the sender's session state is compromised.
6. ***Strong*** **privacy**: If both parties' states are uncompromised, the attacker obtains no information about the messages sent. Assuming both parties have access to fresh randomness, strong privacy also holds unless the receiver's session state, private identity key, and corresponding private pre-key all are compromised.

7. **Randomness leakage/failures**: While both parties' session states are uncompromised, all above security properties (in particular, including strong authenticity and strong privacy) except PCS hold even if the attacker completely controls the parties' local randomness. That is, good randomness is only required for PCS.

Finally, our eSM also pursues two new security properties:

8. ***State compromise/failures*:** While the sender's randomness quality is good and the receiver's private identity key or pre-key is not leaked, the privacy of the messages holds even if both parties' session states are corrupted.
9. ***Periodic privacy recovery*** **(PPR):** If the attacker is passive (i.e., does not inject corrupted messages), the message privacy recovers from the compromise of both parties' all private information after a time period (assuming each has access to fresh randomness).

We stress that the first new property *state compromise/failures* has a particular impact for the secure messaging after an *insecure* key establishment. For instance, consider that the party B initializes a conversation with A using X3DH in Signal. The leakage of the sender B's private identity key and ephemeral randomness in X3DH implies the compromise of the initial shared secret and further both parties' session states in DR. If B continuously sends messages to A without receiving a reply in Signal, all messages in the sequence are leaked, since the attacker can use A's session state to decrypt the ciphertexts. An eSM protocol with the "state compromise/failures" property is able to prevent such attack.

Moreover, the second new property PPR complements the strong privacy. Assuming the secure randomness, the strong privacy ensures the secrecy of *past* messages if the corresponding private pre-keys are not leaked, while PPR ensures the secrecy of *future* messages if new pre-key pairs are randomly sampled and honestly delivered to the partner.

Remark 2. *The relation between* PPR *and* PCS *depends on what we take as the reference point for* PCS*. The term "Post Compromise Security" was introduced in 2016 in [73], which defines both a broader informal security guarantee as well as a specific instantiation.* PPR *can be seen as a subclass of the general initial* PCS *notion from [73].*

Over time, follow-up works have developed more fine-grained notions of PCS*, notably instantiated for specific protocol classes. One such example is [12], whose target protocol class closely matches ours. Compared to the* PCS *instantiation in [12],* PPR *can be regarded as an orthogonal class of privacy that is related to time (aka. temporal privacy). Although both the* PCS *instance from [12] and* PPR *provide healing after compromise and might look similar, they differ in the following three aspects.*

1. ***Different Healing Objects**: While the* PCS *instance from [12] heals the session state (e.g., encryption/decryption keys), and might further impact on other security guarantees*

(e.g. privacy, authenticity, etc.), PPR *heals the (strong) privacy, which is a concrete security guarantee.*

2. ***Different Healing Approaches**: The* PCS *instance from [12] holds only when the session states are healed. Note that (strong) privacy is expected to hold "unless the receiver's session state, private identity key, and corresponding private pre-key all are compromised". Thus,* PPR *might hold when some private materials other than session states are recovered, i.e., is independent of their instance of* PCS*.*

7.4.3 Security Model

The *Extended Secure Messaging* (eSM) security game $\mathrm{Expr}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ for an eSM scheme Π with respect to a parameter $\triangle_{\mathsf{eSM}}$ is depicted in Figure 7.4.

Notation. Our model considers the communication between two distinct parties $\mathtt{A}$ and $\mathtt{B}$. For a party $\mathtt{P} \in \{\mathtt{A}, \mathtt{B}\}$, we use $\neg\mathtt{P}$ to denote the partner, i.e., $\{\mathtt{P}, \neg\mathtt{P}\} = \{\mathtt{A}, \mathtt{B}\}$. For an element x and a set X, we write $X \overset{+}{\leftarrow} x$ for adding x in X, i.e., $X \overset{+}{\leftarrow} x \Leftrightarrow X \leftarrow X \cup \{x\}$. Similarly, we write $X \overset{-}{\leftarrow} x$ for removing x from X, i.e., $X \overset{-}{\leftarrow} x \Leftrightarrow X \leftarrow X \setminus \{x\}$. For a set of tuples X and a variable y, we use $X(y)$ to denote the subset of X, where each tuple x includes y, i.e., $X(y) = \{x \in X \mid y \in x\}$. We say $y \in X$ if there exists a tuple $x \in X$ such that $y \in x$, i.e., $y \in X \Leftrightarrow X(y) \neq \emptyset$.

Trust Model: We assume an *authenticated* channel between each party and the server for key-update and -fetch and therefore no forgery of the public identity keys and pre-keys. This is the common treatment in the security analyses in this domain, e.g. [75], the server is considered to be a bulletin board, where each party can upload their own and fetch other parties' honest public keys. For practical deployments, we require that the key-upload and key-fetch processes between each party and sever use fixed bandwidth and are only executed periodically. We omit the discussion on the frequency of the pre-keys' upload and retrieve[2].

We assume that all session-specific data is stored at the same security level in the state, but the non-session-specific data that can be potentially shared among multiple sessions (i.e., identity keys and pre-keys) might be stored differently. Thus, corruption of session-specific state does not imply leakage of the private identity key and pre-key and vice versa. In fact, as we will show later, an eSM scheme can achieve additional privacy guarantees if the private identity keys (or pre-keys) can be stored in the secure environment on the device, such as a Hardware Security Module (HSM).

Moreover, we also require the eSM scheme Π to be *natural*, which is first defined for SM in [12, Definition 7].

[2]As an example, we can consider a scenario where every party is only allowed to upload and fetch public keys at 12am every day.

Definition 63. *We say an* eSM *scheme is* natural*, if the following holds:*

1. *the receiver state remains unchanged, if the message output by* eRcv *is* $m = \perp$*,*
2. *the values* (t, i) *output by* eRcv *can be efficiently computed from* c*,*
3. *if* eRcv *has already accepted an ciphertext corresponding to the position* (t, i)*, the next ciphertext corresponding to the same position must be rejected,*
4. *a party always rejects ciphertexts corresponding to an epoch in which the party does not act as receiver, and*
5. *if a party* P *accepts a ciphertext corresponding to an epoch* t*, then* $t \leq t_{\mathsf{P}} + 1$*.*

Experiment Variables and Predicates. The security experiment $\mathrm{Expr}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ includes the following global variables:

- $\mathsf{safe}^{\mathsf{idK}}_{\mathtt{A}}, \mathsf{safe}^{\mathsf{idK}}_{\mathtt{B}} \in \{\mathsf{true}, \mathsf{false}\}$: the boolean values indicating whether the private identity keys are revealed.
- $\mathcal{L}^{\mathsf{rev}}_{\mathtt{A}}, \mathcal{L}^{\mathsf{rev}}_{\mathtt{B}}$: the lists that record the indices of the pre-keys that are revealed.
- $\mathcal{L}^{\mathsf{cor}}_{\mathtt{A}}, \mathcal{L}^{\mathsf{cor}}_{\mathtt{B}}$: the lists that record the indices of the epochs where the session states are corrupted.
- $n_{\mathtt{A}}, n_{\mathtt{B}}$: the pre-key counters.
- $t_{\mathtt{A}}, t_{\mathtt{B}}$: the epoch counters.
- $i_{\mathtt{A}}, i_{\mathtt{B}}$: the message index counters.
- trans: a set that records all ciphertexts, which are honestly encrypted but undelivered yet, and their related information. See the helper function record for more details.
- $\mathsf{allTrans}$: a set that records all honest encrypted ciphertexts (including both the delivered and undelivered ones), and their related information.
- chall: a set that records all challenge ciphertexts, which are honestly encrypted but undelivered yet, and their related information.
- $\mathsf{allChall}$: a set that records all challenge ciphertexts (including both the delivered and undelivered ones), and their related information.
- comp: a set that records all compromised ciphertexts, which are honestly encrypted but not delivered yet, and their related information. A compromised ciphertext means that the attacker can trivially forge a new ciphertext at the same position.
- $\mathsf{win}^{\mathsf{corr}}, \mathsf{win}^{\mathsf{auth}}, \mathsf{win}^{\mathsf{priv}} \in \{\mathsf{true}, \mathsf{false}\}$: the winning predicate that indicates whether the attacker wins.
- $\mathsf{b} \in \{0, 1\}$: the challenge bit.

```
$\mathrm{Expr}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A})$:
    $\mathsf{safe}^{\mathsf{idK}}_{\mathtt{A}}, \mathsf{safe}^{\mathsf{idK}}_{\mathtt{B}}, \mathcal{L}^{\mathsf{rev}}_{\mathtt{A}}, \mathcal{L}^{\mathsf{rev}}_{\mathtt{B}}, \mathcal{L}^{\mathsf{cor}}_{\mathtt{A}}, \mathcal{L}^{\mathsf{cor}}_{\mathtt{B}} \leftarrow \bot$
    $(n_{\mathtt{A}}, n_{\mathtt{B}}) \leftarrow (0,0)$
    $() \leftarrow \mathcal{A}^{\mathcal{O}_1}()$
    require $\bot \notin \{\mathsf{safe}^{\mathsf{idK}}_{\mathtt{A}}, \mathsf{safe}^{\mathsf{idK}}_{\mathtt{B}}\}$
    require $n_{\mathtt{A}}, n_{\mathtt{B}} \geq 1$
    $iss \xleftarrow{\$} \mathcal{ISS}$, $(t_{\mathtt{A}}, t_{\mathtt{B}}), (i_{\mathtt{A}}, i_{\mathtt{B}}) \leftarrow (0,0)$
    $\mathsf{st}_{\mathtt{A}} \leftarrow \mathsf{eInit\text{-}A}(iss)$, $\mathsf{st}_{\mathtt{B}} \leftarrow \mathsf{eInit\text{-}B}(iss)$
    $\mathsf{trans}, \mathsf{chall}, \mathsf{comp}, \mathsf{allChall}, \mathsf{allTrans} \leftarrow \emptyset$
    $\mathsf{win}^{\mathsf{corr}}, \mathsf{win}^{\mathsf{auth}} \leftarrow \mathsf{false}$
    $\mathsf{b} \xleftarrow{\$} \{0,1\}$, $\mathsf{b}' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}2}()$
    $\mathsf{win}^{\mathsf{priv}} \leftarrow [\![\mathsf{b} = \mathsf{b}']\!]$
    return $(\mathsf{win}^{\mathsf{corr}}, \mathsf{win}^{\mathsf{auth}}, \mathsf{win}^{\mathsf{priv}})$

$\mathcal{O}_{\mathsf{NewIdKey\text{-}A}}(r)$:
    require $\mathsf{safe}^{\mathsf{idK}}_{\mathtt{A}} = \bot$
    $(r, \mathsf{flag}) \xleftarrow{\$} \mathsf{sam\text{-}if\text{-}nec}(r)$
    $(ipk_{\mathtt{A}}, ik_{\mathtt{A}}) \xleftarrow{\$} \mathsf{IdKGen}(r)$
    $\mathsf{safe}^{\mathsf{idK}}_{\mathtt{A}} \leftarrow [\![\mathsf{flag} = \mathsf{good}]\!]$
    return $ipk_{\mathtt{A}}$

$\mathcal{O}_{\mathsf{RevIdKey\text{-}A}}$:
    $\mathsf{safe}^{\mathsf{idK}}_{\mathtt{A}} \leftarrow \mathsf{false}$, $\mathsf{corruption\text{-}update}()$
    foreach $(\mathsf{P}, \mathsf{ind}, \mathsf{flag}, t, i, m, c) \in \mathsf{allChall}$
        require $\mathsf{safe\text{-}ch}_{\mathsf{P}}(\mathsf{flag}, t, \mathsf{ind})$
    foreach $(\mathsf{P}, t) \in \mathsf{trans}$ and $\neg\mathsf{safe\text{-}inj}_{\neg\mathsf{P}}(t)$
        $\mathsf{comp} \xleftarrow{+} \mathsf{trans}(\mathsf{P}, t)$
    return $ik_{\mathtt{A}}$

$\mathcal{O}_{\mathsf{NewPreKey\text{-}A}}(r)$:
    $n_{\mathtt{A}}{+}{+}$
    $(r, \mathsf{flag}) \xleftarrow{\$} \mathsf{sam\text{-}if\text{-}nec}(r)$
    $(prepk^{n_{\mathtt{A}}}_{\mathtt{A}}, prek^{n_{\mathtt{A}}}_{\mathtt{A}}) \xleftarrow{\$} \mathsf{PreKGen}(r)$
    if $\mathsf{flag} = \mathsf{bad}$ : $\mathcal{L}^{\mathsf{rev}}_{\mathtt{A}} \xleftarrow{+} n_{\mathtt{A}}$
    return $prepk_{\mathtt{A}}$

$\mathcal{O}_{\mathsf{RevPreKey\text{-}A}}(n)$:
    require $n \leq n_{\mathtt{A}}$
    $\mathcal{L}^{\mathsf{rev}}_{\mathtt{A}} \xleftarrow{+} n$, $\mathsf{corruption\text{-}update}()$
    foreach $(\mathsf{P}, \mathsf{ind}, \mathsf{flag}, t, i, m, c) \in \mathsf{allChall}$
        require $\mathsf{safe\text{-}ch}_{\mathsf{P}}(\mathsf{flag}, t, \mathsf{ind})$
    foreach $(\mathsf{P}, t) \in \mathsf{trans}$ and $\neg\mathsf{safe\text{-}inj}_{\neg\mathsf{P}}(t)$
        $\mathsf{comp} \xleftarrow{+} \mathsf{trans}(\mathsf{P}, t)$
    return $prek^{n}_{\mathtt{A}}$

$\mathcal{O}_{\mathsf{Transmit\text{-}A}}(\mathsf{ind}, m, r)$:
    require $\mathsf{ind} \leq n_{\mathtt{B}}$
    $(r, \mathsf{flag}) \xleftarrow{\$} \mathsf{sam\text{-}if\text{-}nec}(r)$
    $\mathsf{ep\text{-}mgmt}(\mathtt{A}, \mathsf{flag}, \mathsf{ind})$
    $i_{\mathtt{A}}{+}{+}$
    $(\mathsf{st}_{\mathtt{A}}, c) \xleftarrow{\$} \mathsf{eSend}(\mathsf{st}_{\mathtt{A}}, ipk_{\mathtt{B}}, prepk^{\mathsf{ind}}_{\mathtt{B}}, m; r)$
    $\mathsf{record}(\mathtt{A}, \mathsf{norm}, \mathsf{flag}, \mathsf{ind}, m, c)$
    return $c$

$\mathcal{O}_{\mathsf{Deliver\text{-}A}}(c)$:
    require $(\mathtt{B}, \mathsf{ind}, t, i, m, c) \in \mathsf{trans}$ for some $\mathsf{ind}, t, i, m$
    $(\mathsf{st}_{\mathtt{A}}, t', i', m') \leftarrow \mathsf{eRcv}(\mathsf{st}_{\mathtt{A}}, ik_{\mathtt{A}}, prek^{\mathsf{ind}}_{\mathtt{A}}, c)$
    if $(t', i', m') \neq (t, i, m)$: $\mathsf{win}^{\mathsf{corr}} \leftarrow \mathsf{true}$
    if $(t, i, m) \in \mathsf{chall}$: $m' \leftarrow \bot$
    $t_{\mathtt{A}} \leftarrow \max(t_{\mathtt{A}}, t')$, $\mathsf{delete}(t, i)$
    return $(t', i', m')$

$\mathcal{O}_{\mathsf{Inject\text{-}A}}(\mathsf{ind}, c)$:
    require $(\mathtt{B}, c) \notin \mathsf{trans}$ and $\mathsf{ind} \leq n_{\mathtt{A}}$
    require $\mathsf{safe\text{-}inj}_{\mathtt{A}}(t_{\mathtt{B}})$ and $\mathsf{safe\text{-}inj}_{\mathtt{A}}(t_{\mathtt{A}})$
    $(\mathsf{st}_{\mathtt{A}}, t', i', m') \leftarrow \mathsf{eRcv}(\mathsf{st}_{\mathtt{A}}, ik_{\mathtt{A}}, prek^{\mathsf{ind}}_{\mathtt{A}}, c)$
    if $m' \neq \bot$ and $(\mathtt{B}, t', i') \notin \mathsf{comp}$ : $\mathsf{win}^{\mathsf{auth}} \leftarrow \mathsf{true}$
    $t_{\mathtt{A}} \leftarrow \max(t_{\mathtt{A}}, t')$, $\mathsf{delete}(t', i')$
    return $(t', i', m')$

$\mathcal{O}_{\mathsf{Challenge\text{-}A}}(\mathsf{ind}, m_0, m_1, r)$:
    require $\mathsf{ind} \leq n_{\mathtt{B}}$
    $(r, \mathsf{flag}) \xleftarrow{\$} \mathsf{sam\text{-}if\text{-}nec}(r)$
    $\mathsf{ep\text{-}mgmt}(\mathtt{A}, \mathsf{flag}, \mathsf{ind})$
    require $\mathsf{safe\text{-}ch}_{\mathtt{A}}(\mathsf{flag}, t_{\mathtt{A}}, \mathsf{ind})$ and $|m_0| = |m_1|$
    $i_{\mathtt{A}}{+}{+}$
    $(\mathsf{st}_{\mathtt{A}}, c) \xleftarrow{\$} \mathsf{eSend}(\mathsf{st}_{\mathtt{A}}, ipk_{\mathtt{B}}, prepk^{\mathsf{ind}}_{\mathtt{B}}, m_{\mathsf{b}}; r)$
    $\mathsf{record}(\mathtt{A}, \mathsf{chall}, \mathsf{flag}, \mathsf{ind}, m_{\mathsf{b}}, c)$
    return $c$

$\mathcal{O}_{\mathsf{Corrupt\text{-}A}}$:
    $\mathcal{L}^{\mathsf{cor}}_{\mathtt{A}} \xleftarrow{+} t_{\mathtt{A}}$, $\mathsf{corruption\text{-}update}()$
    require $(\mathtt{B}, \mathsf{ind}, \mathsf{flag}) \notin \mathsf{chall}$ or $\left(\mathsf{flag} = \mathsf{good}\right.$ and $\left.\mathsf{safe}^{\mathsf{idK}}_{\mathtt{A}}\right)$ or $\left(\mathsf{flag} = \mathsf{good}\right.$ and $\left.\mathsf{safe}^{\mathsf{preK}}_{\mathtt{A}}(\mathsf{ind})\right)$
    foreach $(\mathtt{B}, t) \in \mathsf{trans}$ and $\neg\mathsf{safe\text{-}st}_{\mathtt{B}}(t)$
        $\mathsf{comp} \xleftarrow{+} \mathsf{trans}(\mathtt{B}, t)$
    foreach $(\mathtt{A}, t_{\mathtt{A}}) \in \mathsf{trans}$ and $\neg\mathsf{safe\text{-}st}_{\mathtt{B}}(t_{\mathtt{B}})$
        $\mathsf{comp} \xleftarrow{+} \mathsf{trans}(\mathtt{A}, t_{\mathtt{A}})$
    return $\mathsf{st}_{\mathtt{A}}$
```

Figure 7.4: The extended secure messaging experiment $\mathrm{Expr}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ for an eSM scheme Π with respect to a parameter $\triangle_{\mathsf{eSM}}$. $\mathcal{O}_1 := \{\mathcal{O}_{\mathsf{NewIdKey\text{-}A}}, \mathcal{O}_{\mathsf{NewIdKey\text{-}B}}, \mathcal{O}_{\mathsf{NewPreKey\text{-}A}}, \mathcal{O}_{\mathsf{NewPreKey\text{-}B}}\}$ and $\mathcal{O}2$ denotes all oracles. This figure only depicts the oracles for $\mathtt{A}$ (ending with -A). The oracles for $\mathtt{B}$ are defined analogously. We highlight the difference to the SM-security game for a SM scheme in [12] with blue color. We give more helper functions and safe predicates in Figure 7.5.

Moreover, the experiment $\mathrm{Expr}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ also includes four predicates as shown in Figure 7.5.

- $\mathsf{safe}^{\mathsf{preK}}_{\mathtt{P}}(\mathsf{ind})$: indicating whether ind-th pre-key of party $\mathtt{P}$ is leaked. We define it true if ind is not included in $\mathcal{L}^{\mathsf{rev}}_{\mathtt{P}}$.
- $\mathsf{safe\text{-}st}_{\mathtt{P}}(t)$: indicating whether the state of party $\mathtt{P}$ at epoch t is expected to be safe. This predicate simplifies the definition of $\mathsf{safe\text{-}ch}_{\mathtt{P}}$ and $\mathsf{safe\text{-}inj}_{\mathtt{P}}$ predicates below. We define it true if none of epochs from t to $(t - \triangle_{\mathsf{eSM}} + 1)$ is included in the list $\mathcal{L}^{\mathsf{cor}}_{\mathtt{P}}$.
- $\mathsf{safe\text{-}ch}_{\mathtt{P}}(\mathsf{flag}, t, \mathsf{ind})$: indicating whether the privacy of the message sent by $\mathtt{P}$ is expected to hold, under the randomness quality $\mathsf{flag} \in \{\mathsf{good}, \mathsf{bad}\}$, the sending epoch t, and the receiver $\neg\mathtt{P}$'s pre-key index ind. We define it to be true if any of the following conditions hold:

  1. both parties' states are safe at epoch t,
  2. the partner $\neg\mathtt{P}$'s state is safe and the randomness quality is $\mathsf{flag} = \mathsf{good}$,
  3. the partner $\neg\mathtt{P}$'s identity key is safe and the randomness quality is $\mathsf{flag} = \mathsf{good}$, or
  4. the partner $\neg\mathtt{P}$'s ind-th pre-key is safe and the randomness quality is $\mathsf{flag} = \mathsf{good}$.

- $\mathsf{safe\text{-}inj}_{\mathtt{P}}(t)$: indicating whether the authenticity at the party $\mathtt{P}$'s epoch t (i.e., $\mathtt{P}$ is expected not to accept a forged ciphertext corresponding to epoch t) holds. We define it to be true if the partner's state is safe at epoch t.

Helper Functions. To simplify the security experiment definition, we use five helper functions as shown in Figure 7.5.

- $\mathsf{sam\text{-}if\text{-}nec}(r)$: If $r \neq \perp$, this function outputs (r, bad) indicating that the randomness is attacker-controlled. Otherwise, a new random string r is sampled from the space $\mathcal{R}$[3] and is output together with a flag good.
- $\mathsf{record}(\mathtt{P}, \mathsf{type}, \mathsf{flag}, \mathsf{ind}, m, c)$: A record rec, which includes the party's identity $\mathtt{P}$, the partner's pre-key index ind, the randomness flag flag, the epoch counter $t_{\mathtt{P}}$, the message index counter $i_{\mathtt{P}}$, the message m, and the ciphertext c, is added into the transcript sets trans and $\mathsf{allTrans}$. If the $\mathsf{safe\text{-}inj}_{\mathtt{P}}(t_{\mathtt{P}})$ predicate is false, then this record is also added into the compromise set comp. If c is a challenge ciphertext, indicated by whether $\mathsf{type} = \mathsf{chall}$, the record rec is also added into the challenge sets chall and $\mathsf{allChall}$.
- $\mathsf{ep\text{-}mgmt}(\mathtt{P}, \mathsf{flag}, \mathsf{ind})$: When the party $\mathtt{P}$ enters a new epoch as the sender upon the partner's ind-th pre-key, the new epoch number is added to the state corruption list $\mathcal{L}^{\mathsf{cor}}_{\mathtt{P}}$ if the safe challenge predicate is false. Then, the epoch counter $t_{\mathtt{P}}$ is incremented by 1 and the message index counter i is set to 0.

[3]The randomness space $\mathcal{R}$ is not specific and depends on the concrete functions and algorithms. Here, we use $\mathcal{R}$ only for simplicity.

```
$\mathsf{sam\text{-}if\text{-}nec}(r)$:
    $\mathsf{flag} \leftarrow \mathsf{bad}$
    if $r = \bot$
        $r \xleftarrow{\$} \mathcal{R}$, $\mathsf{flag} \leftarrow \mathsf{good}$
    return $(r, \mathsf{flag})$

$\mathsf{ep\text{-}mgmt}(\mathtt{P}, \mathsf{flag}, \mathsf{ind})$:
    if $(\mathtt{P} = \mathtt{A}$ and $t_\mathtt{P}$ even$)$ or $(\mathtt{P} = \mathtt{B}$ and $t_\mathtt{P}$ odd$)$
        if $\neg\mathsf{safe\text{-}ch}_\mathtt{P}(\mathsf{flag}, t_\mathtt{P}, \mathsf{ind})$
            $\mathcal{L}^{\mathsf{cor}}_\mathtt{P} \xleftarrow{+} t_\mathtt{P} + 1$
        $t_\mathtt{P}{+}{+}$, $i_\mathtt{P} \leftarrow 0$

$\mathsf{delete}(t, i)$:
    $\mathsf{rec} \leftarrow (\mathtt{P}, \mathsf{ind}, \mathsf{flag}, t, i, m, c)$ for some $\mathtt{P}, \mathsf{ind}, \mathsf{flag}, m, c$
    $\mathsf{trans}, \mathsf{chall}, \mathsf{comp} \xleftarrow{-} \mathsf{rec}$

$\mathsf{record}(\mathtt{P}, \mathsf{type}, \mathsf{flag}, \mathsf{ind}, m, c)$:
    $\mathsf{rec} \leftarrow (\mathtt{P}, \mathsf{ind}, \mathsf{flag}, t_\mathtt{P}, i_\mathtt{P}, m, c)$
    $\mathsf{allTrans}, \mathsf{trans} \xleftarrow{+} \mathsf{rec}$
    if $\neg\mathsf{safe\text{-}inj}_{\neg\mathtt{P}}(t_\mathtt{P})$: $\mathsf{comp} \xleftarrow{+} \mathsf{rec}$
    if $\mathsf{type} = \mathsf{chall}$: $\mathsf{allChall}, \mathsf{chall} \xleftarrow{+} \mathsf{rec}$

$\mathsf{corruption\text{-}update}()$:
    foreach $(\mathtt{P}, \mathsf{ind}, \mathsf{flag}, t, 1, m, c) \in \mathsf{allTrans}$
        if $\neg\mathsf{safe\text{-}ch}_\mathtt{P}(\mathsf{flag}, (t-1), \mathsf{ind})$
            $\mathcal{L}^{\mathsf{cor}}_\mathtt{P} \xleftarrow{+} t$
```

$\mathsf{safe}^{\mathsf{preK}}_\mathtt{P}(\mathsf{ind}) \Leftrightarrow \mathsf{ind} \notin \mathcal{L}^{\mathsf{rev}}_\mathtt{P}$

$\mathsf{safe\text{-}st}_\mathtt{P}(t) \Leftrightarrow t, (t-1), \ldots, (t - \triangle_{\mathsf{eSM}} + 1) \notin \mathcal{L}^{\mathsf{cor}}_\mathtt{P}$

$\mathsf{safe\text{-}ch}_\mathtt{P}(\mathsf{flag}, t, \mathsf{ind}) \Leftrightarrow \Big(\mathsf{safe\text{-}st}_\mathtt{P}(t) \textbf{ and } \mathsf{safe\text{-}st}_{\neg\mathtt{P}}(t)\Big) \textbf{ or } \Big(\mathsf{flag} = \mathsf{good} \textbf{ and } \mathsf{safe\text{-}st}_{\neg\mathtt{P}}(t)\Big) \textbf{ or } \Big(\mathsf{flag} = \mathsf{good} \textbf{ and } \mathsf{safe}^{\mathsf{idK}}_{\neg\mathtt{P}}\Big) \textbf{ or } \Big(\mathsf{flag} = \mathsf{good} \textbf{ and } \mathsf{safe}^{\mathsf{preK}}_{\neg\mathtt{P}}(\mathsf{ind})\Big)$

$\mathsf{safe\text{-}inj}_\mathtt{P}(t) \Leftrightarrow \mathsf{safe\text{-}st}_{\neg\mathtt{P}}(t)$

Figure 7.5: The helping functions in extended secure messaging experiment $\mathsf{Expr}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ for an eSM scheme Π with respect to a parameter $\triangle_{\mathsf{eSM}}$. We highlight the difference to the SM-security game for a SM scheme in [12] with blue color.

- $\mathsf{delete}(t, i)$: deletes all records that includes (t, i) from the sets trans, chall, and comp.
- $\mathsf{corruption\text{-}update}()$: checks all records in the $\mathsf{allTrans}$ list whether the safe challenge predicates for the first messages in each epoch (still) hold or not. If it does not hold, then adds the epoch into the corruption list.

Notably, the helper function $\mathsf{corruption\text{-}update}$ is invoked in the key-revealing and state-corruption oracles to capture the impact of the leakage of any secret on the secrecy of the (past) session states.

Experiment Execution and Oracles. At the beginning of the $\mathsf{Expr}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ security model, the safe predicates for identity keys, the reveal and corruption lists for pre-keys and states, and the pre-key counters are initialized. Then, the attacker is given access to $\mathcal{O}_1 := \{\mathcal{O}_{\mathsf{NewIdKey\text{-}A}}, \mathcal{O}_{\mathsf{NewIdKey\text{-}B}}, \mathcal{O}_{\mathsf{NewPreKey\text{-}A}}, \mathcal{O}_{\mathsf{NewPreKey\text{-}B}}\}$ oracles for generating both parties' identity keys and at least one pre-keys. A random initial shared secret *iss* is sampled from the space $\mathcal{ISS}$. Then, the session states $\mathsf{st}_\mathtt{A}$ and $\mathsf{st}_\mathtt{B}$ are respectively initialized by $\mathsf{eInit\text{-}A}$ and $\mathsf{eInit\text{-}B}$ of eSM. After initializing the epoch and message index counters, the sets, and the winning predicates $\mathsf{win}^{\mathsf{corr}}$ and $\mathsf{win}^{\mathsf{auth}}$, a challenge bit b is randomly sampled.

The attacker is given access to all eighteen oracles and terminates the experiment by outputting a bit b' for evaluating the winning predicate $\mathsf{win}^{\mathsf{priv}}$. Finally, the experiment outputs all these three winning predicates. In Figure 7.4, we only depict nine oracles with suffix -A for party $\mathtt{A}$. The oracles for party $\mathtt{B}$ are defined analogously.

Oracle Category 1: Identity and pre-keys. The first eight oracles are related to the generation and the leakage of identity keys and pre-keys.

- $\mathcal{O}_{\mathsf{NewIdKey\text{-}A}}(r)$, $\mathcal{O}_{\mathsf{NewIdKey\text{-}B}}(r)$: Both oracles can be queried at most once. The input random string, which is sampled when necessary, is used to produce a public-private identity key pair by using $\mathsf{IdKGen}(r)$. The corresponding safety flags are set according to whether the input $r = \perp$ or not. The public key is returned.
- $\mathcal{O}_{\mathsf{NewPreKey\text{-}A}}(r)$, $\mathcal{O}_{\mathsf{NewPreKey\text{-}B}}(r)$: Similar to the oracles above, a public-private pre-key pair is generated. The corresponding pre-key index is added into the list $\mathcal{L}_{\mathtt{A}}^{\mathsf{rev}}$ or $\mathcal{L}_{\mathtt{B}}^{\mathsf{rev}}$ if the input $r \neq \perp$. The public key is returned.
- $\mathcal{O}_{\mathsf{RevIdKey\text{-}A}}$, $\mathcal{O}_{\mathsf{RevIdKey\text{-}B}}$: These oracles simulate the reveal of the identity private key of a party $\mathtt{P} \in \{\mathtt{A}, \mathtt{B}\}$. The corresponding safe predicate is set to false. Then, the $\mathsf{corruption\text{-}update}$ helper function is invoked to update whether the current and past states are still secure or not. We require that this oracle invocation does not cause the change of safe challenge predicate for any record in the all-challenge set $\mathsf{allChall}$. Otherwise, this oracle undoes all actions during this invocation and exits. This step prevents the attacker from distinguishing the challenge bit by trivially revealing enough information to decrypt the past challenge ciphertexts.

  Then, all records in the transcript set trans, whose safe injection predicate turns to false, are added into the compromise set comp. This step prevents the attacker from making a trivial forgery by using the information leaked by the reveal of the identity key.

  Finally, the corresponding private identity key is returned.
- $\mathcal{O}_{\mathsf{RevPreKey\text{-}A}}(n)$, $\mathcal{O}_{\mathsf{RevPreKey\text{-}B}}(n)$: These oracles simulate the reveal of the n-th private pre-key of a party $\mathtt{P}$. The input n must indicate a valid prekey counter, i.e., $n \leq n_{\mathtt{P}}$, and is added into the reveal list $\mathcal{L}_{\mathtt{P}}^{\mathsf{rev}}$. The rest of these oracles are same as above: (1) runs $\mathsf{corruption\text{-}update}$, (2) aborts the oracles if the safe challenge predicates of any record in the $\mathsf{allChall}$ set is violated, and (3) adds all records in the trans set, whose safe injection predicate is violated, into the set comp.

  Finally, the corresponding private pre-key is returned.

Oracle Category 2: State Corruption. The following two oracles allow attackers to corrupt session states.

- $\mathcal{O}_{\textsf{Corrupt-A}}$, $\mathcal{O}_{\textsf{Corrupt-B}}$: These oracles simulate the corruption of party $\mathtt{P}$'s session states. First, the current epoch counter is added to the state corruption list $\mathcal{L}^{\textsf{cor}}_{\mathtt{P}}$, followed running $\textsf{corruption-update}$ to update whether this corruption impacts the safety of other session states. Next, we require that either the set $\textsf{chall}$ does not include the record produced by the partner $\neg\mathtt{P}$, or such a record exists but (1) the $\textsf{flag}$ in the record is $\textsf{good}$ and (2) $\mathtt{P}$'s identity key or $\mathtt{P}$'s pre-key corresponding to the pre-key index in the record is safe. If the requirement is not satisfied, this oracle undoes all actions in this invocation and exits. This requirement prevents the attacker from distinguishing the challenge bit by trivially revealing enough information to decrypt the past challenge ciphertexts.

  After that, we add all records $\textsf{rec} \in \textsf{trans}$, which are produced by $\neg\mathtt{P}$ at an unsafe epoch t, into the compromise set $\textsf{comp}$. We also add all records $\textsf{rec} \in \textsf{trans}$, which are produced by $\mathtt{P}$ at current epoch if the partner's session at current epoch is not safe. This requirement prevents the attacker from trivially breaking the strong authenticity by corrupting the sender's state and forging the corresponding undelivered messages.

  Finally, the session states are returned.

Oracle Category 3: Message Transmission. The final eight oracles simulate the honest message encryptions and the attacker's capability of manipulating the message transmission.

- $\mathcal{O}_{\textsf{Transmit-A}}(\textsf{ind}, m, r)$, $\mathcal{O}_{\textsf{Transmit-B}}(\textsf{ind}, m, r)$: These transmission oracles simulate the real sending execution. The input index $\textsf{ind}$ must not exceed the partner's current pre-key counter. The random string r is sampled when necessary. The epoch information is updated if entering a new epoch. After incrementing the message index, the $\textsf{eSend}$ algorithm is executed using the controlled or freshly sampled randomness r to transmit the message m upon the partner's identity key and $\textsf{ind}$-th pre-key. After recording the transcript, the ciphertext is returned.
- $\mathcal{O}_{\textsf{Challenge-A}}(\textsf{ind}, m_0, m_1, r)$, $\mathcal{O}_{\textsf{Challenge-B}}(\textsf{ind}, m_0, m_1, r)$: These challenge oracles simulate the sending execution, where the attacker tries to distinguish the encrypted message m_0 or m_1. These oracles are defined similar to the execution of transmission oracles with input $(\textsf{ind}, m_{\textsf{b}}, r)$ for the challenge bit $\textsf{b} \in \{0, 1\}$ sampled at the beginning of the experiment. The only difference is that the safety predicate $\textsf{safe-ch}_{\mathtt{P}}(\textsf{flag}, t_{\mathtt{P}}, \textsf{ind})$ for $\mathtt{P} \in \{\mathtt{A}, \mathtt{B}\}$ must hold and that the input messages m_0 and m_1 must have the same length.
- $\mathcal{O}_{\textsf{Deliver-A}}(\textsf{ind}, c)$, $\mathcal{O}_{\textsf{Deliver-B}}(\textsf{ind}, c)$: These delivery oracles simulate the receiving execution of a ciphertext generated by the honest party. This means, there must exist a record $(\mathtt{P}, \textsf{ind}, t, i, m, c)$ in the transcript set $\textsf{trans}$. The $\textsf{eRcv}$ is invoked. If the output epoch t', message index i', and decrypted message m' does not match the one in the record, the attacker wins via the predicate $\textsf{win}^{\textsf{corr}}$. If the output is in the challenge set $\textsf{chall}$, the

decrypted message m' is set to $\perp$ to prevent the attacker from trivially distinguishing the challenge bit. After updating the epoch counter, the record is deleted from transcript set, challenge set, and compromise set. This in particular means that the ciphertext c is considered as a forgery after this delivery. Finally, the output epoch t', the message index i', and the decrypted message m' is are returned.

- $\mathcal{O}_{\mathsf{Inject\text{-}A}}(\mathsf{ind}, c)$, $\mathcal{O}_{\mathsf{Inject\text{-}B}}(\mathsf{ind}, c)$: These oracles simulate a party P's receiving execution of a ciphertext forged by the attacker. The input $\mathsf{ind} \leq n_{\mathsf{P}}$ specifies a pre-key for running eRcv and the input c must be not produced by the partner in the transcript set. We require that eRcv is invoked under the condition that the safety predicates $\mathsf{safe\text{-}inj}_{\mathsf{P}}(t_{\mathtt{A}})$ and $\mathsf{safe\text{-}inj}_{\mathsf{P}}(t_{\mathtt{B}})$ both are true. If the decrypted message is not $\perp$ and the ciphertext at the same position is not compromised, the attacker wins via the $\mathsf{win}^{\mathsf{auth}}$ predicate. The rest of this oracle is identical to the delivery oracles.

Definition 64. *An* eSM *scheme* Π *is* $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \epsilon)$-eSM *secure if the below defined advantage for all attackers* $\mathcal{A}$ *against the* $\mathrm{Expr}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ *experiment in Figure 7.4 in time t is bounded by*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) := \max\Big(&\Pr[\mathrm{Expr}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (1,0,0)],\\ &\Pr[\mathrm{Expr}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (0,1,0)],\\ &|\Pr[\mathrm{Expr}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (0,0,1)] - \frac{1}{2}|\Big) \leq \epsilon,\end{aligned}$$

where q, q_{ep}, and q_{M} respectively denote the maximal number of queries $\mathcal{A}$ can make, of epochs, and of each party's pre-keys in the $\mathrm{Expr}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ *experiment.*

Conclusion. Finally, we explain how our eSM security captures all security properties listed in Section 7.4.2.

- **Correctness:** No correctness means the encrypted message cannot be recovered correctly and causes the winning event via Line 45.
- **Immediate decryption and message-loss resilience:** No immediate decryption or message-loss resilience means that some messages cannot be recovered to the correct position from the delivered ciphertext when the attacker invokes the transmission and delivery oracles in an arbitrary order, which causes the winning event via Line 45.
- **Forward secrecy**: Note that the attacker can freely access the corruption oracles if all challenge ciphertexts have been delivered. No FS means that the attacker can distinguish the challenge bit from the past encrypted messages and wins via Line 11.
- **Post-compromise security:** Note that the states are not leaked to a passive attacker after the owner sends a reply in a new epoch (i.e., epochs are not added into the state

corruption list in Line 80), assuming fresh randomness and the partner's uncorrupted state, or identity key or pre-key, see Line 79.

No PCS indicates that a state at an epoch not in the state corruption lists might still be corrupted, which causes the lose of other security properties.

- **Strong authenticity:** The attacker can inject a forged ciphertext (Line 49) that does not correspond to a compromised ciphertext position (Line 52) if sender's session state is safe. Recall that a ciphertext is compromised only when the session state of the sender is unsafe (see Line 21, 33, 65, 68, 76).

  No strong authenticity means that the forged ciphertext can be decrypted to a non-$\perp$ message when the sender is not corrupted, and further causes the winning of the attacker via Line 52.

- **Strong privacy**: Note that the challenge ciphertexts must be produced without the violation the safety predicate $\mathsf{safe\text{-}ch}$ in Line 58, i.e., at least one of the following combinations are not leaked: (1) both parties' states, (2) the encryption randomness and the receiver's state, (3) the encryption randomness and the receiver's private identity key, or (4) the encryption randomness and the receiver's corresponding private pre-key. Moreover, our identity key reveal oracles, pre-key reveal oracles, and state corruption oraclesalso prevent the attacker from knowing all of the above combinations related to any challenge ciphertext at the same time (see Line 20, 32, 64).

  No strong privacy means that the attacker can distinguish the challenge bit even when at least one of the above four combinations holds, which further causes the winning event via Line 11.

- **Randomness leakage/failures:** This is ensured by the fact that all of the above properties hold if the parties' session states are uncompromised.
- **State compromise/failures:** This is ensured by the strong privacy even when both parties' state are corrupted, as explained above.
- **Periodic privacy recovery (PPR):** Note that the pre-keys can be periodically generated optionally under fresh randomness. The PPR is ensured by the strong privacy when the sender's randomness is good and the receiver's newly freshly sampled pre-key is safe, as explained above.

Moreover, we can also observe that higher security can be obtained if the device of a party (assume $\mathtt{A}$) supports a secure environment, such as an HSM. If $\mathtt{A}$'s identity key pair is generated in a secure environment, the private identity key can be neither manipulated nor predicted by any attacker. This means that the attacker can only query $\mathcal{O}_{\mathsf{NewIdKey\text{-}A}}(r)$ with input $r = \perp$ and never query $\mathcal{O}_{\mathsf{RevIdKey\text{-}A}}$ oracle in $\mathrm{Expr}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$. Thus, the predicate

$\mathsf{safe}_{\mathtt{A}}^{\mathsf{idK}}$ is always true. If the partner B has access to the fresh randomness, then the privacy of the messages sent from B to A always holds.

We stress that our eSM model is strictly stronger than the SM model [12], even without taking the usage of identity keys and pre-keys into account. We provide a detailed comparison in Section 7.9 for interested readers.

7.4.4 Security Model Modularization

The analysis for the security of messaging protocols are often very tedious, since both the security model and the protocols are usually highly complex. Alwen et al. [12] opt to first reduce the SM-security into several simplified security notions: correctness, privacy, and authenticity. Then, they respectively prove the individual simplified security of their proposal ACD19.

We adopt the similar strategies: we split the eSM-security into several new simplified security notions and prove the reduction between eSM and the new simplified security notions.

Correctness: We define our correctness model $\text{Expr}_{\Pi,\triangle_{\mathsf{eSM}}}^{\mathsf{CORR}}$ for an eSM scheme Π with respect to a parameter $\triangle_{\mathsf{eSM}}$ identical to the model $\text{Expr}_{\Pi,\triangle_{\mathsf{eSM}}}^{\mathsf{eSM}}$ with the same parameter $\triangle_{\mathsf{eSM}}$, except for the following modifications:

1. there are no $\mathcal{O}_{\mathsf{Challenge\text{-}A}}$ and $\mathcal{O}_{\mathsf{Challenge\text{-}B}}$ oracles
2. the $\mathcal{O}_{\mathsf{Inject\text{-}A}}$ and $\mathcal{O}_{\mathsf{Inject\text{-}B}}$ are replaced by a reduced injection oracle, which is identical to the injection oracle except for the following two modifications:
   - if the input ciphertext c does not correspond to any position $(t', i') \in \mathsf{comp}$, $\mathcal{O}_{\mathsf{Inject\text{-}A}}$ and $\mathcal{O}_{\mathsf{Inject\text{-}B}}$ immediately returns $(t', i', \perp)$
   - the if-clause in Line 52 and 52 are removed

This simplified correctness experiment is defined similar to the one in [12].

Note that the attacker receives no information about the challenge bit, since the challenge oracles are removed. The attacker cannot win via the predicate $\mathsf{win}^{\mathsf{priv}}$ except by randomly guessing. Moreover, the predicate $\mathsf{win}^{\mathsf{auth}}$ in the injection oracles is removed. The $\mathsf{win}^{\mathsf{auth}}$ predicate is never set to true. Intuitively, the attacker can win the correctness game with non-zero advantage only via $\mathsf{win}^{\mathsf{corr}}$ in the $\mathcal{O}_{\mathsf{Deliver\text{-}A}}$ and $\mathcal{O}_{\mathsf{Deliver\text{-}B}}$ oracles.

Definition 65. *An* eSM *scheme* Π *is* $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \epsilon)$-CORR *secure if the below defined advantage for any attacker* $\mathcal{A}$ *in time* t *is bounded by*

$$\mathsf{Adv}_{\Pi,\triangle_{\mathsf{eSM}}}^{\mathsf{CORR}}(\mathcal{A}) := \Pr[\text{Expr}_{\Pi,\triangle_{\mathsf{eSM}}}^{\mathsf{CORR}}(\mathcal{A}) = (1, 0, 0)] \leq \epsilon,$$

where q, q_{ep}, *and* q_{M} *respectively denote the maximal number of queries* $\mathcal{A}$ *can make, the maximal number of epochs, and the maximal number of pre-keys of each party in the experiment* $\text{Expr}_{\Pi,\triangle_{\mathsf{eSM}}}^{\mathsf{CORR}}$.

Authenticity: We define our authenticity model $\mathrm{Expr}^{\mathsf{AUTH}}_{\Pi,\triangle_{\mathsf{eSM}}}$ for an eSM scheme Π with respect to a parameter $\triangle_{\mathsf{eSM}}$ identical to the model $\mathrm{Expr}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ with the same parameter $\triangle_{\mathsf{eSM}}$, except for the following modifications:

1. there are no $\mathcal{O}_{\mathsf{Challenge\text{-}A}}$ and $\mathcal{O}_{\mathsf{Challenge\text{-}B}}$ oracles
2. the winning predicate $\mathsf{win}^{\mathsf{corr}}$ is never set to true in the $\mathcal{O}_{\mathsf{Deliver\text{-}A}}$ and $\mathcal{O}_{\mathsf{Deliver\text{-}B}}$, i.e., the if-clause in Line 45 is removed.
3. the attacker has to output an epoch $t^\star$ at the beginning of the experiment
4. the $\mathcal{O}_{\mathsf{Inject\text{-}A}}$ and $\mathcal{O}_{\mathsf{Inject\text{-}B}}$ are replaced by a reduced injection oracle (see above) unless the input ciphertext c corresponds to the epoch $t^\star$. (Recall that the position including the epoch and message index is assumed to be efficiently computable from c for natural eSM.)

This simplified authenticity experiment is defined differently from the one in [12], as the attacker has to output only one epoch $t^\star$, which indicates the epoch of the forged ciphertext, without outputting another epoch $t^\star_L$ as in [12], which indicating the last corruption event before the $t^\star$.

Note that the attacker receives no information about the challenge bit, since the challenge oracles are removed. The attacker cannot win via the predicate $\mathsf{win}^{\mathsf{priv}}$ except by randomly guessing. Moreover, the predicate $\mathsf{win}^{\mathsf{corr}}$ in the deliver oracles is removed. The $\mathsf{win}^{\mathsf{corr}}$ predicate is never set to true. Intuitively, the attacker can win the authenticity game with non-zero advantage only via $\mathsf{win}^{\mathsf{auth}}$ in the $\mathcal{O}_{\mathsf{Inject\text{-}A}}$ and $\mathcal{O}_{\mathsf{Inject\text{-}B}}$ oracles for a forged ciphertext corresponding to the epoch $t^\star$, which is claimed by the attacker at the beginning of the experiment.

Definition 66. *An* eSM *scheme* Π *is* $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \epsilon)$-$\mathsf{AUTH}$ *secure if the below defined advantage for any attacker* $\mathcal{A}$ *in time* t *is bounded by*

$$\mathsf{Adv}^{\mathsf{AUTH}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) := \Pr[\mathrm{Expr}^{\mathsf{AUTH}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (0,1,0)] \leq \epsilon,$$

where q, q_{ep}, *and* q_{M} *respectively denote the maximal number of queries* $\mathcal{A}$ *can make, the maximal number of epochs, and the maximal number of pre-keys of each party in the experiment* $\mathrm{Expr}^{\mathsf{AUTH}}_{\Pi,\triangle_{\mathsf{eSM}}}$.

Privacy: We define our privacy model $\mathrm{Expr}^{\mathsf{PRIV}}_{\Pi,\triangle_{\mathsf{eSM}}}$ for an eSM scheme Π with respect to a parameter $\triangle_{\mathsf{eSM}}$ identical to the model $\mathrm{Expr}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ with the same parameter $\triangle_{\mathsf{eSM}}$, except for the following modifications:

1. the winning predicate $\mathsf{win}^{\mathsf{corr}}$ is never set to true in the $\mathcal{O}_{\mathsf{Deliver\text{-}A}}$ and $\mathcal{O}_{\mathsf{Deliver\text{-}B}}$, i.e., the if-clause in Line 45 is removed.

2. the $\mathcal{O}_{\mathsf{Inject\text{-}A}}$ and $\mathcal{O}_{\mathsf{Inject\text{-}B}}$ are replaced by a reduced injection oracle (see above).
3. the attacker has to output an epoch $t^\star$ at the beginning of the experiment.
4. the challenge oracle $\mathcal{O}_{\mathsf{Challenge\text{-}A}}$ (resp. $\mathcal{O}_{\mathsf{Challenge\text{-}B}}$) can only be queried if $t_{\mathtt{A}} = t^\star$ (resp. $t_{\mathtt{B}} = t^\star$)

This simplified privacy experiment is also defined differently from the one in [12], as the attacker has to output only one epoch, which indicates the epoch of the challenge query, without outputting another epoch $t_L^\star$ as in [12], which indicating the last corruption event before the $t^\star$.

Note that the predicate $\mathsf{win}^{\mathsf{corr}}$ in the deliver oracles and the $\mathsf{win}^{\mathsf{auth}}$ in the injection oracles are removed. The $\mathsf{win}^{\mathsf{corr}}$ and $\mathsf{win}^{\mathsf{auth}}$ predicates are never set to true. Intuitively, the attacker can win the privacy game only via $\mathsf{win}^{\mathsf{priv}}$ predicate by distinguishing the challenge bit using the challenge ciphertexts corresponding to the epoch $t^\star$, which is claimed by the attacker at the beginning of the experiment.

Definition 67. *An* eSM *scheme* Π *is* $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \epsilon)$-PRIV *secure if the below defined advantage for any attacker* $\mathcal{A}$ *in time* t *is bounded by*

$$\mathsf{Adv}^{\mathsf{PRIV}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) := \Pr[\mathrm{Expr}^{\mathsf{PRIV}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (0,0,1)] \leq \epsilon,$$

where q*,* q_{ep}*, and* q_{M} *respectively denote the maximal number of queries* $\mathcal{A}$ *can make, the maximal number of epochs, and the maximal number of pre-keys of each party in the experiment* $\mathrm{Expr}^{\mathsf{PRIV}}_{\Pi,\triangle_{\mathsf{eSM}}}$.

7.5 Extended Secure Messaging Scheme

In Section 7.5.1 we describe the intuition behind our eSM construction, followed by a detailed description in Section 7.5.2. In Section 7.5.3, we prove the eSM security of our eSM construction and provide concrete instantiations.

7.5.1 Intuition behind the eSM Construction

Our eSM construction, depicted in Figure 7.6, uses a key encapsulation mechanism $\mathsf{KEM} = (\mathsf{KEM.KGen}, \mathsf{KEM.Encaps}, \mathsf{KEM.Decaps})$, a digital signature $\mathsf{DS} = (\mathsf{DS.KGen}, \mathsf{DS.Sign}, \mathsf{DS.Vrfy})$, a symmetric key encryption $\mathsf{SKE} = (\mathsf{SKE.Enc}, \mathsf{SKE.Dec})$, and five key derivation functions KDF_i for $i \in [5]$.

To send a message, the sender runs the KEM encapsulation algorithm three times: the encapsulation upon the partner's latest per-epoch public key, which ensures the privacy against fine-grained state compromise and PCS; the one upon the partner's latest public pre-key, which ensures temporal privacy and the PPR property; and finally the one upon the partner's latest public identity key, which ensures even stronger privacy if the device supports an HSM for storing private identity keys. The sender also signs the outgoing ciphertext using DS and his latest per-epoch signing key to ensure the authenticity against fine-grained state compromise.

Moreover, our eSM construction uses three variants of the NAXOS trick [134], in which ephemeral randomness is combined with a local secret to strengthen against randomness compromise or manipulation attack. First, a symmetric root key $\mathsf{st}.rk$ together with ephemeral randomness is used to derive new shared state when sending the first message in each epoch. This provides strong privacy for new epochs against randomness leakage and manipulation; Second, the sender's local NAXOS string $\mathsf{st}.nxs$ together with the ephemeral randomness is used to improve key generation when sending the first message in each epoch. This provides strong authenticity for the new epoch and strong privacy for the next epoch against randomness leakage and manipulation; Third, the unidirectional ratchet keys urk (derived from the shared state) together with the ephemeral randomness are used to derive the real message keys. This ensures FS while preserving immediate decryption with constant-size overhead.

7.5.2 The eSM Construction in Detail

For simplicity, we assume all symmetric keys in our construction (including the root key rk, the chain key ck, the unidirectional ratchet key urk, and the message key mk) have the same domain $\{0,1\}^\lambda$. We assume the key generation randomness spaces of KEM and DS are also $\{0,1\}^\lambda$. The underlying DS and SKE are assumed to be deterministic. We first introduce the state in our construction.

Definition 68. *The state in our* eSM *construction in Figure 7.6 consists of following variables:*

- $\mathsf{st.id}$*: the state owner. In this paper, we have* $\mathsf{st_A.id} = \mathtt{A}$ *and* $\mathsf{st_B.id} = \mathtt{B}$*.*
- $\mathsf{st}.t$*: the local epoch counter. It starts with* 0*.*
- $\mathsf{st}.i^0$, $\mathsf{st}.i^1$, ...*: the local message index counter of each epoch. They start with* 0*.*
- $\mathsf{st}.rk \in \{0,1\}^\lambda$*: the (symmetric) root key. This key is initialized from the initial shared secret and updated only when entering next epoch. The root key is used to initialize the chain key at the time of update.*
- $\mathsf{st}.ck^0, \mathsf{st}.ck^1, ... \in \{0,1\}^\lambda$*: the (symmetric) chain keys at each epoch. These keys are initialized at the beginning of each epoch and updated when sending messages. The chain keys are used to deterministically derive the (one-time symmetric) unidirectional ratchet keys (urk).*
- $\mathsf{st}.nxs \in \{0,1\}^\lambda$*: a local NAXOS random string, which is used to improve the randomness when generating new* KEM *and* DS *key pairs.*

- $\mathsf{st}.\mathcal{D}_l$: *the dictionary that stores the maximal number (aka. the length) of the transmissions in the previous epochs.*
- $\mathsf{st}.\mathsf{prtr}$: *the pre-transcript that is produced at the beginning of each epoch and is attached to the ciphertext whenever sending messages in the same epoch.*
- $\mathsf{st}.\mathcal{D}^0_{urk}, \mathsf{st}.\mathcal{D}^1_{urk}, \ldots$: *the dictionaries that store the (one-time symmetric) unidirectional ratchet keys urk for each epoch. The urks are used to derive the (one-time symmetric) message keys (mk) for real message encryption and decryption using* SKE*.*
- $(\mathsf{st}.ek^0, \mathsf{st}.dk^0), (\mathsf{st}.ek^1, \mathsf{st}.dk^1), \ldots$: *the (asymmetric)* KEM *public key pairs. These key pairs are used to encapsulate and decapsulate the randomness, which (together with the unidirectional ratchet key urk) is used to derive the message keys (mk) of* SKE*.*
- $(\mathsf{st}.sk^{-1}, \mathsf{st}.vk^{-1}), (\mathsf{st}.sk^0, \mathsf{st}.vk^0), (\mathsf{st}.sk^1, \mathsf{st}.vk^1), \ldots$[4]: *the (asymmetric)* DS *private key pairs, which are used to sign and verify the (new) pre-transcript output by* eSend*.*

Our eSM construction makes use of two auxiliary functions: eSend-Stop and eRcv-Max for practical memory management. Here, we only explain the underlying mechanism and omit their concrete instantiation.

- eRcv-Max(st, l): This algorithm is called in eRcv algorithm when the caller switches its role from message sender in epoch $\mathsf{st}.t$ to message receiver in a new epoch $\mathsf{st}.t+1$. This algorithm inputs (the caller's) state st and a number l and remembers the value l together with the epoch counter $t' = \mathsf{st}.t - 1$ locally. Once l messages corresponds to the old epoch t' are received, the state values for receiving messages in epoch t', i.e., $\mathsf{st}.i^{t'}$, $\mathsf{st}.ck^{t'}$, $\mathsf{st}.dk^{t'}$, $\mathsf{st}.vk^{t'}$, $\mathsf{st}.\mathcal{D}^{t'}_{urk}$, $\mathsf{st}.\mathcal{D}_l[t']$ are erased, i.e., set to $\perp$. Moreover, the number how many times the chain key $\mathsf{st}.ck^{\mathsf{st}.t}$ has been forwarded (i.e., how many messages have been sent) in the epoch $\mathsf{st}.t$ is stored, while the chain key $\mathsf{st}.ck^{\mathsf{st}.t}$ itself together with the encryption key $\mathsf{st}.ek^{\mathsf{st}.t}$ is erased.
- eSend-Stop(st): This algorithm is called in eSend algorithm when the caller switches its role from the message receiver in epoch $\mathsf{st}.t$ to the message sender in a new epoch $\mathsf{st}.t+1$. This algorithm inputs (the caller's) state st and outputs how many messages are sent in the epoch $\mathsf{st}.t-1$, which is locally stored during the previous eRcv-Max invocation, denoted by l. The signing key $\mathsf{st}.sk^t$ is also erased after its signs the next verification key $\mathsf{st}.vk^{t+2}$ later. We write $l \leftarrow \text{eSend-Stop}(\mathsf{st})$.

Following the syntax in Definition 62, our eSM construction consists of following six algorithms below.

IdKGen()**:** The identity key generation algorithm samples and outputs a public-private KEM key pair.

[4]The superscript of the signing/verification keys indicates the epochs when the DS key pairs are generated and used until the next key generation two epochs later. Here, we slightly abuse the notation and have $\mathsf{st}.sk^{-1}$ and $\mathsf{st}.vk^{-1}$, which are used only to sign/verify the verification key in epoch 1.

```
$\underline{\mathsf{IdKGen}():}$
$(ipk, ik) \xleftarrow{\$} \mathsf{KEM.KGen}()$
$\textbf{return } (ipk, ik)$

$\underline{\mathsf{PreKGen}():}$
$(prepk, prek) \xleftarrow{\$} \mathsf{KEM.KGen}()$
$\textbf{return } (prepk, prek)$

$\underline{\mathsf{eInit\text{-}A}(iss):}$
$\mathsf{st}_{\mathtt{A}}.nxs \,\|\, \_ \,\|\, \mathsf{st}_{\mathtt{A}}.rk \,\|\, \mathsf{st}_{\mathtt{A}}.ck^0 \,\|\, r_{\mathtt{A}}^{\mathsf{KEM}} \,\|\, r_{\mathtt{B}}^{\mathsf{KEM}} \,\|\, r_{\mathtt{A}}^{\mathsf{DS}} \,\|\, r_{\mathtt{B}}^{\mathsf{DS}} \leftarrow iss$
$(\_, \mathsf{st}_{\mathtt{A}}.dk^0) \xleftarrow{\$} \mathsf{KEM.KGen}(r_{\mathtt{A}}^{\mathsf{KEM}}),\ (\mathsf{st}_{\mathtt{A}}.ek^1, \_) \xleftarrow{\$} \mathsf{KEM.KGen}(r_{\mathtt{B}}^{\mathsf{KEM}})$
$(\mathsf{st}_{\mathtt{A}}.sk^{-1}, \_) \xleftarrow{\$} \mathsf{DS.KGen}(r_{\mathtt{A}}^{\mathsf{DS}}),\ (\_, \mathsf{st}_{\mathtt{A}}.vk^0) \xleftarrow{\$} \mathsf{DS.KGen}(r_{\mathtt{B}}^{\mathsf{DS}})$
$\mathsf{st}_{\mathtt{A}}.\mathsf{id} \leftarrow \mathtt{A},\ \mathsf{st}_{\mathtt{A}}.\mathsf{prtr} \leftarrow \perp,\ \mathsf{st}_{\mathtt{A}}.t \leftarrow 0,\ \mathsf{st}_{\mathtt{A}}.i^0 \leftarrow 0,\ \mathsf{st}_{\mathtt{A}}.\mathcal{D}_l[\cdot] \leftarrow \perp,\ \mathsf{st}_{\mathtt{A}}.\mathcal{D}_{urk}^0[\cdot] \leftarrow \perp$
$\textbf{return } \mathsf{st}_{\mathtt{A}}$

$\underline{\mathsf{eInit\text{-}B}(iss):}$
$\_ \,\|\, \mathsf{st}_{\mathtt{B}}.nxs \,\|\, \mathsf{st}_{\mathtt{B}}.rk \,\|\, \mathsf{st}_{\mathtt{B}}.ck^0 \,\|\, r_{\mathtt{A}}^{\mathsf{KEM}} \,\|\, r_{\mathtt{B}}^{\mathsf{KEM}} \,\|\, r_{\mathtt{A}}^{\mathsf{DS}} \,\|\, r_{\mathtt{B}}^{\mathsf{DS}} \leftarrow iss$
$(\mathsf{st}_{\mathtt{B}}.ek^0, \_) \xleftarrow{\$} \mathsf{KEM.KGen}(r_{\mathtt{A}}^{\mathsf{KEM}}),\ (\_, \mathsf{st}_{\mathtt{B}}.dk^1) \xleftarrow{\$} \mathsf{KEM.KGen}(r_{\mathtt{B}}^{\mathsf{KEM}})$
$(\_, \mathsf{st}_{\mathtt{B}}.vk^{-1}) \xleftarrow{\$} \mathsf{DS.KGen}(r_{\mathtt{A}}^{\mathsf{DS}}),\ (\mathsf{st}_{\mathtt{B}}.sk^0, \_) \xleftarrow{\$} \mathsf{DS.KGen}(r_{\mathtt{B}}^{\mathsf{DS}})$
$\mathsf{st}_{\mathtt{B}}.\mathsf{id} \leftarrow \mathtt{B},\ \mathsf{st}_{\mathtt{B}}.\mathsf{prtr} \leftarrow \perp,\ \mathsf{st}_{\mathtt{B}}.t \leftarrow 0,\ \mathsf{st}_{\mathtt{B}}.i^0 \leftarrow 0,\ \mathsf{st}_{\mathtt{B}}.\mathcal{D}_l[\cdot] \leftarrow \perp$
$\textbf{return } \mathsf{st}_{\mathtt{B}}$

$\underline{\mathsf{eSend}(\mathsf{st}, ipk, prepk, m):}$
$(c_1, k_1) \xleftarrow{\$} \mathsf{KEM.Encaps}(\mathsf{st}.ek^{\mathsf{st}.t}),\ (c_2, k_2) \xleftarrow{\$} \mathsf{KEM.Encaps}(ipk),\ (c_3, k_3) \xleftarrow{\$} \mathsf{KEM.Encaps}(prepk)$
$(\mathsf{upd}^{\mathsf{ar}}, \mathsf{upd}^{\mathsf{ur}}) \leftarrow \mathsf{KDF}_1(k_1, k_2, k_3)$
$\textbf{if } (\mathsf{st.id} = \mathtt{A} \textbf{ and } \mathsf{st}.t\ \ even) \textbf{ or } (\mathsf{st.id} = \mathtt{B} \textbf{ and } \mathsf{st}.t\ \ odd)$
    $l \leftarrow \mathsf{eSend\text{-}Stop}(\mathsf{st}),\ \mathsf{st}.t{+}{+},\ \mathsf{st}.i^{\mathsf{st}.t} \leftarrow 0,\ r \xleftarrow{\$} \{0,1\}^\lambda,\ (\mathsf{st}.nxs, r^{\mathsf{KEM}}, r^{\mathsf{DS}}) \leftarrow \mathsf{KDF}_2(\mathsf{st}.nxs, r)$
    $(ek, \mathsf{st}.dk^{\mathsf{st}.t+1}) \xleftarrow{\$} \mathsf{KEM.KGen}(r^{\mathsf{KEM}}),\ (\mathsf{st}.sk^{\mathsf{st}.t}, vk) \xleftarrow{\$} \mathsf{DS.KGen}(r^{\mathsf{DS}})$
    $\mathsf{prtr}^{\mathsf{ar}} \leftarrow (l, c_1, c_2, c_3, ek, vk),\ \sigma^{\mathsf{ar}} \leftarrow \mathsf{DS.Sign}(\mathsf{st}.sk^{\mathsf{st}.t-2}, \mathsf{prtr}^{\mathsf{ar}})$
    $\mathsf{st.prtr} \leftarrow (\mathsf{prtr}^{\mathsf{ar}}, \sigma^{\mathsf{ar}}),\ (\mathsf{st}.rk, \mathsf{st}.ck^{\mathsf{st}.t}) \leftarrow \mathsf{KDF}_3(\mathsf{st}.rk, \mathsf{upd}^{\mathsf{ar}})$
$(\mathsf{st}.ck^{\mathsf{st}.t}, urk) \leftarrow \mathsf{KDF}_4(\mathsf{st}.ck^{\mathsf{st}.t}),\ mk \leftarrow \mathsf{KDF}_5(urk, \mathsf{upd}^{\mathsf{ur}})\ ,\ c' \leftarrow \mathsf{SKE.Enc}(mk, m)$
$\mathsf{prtr}^{\mathsf{ur}} \leftarrow (\mathsf{st}.t, \mathsf{st}.i^{\mathsf{st}.t}, c', c_1, c_2, c_3),\ \sigma^{\mathsf{ur}} \leftarrow \mathsf{DS.Sign}(\mathsf{st}.sk^{\mathsf{st}.t}, \mathsf{prtr}^{\mathsf{ur}})$
$\textbf{return } (\mathsf{st}, (\mathsf{st.prtr}, \mathsf{prtr}^{\mathsf{ur}}, \sigma^{\mathsf{ur}}))$

$\underline{\mathsf{eRcv}(\mathsf{st}, ik, prek, c):}$
$((\mathsf{prtr}^{\mathsf{ar}}, \sigma^{\mathsf{ar}}), \mathsf{prtr}^{\mathsf{ur}}, \sigma^{\mathsf{ur}}) \leftarrow c,\ (l, c_1, c_2, c_3, ek, vk) \leftarrow \mathsf{prtr}^{\mathsf{ar}},\ (t, i, c', c_1', c_2', c_3') \leftarrow \mathsf{prtr}^{\mathsf{ur}}$
$\textbf{if } t \leq \mathsf{st}.t - 2\text{: } \textbf{require } \mathsf{st}.\mathcal{D}_l[t] \neq \perp \textbf{ and } i \leq \mathsf{st}.\mathcal{D}_l[t]$
$\textbf{require } t \leq \mathsf{st}.t + 1 \textbf{ and } \Big((\mathsf{st.id} = \mathtt{A} \textbf{ and } t\ \ even) \textbf{ or } (\mathsf{st.id} = \mathtt{B} \textbf{ and } t\ \ odd)\Big)$
$\textbf{if } t = \mathsf{st}.t + 1$
    $\textbf{require } \mathsf{DS.Vrfy}(\mathsf{st}.vk^{t-2}, \mathsf{prtr}^{\mathsf{ar}}, \sigma^{\mathsf{ar}})$
    $\mathsf{eRcv\text{-}Max}(\mathsf{st}, l),\ \mathsf{st}.\mathcal{D}_l[t-2] \leftarrow l,\ \mathsf{st}.t{+}{+}$
    $k_1 \leftarrow \mathsf{KEM.Decaps}(\mathsf{st}.dk^{\mathsf{st}.t}, c_1),\ k_2 \leftarrow \mathsf{KEM.Decaps}(ik, c_2),\ k_3 \leftarrow \mathsf{KEM.Decaps}(prek, c_3)$
    $(\mathsf{upd}^{\mathsf{ar}}, \_) \leftarrow \mathsf{KDF}_1(k_1, k_2, k_3),\ (\mathsf{st}.rk, \mathsf{st}.ck^{\mathsf{st}.t}) \leftarrow \mathsf{KDF}_3(\mathsf{st}.rk, \mathsf{upd}^{\mathsf{ar}})$
    $\mathcal{D}_{urk}^{\mathsf{st}.t}[\cdot] \leftarrow \perp,\ \mathsf{st}.i^{\mathsf{st}.t} \leftarrow 0,\ \mathsf{st}.ek^{\mathsf{st}.t+1} \leftarrow ek,\ \mathsf{st}.vk^{\mathsf{st}.t} \leftarrow vk$
$\textbf{require } \mathsf{DS.Vrfy}(\mathsf{st}.vk^t, \mathsf{prtr}^{\mathsf{ur}}, \sigma^{\mathsf{ur}})$
$k_1' \leftarrow \mathsf{KEM.Decaps}(\mathsf{st}.dk^t, c_1'),\ k_2' \leftarrow \mathsf{KEM.Decaps}(ik, c_2'),\ k_3' \leftarrow \mathsf{KEM.Decaps}(prek, c_3')$
$(\_, \mathsf{upd}^{\mathsf{ur}}) \leftarrow \mathsf{KDF}_1(k_1', k_2', k_3')$
$\textbf{while } \mathsf{st}.i^t \leq i$
    $(\mathsf{st}.ck^t, urk) \leftarrow \mathsf{KDF}_4(\mathsf{st}.ck^t),\ \mathcal{D}_{urk}^t[\mathsf{st}.i^t] \leftarrow urk,\ \mathsf{st}.i^t{+}{+}$
$urk \leftarrow \mathcal{D}_{urk}^t[i],\ \mathcal{D}_{urk}^t[i] \leftarrow \perp,\ \textbf{require } urk \neq \perp$
$mk \leftarrow \mathsf{KDF}_5(urk, \mathsf{upd}^{\mathsf{ur}}),\ m \leftarrow \mathsf{SKE.Dec}(mk, c')$
$\textbf{return } (\mathsf{st}, t, i, m)$
```

Figure 7.6: Our eSM construction. $\mathsf{KEM} = (\mathsf{KEM.KGen}, \mathsf{KEM.Encaps}, \mathsf{KEM.Decaps})$, $\mathsf{DS} = (\mathsf{DS.KGen}, \mathsf{DS.Sign}, \mathsf{DS.Vrfy})$, and $\mathsf{SKE} = (\mathsf{SKE.Enc}, \mathsf{SKE.Enc})$ respectively denote a key encapsulation mechanism, a deterministic digital signature and a deterministic authenticated encryption schemes. The KDF_i for $i \in [5]$ denote five independent key derivation functions.

$\mathsf{PreKGen}()$**:** The pre-key generation algorithm samples and outputs a public-private KEM key pair.

$\mathsf{eInit\text{-}A}(iss)$**:** The $\mathtt{A}$'s extended initialization algorithm inputs an initial shared secret $iss \in \mathcal{ISS}$. First, $\mathtt{A}$ parses iss into seven components: the initial NAXOS string $\mathsf{st}_{\mathtt{A}}.nxs$, the shared root key $\mathsf{st}_{\mathtt{A}}.rk$, the shared chain key $\mathsf{st}_{\mathtt{A}}.ck^0$, and four randomness for $\mathtt{A}$'s and $\mathtt{B}$'s KEM and DS key generation: $r_{\mathtt{A}}^{\mathsf{KEM}}$, $r_{\mathtt{B}}^{\mathsf{KEM}}$, $r_{\mathtt{A}}^{\mathsf{DS}}$, $r_{\mathtt{B}}^{\mathsf{DS}}$. Then, $\mathtt{A}$ respectively runs $\mathsf{KEM.KGen}$ and $\mathsf{DS.KGen}$ on the above randomness and stores $\mathsf{st}_{\mathtt{A}}.dk^0$, $\mathsf{st}_{\mathtt{A}}.ek^1$, $\mathsf{st}_{\mathtt{A}}.sk^{\text{-}1}$, $\mathsf{st}_{\mathtt{A}}.vk^0$, which are respectively generated using $r_{\mathtt{A}}^{\mathsf{KEM}}$, $r_{\mathtt{B}}^{\mathsf{KEM}}$, $r_{\mathtt{A}}^{\mathsf{DS}}$, and $r_{\mathtt{B}}^{\mathsf{DS}}$. The other values generated in the meantime are discarded.

Finally, $\mathtt{A}$ sets the identity $\mathsf{st}_{\mathtt{A}}.\mathsf{id}$ to $\mathtt{A}$, the local pre-transcript $\mathsf{st}_{\mathtt{A}}.\mathsf{prtr}$ to $\perp$, the epoch counter $\mathsf{st}_{\mathtt{A}}.t$ to 0, the message index $\mathsf{st}_{\mathtt{A}}.i^0$ to 0, and initializes the maximal transmission length dictionary $\mathcal{D}_l$ and the unidirectional ratchet dictionary $\mathcal{D}_{urk}^0$, followed by outputting the state $\mathsf{st}_{\mathtt{A}}$.

$\mathsf{eInit\text{-}B}(iss)$**:** The $\mathtt{B}$'s extended initialization algorithm inputs an initial shared secret $iss \in \mathcal{ISS}$ and runs very similar to $\mathsf{eInit\text{-}A}$. First, $\mathtt{B}$ parses iss into seven components: the initial NAXOS string $\mathsf{st}_{\mathtt{B}}.nxs$, the shared root key $\mathsf{st}_{\mathtt{B}}.rk$, the shared chain key $\mathsf{st}_{\mathtt{B}}.ck^0$, and four randomness for $\mathtt{A}$'s and $\mathtt{B}$'s KEM and DS key generation: $r_{\mathtt{A}}^{\mathsf{KEM}}$, $r_{\mathtt{B}}^{\mathsf{KEM}}$, $r_{\mathtt{A}}^{\mathsf{DS}}$, $r_{\mathtt{B}}^{\mathsf{DS}}$. Then, $\mathtt{B}$ respectively runs $\mathsf{KEM.KGen}$ and $\mathsf{DS.KGen}$ on the above randomness and stores $\mathsf{st}_{\mathtt{B}}.ek^0$, $\mathsf{st}_{\mathtt{B}}.dk^1$, $\mathsf{st}_{\mathtt{B}}.vk^{\text{-}1}$, $\mathsf{st}_{\mathtt{A}}.sk^0$, which are respectively generated using $r_{\mathtt{A}}^{\mathsf{KEM}}$, $r_{\mathtt{B}}^{\mathsf{KEM}}$, $r_{\mathtt{A}}^{\mathsf{DS}}$, and $r_{\mathtt{B}}^{\mathsf{DS}}$. The other values generated in the meantime are discarded. Note that the values stored by $\mathtt{B}$ is the ones discarded by $\mathtt{A}$, and vice versa.

Finally, $\mathtt{B}$ sets the identity $\mathsf{st}_{\mathtt{B}}.\mathsf{id}$ to $\mathtt{B}$, the local pre-transcript $\mathsf{st}_{\mathtt{B}}.\mathsf{prtr}$ to $\perp$, the epoch counter $\mathsf{st}_{\mathtt{B}}.t$ to 0, the message index $\mathsf{st}_{\mathtt{B}}.i^0$ to 0, and initializes the maximal transmission length dictionary $\mathcal{D}_l$, followed by outputting the state $\mathsf{st}_{\mathtt{B}}$. Note that no unidirectional ratchet dictionary $\mathcal{D}_{urk}^0$ is initialized, since $\mathtt{B}$ acts as the sender in the epoch 0.

$\mathsf{eSend}(\mathsf{st}, ipk, prepk, m)$**:** The sending algorithm inputs the (caller's) state st, the (caller's partner's) public identity key ipk and pre-key $prepk$, and a message m.

First, the caller runs the encapsulation algorithm of KEM and obtains three ciphertext-key tuples (c_1, k_1), (c_2, k_2), and (c_3, k_3) respectively using the local key $\mathsf{st}.ek^{\mathsf{st}.t}$, and the identity key ipk, and the pre-key $prepk$. Next, the caller applies KDF_1 to k_1, k_2, and k_3, for deriving two update values $\mathsf{upd}^{\mathsf{ar}}$ and $\mathsf{upd}^{\mathsf{ur}}$.

If the caller switches its role from receiver to sender, i.e. the caller $\mathsf{st}.\mathsf{id}$ is $\mathtt{A}$ and the epoch $\mathsf{st}_{\mathtt{A}}.t$ is even or the caller is $\mathtt{B}$ and the epoch is odd, it first executes the following so-called *asymmetric ratchet* (ar) framework: First, the caller runs $\mathsf{eSend\text{-}Stop}(\mathsf{st})$ for a value l that counts the sent messages in the previous epoch, followed by incrementing the epoch counter $\mathsf{st}.t$ by 1 and initializing the message index counter $\mathsf{st}.i^{\mathsf{st}.t}$ to 0. Next, the caller samples a random string r, which together with the local NAXOS string $\mathsf{st}.nxs$ is applied to a key derivation function KDF_2, in order to produce a new NAXOS string, a KEM key generation randomness r^{KEM}, which is used to produce a new KEM key pair for receiving messages in the next epoch, and a DS key generation r^{DS}, which is used to produce a new DS key pair for sending messages in this epoch. The caller stores the private decapsulation keys and signing keys into the state. Then, the caller signs the pre-transcript for the ar framework $\mathsf{prtr}^{\mathsf{ar}}$, including the value l, the ciphertext c_1, c_2, and c_3, the newly sampled encapsulation key ek and the verification key vk, using the signing key produced two epochs earlier $\mathsf{st}.sk^{\mathsf{st}.t-2}$ for a signature σ^{ar}. The pre-transcript $\mathsf{prtr}^{\mathsf{ar}}$ and signature σ^{ar} are stored into the state $\mathsf{st.prtr}$. Finally, the caller forwards the ar framework by applying a KDF_3 to the root key $\mathsf{st}.rk$ and the update $\mathsf{upd}^{\mathsf{ar}}$ for deriving new root key and chain key $\mathsf{st}.ck^{\mathsf{st}.t}$.

Next, the caller executes the so-called *unidirectional ratchet* (ur) framework, no matter whether the ar framework is executed in this algorithm invocation or not: First, the caller forwards the unidirectional ratchet chain by applying a KDF_4 to the current chain key $\mathsf{st}.ck^{\mathsf{st}.t}$ for deriving next chain key and a unidirectional ratchet key urk. Next, the caller applies a KDF_5 to the unidirectional ratchet key urk and the update $\mathsf{upd}^{\mathsf{ur}}$ for the message key mk, followed by encrypting the message m by $c' \leftarrow \mathsf{SKE.Enc}(mk, m)$. Finally, the caller signs the pre-transcript $\mathsf{prtr}^{\mathsf{ur}}$ of the ur framework, including the epoch $\mathsf{st}.t$, the message index $\mathsf{st}.i^{\mathsf{st}.t}$, and the ciphertexts c', c_1, c_2, and c_3, for a signature σ^{ur} using the signing key $\mathsf{st}.sk^{\mathsf{st}.t}$. This algorithm outputs a new state st and a final ciphertext, which is a tuple of the ar pre-transcript and signature $\mathsf{st.prtr} = (\mathsf{prtr}^{\mathsf{ar}}, \sigma^{\mathsf{ar}})$, the ur pre-transcript $\mathsf{prtr}^{\mathsf{ur}}$, and the signature σ^{ur}.

$\mathsf{eRcv}(\mathsf{st}, ik, prek, c)$**:** The receiving algorithm inputs the (caller's) state st, private identity key ik and pre-key $prek$, and a ciphertext c, and does the mirror execution of eSend.

First, the caller parses the input ciphertext c into the pre-transcript and signature of ar framework $(\mathsf{prtr}^{\mathsf{ar}}, \sigma^{\mathsf{ar}})$, the unidirectional ratchet pre-transcript $\mathsf{prtr}^{\mathsf{ur}}$, and the signature σ^{ur}. Next, the caller further parses the pre-transcript $\mathsf{prtr}^{\mathsf{ar}}$ into one number l, three ciphertexts c_1, c_2, and c_3, an encapsulation key ek, and a verification key vk, and parses $\mathsf{prtr}^{\mathsf{ur}}$ into an epoch counter t, a message index counter i, and four ciphertexts c', c'_1, c'_2, and c'_3.

If the parsed epoch counter indicates a past epoch, i.e., $t \leq \mathsf{st}.t - 2$, the caller checks whether the maximal transmission length has been set (and not erased) and whether the parsed message index does not exceed the corresponding maximal transmission length. Then, the caller checks whether the parsed epoch counter is valid (by checking whether $\mathsf{st.id} = \mathtt{A}$ or $\mathtt{B}$ if the parsed epoch counter is even or odd) and in a meaningful range (by checking whether $t \leq \mathsf{st}.t + 1$). If any check is wrong, the eRcv aborts and outputs $m = \perp$.

If the parsed epoch counter t is the next epoch, i.e., $t = \mathsf{st}.t + 1$, the caller executes the asymmetric ratchet framework: The caller first checks whether the signature σ^{ar} is valid under the verification key $\mathsf{st}.vk^{t-2}$ and pre-transcript $\mathsf{prtr}^{\mathsf{ar}}$ and aborts if the check fails. Next, the caller invokes $\mathsf{eRcv\text{-}Max}(\mathsf{st}, l)$, records the transmission length l, and increments the epoch counter. Then, three keys k_1, k_2, and k_3 are respectively decapsulated from c_1, c_2, and c_3 using local keys $\mathsf{st}.dk^{\mathsf{st}.t}$, the private identity key ik, and pre-key $prek$. After that, the caller applies KDF_1 to above keys for update value $\mathsf{upd}^{\mathsf{ar}}$, which then together with the root key $\mathsf{st}.rk$ is applied to KDF_3 for a new root key and chain key $\mathsf{st}.ck^{\mathsf{st}.t}$. Finally, the caller initializes a dictionary $\mathcal{D}^{\mathsf{st}.t}_{urk}$ for storing the unidirectional ratchet keys in this epoch, sets the message counter $\mathsf{st}.i^{\mathsf{st}.t}$ to 0, and locally stores the encapsulation key for the next epoch and verification key for this epoch.

Then, the caller executes the unidirectional ratchet framework, no matter whether ar is executed in this algorithm invocation or not: First, the caller also checks whether the signature σ^{ur} is valid under the verification key $\mathsf{st}.vk^t$ and pre-transcript $\mathsf{prtr}^{\mathsf{ur}}$. Next, three keys k'_1, k'_2, and k'_3 are respectively decapsulated from c'_1, c'_2, and c'_3 using local keys $\mathsf{st}.dk^{\mathsf{st}.t}$, the private identity key ik, and pre-key $prek$. Then, the caller applies KDF_1 to above three keys for the update value $\mathsf{upd}^{\mathsf{ur}}$. After that, the caller continuously forwards the unidirectional ratchet chain, followed by storing the unidirectional ratchet keys into the dictionary and incrementing the message index by 1, until the local message index $\mathsf{st}.i^t$ reaches the parsed message index i. In the end, the caller reads the unidirectional ratchet key urk from the dictionary corresponding to the parsed message index, followed by erasing it from the dictionary. It must hold that $urk = \perp$ and aborts otherwise. The caller then derives the message key mk by applying KDF_5 to urk and the update $\mathsf{upd}^{\mathsf{ur}}$, and finally decrypts the message m from ciphertext c' using mk.

This algorithm outputs a new state st, the parsed epoch t and message index i, and the decrypted message m.

7.5.3 Security Conclusion and Concrete Instantiation

Theorem 28. *Let Π denote our* eSM *construction in Section 7.5.2. If the underlying* KEM *is δ_{KEM}-strongly correct*[5] *and $\epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}}$-secure,* DS *is δ_{DS}-strongly correct and $\epsilon^{\mathsf{suf\text{-}cma}}_{\mathsf{DS}}$-secure,* SKE *is δ_{SKE}-strongly correct and $\epsilon^{\mathsf{ind\text{-}1cca}}_{\mathsf{SKE}}$-secure, KDF_1 is $\epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1}$-secure*[6]*, KDF_2 is $\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_2}$ secure, KDF_3 is $\epsilon^{\mathsf{prf}}_{\mathsf{KDF}_3}$-secure, KDF_4 is $\epsilon^{\mathsf{prg}}_{\mathsf{KDF}_4}$-secure, KDF_5 is $\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5}$-secure, in time t, then Π is $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \epsilon)$-*eSM *secure for $\triangle_{\mathsf{eSM}} = 2$, where*

$$\begin{aligned}\epsilon \leq & (q_{\mathsf{ep}} + q)\delta_{\mathsf{DS}} + 3(q_{\mathsf{ep}} + q)\delta_{\mathsf{KEM}} + q\delta_{\mathsf{SKE}} + q_{\mathsf{ep}}\epsilon^{\mathsf{suf\text{-}cma}}_{\mathsf{DS}} \\ & + q^2_{\mathsf{ep}}q_{\mathsf{M}}(q+1)\epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}} + q_{\mathsf{ep}}(q_{\mathsf{M}}+2)q\epsilon^{\mathsf{ind\text{-}1cca}}_{\mathsf{SKE}} \\ & + q^2_{\mathsf{ep}}q_{\mathsf{M}}(q+1)\epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + q^2_{\mathsf{ep}}(q_{\mathsf{ep}}q + q_{\mathsf{ep}} + 1)\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_2} \\ & + q^2_{\mathsf{ep}}(q+1)\epsilon^{\mathsf{prf}}_{\mathsf{KDF}_3} + q_{\mathsf{ep}}q(q+1)\epsilon^{\mathsf{prg}}_{\mathsf{KDF}_4} \\ & + q_{\mathsf{ep}}(q_{\mathsf{ep}}q_{\mathsf{M}}q + q_{\mathsf{ep}}q_{\mathsf{M}} + 2q)\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5}\end{aligned}$$

[5]By strongly correct, we mean that the schemes are conventionally correct for all randomness. See Section 2.2.3, Section 2.2.5, and Section 2.2.6.

[6]By 3prf security, we mean that a function is indistinguishable from a random function w.r.t any of the three inputs. See Section 2.2.2.

Instantiation: We give the concrete instantiation for both classical and PQ settings. The deterministic DS can be instantiated with Ed25519 for classical setting, the formal analysis was given in [62], and the NIST suggested CRYSTALS-Dilithium for the PQ security, which is analyzed in [17]. A generic approach to instantiating KEM is to encrypt random strings using deterministic OW-CCA or merely OW-CPA secure PKE for strong correctness [26], [46]. The NIST suggested NTRU is also available for IND-CCA security and strong correctness [68]. The deterministic IND-1CCA secure authenticated encryption SKE can be instantiated with the Encrypt-then-MAC construction in [30]. The dual or prg-secure KDF_i for $i \in \{2, ..., 5\}$ can be instantiated with HMAC-SHA256 or HKDF. The 3prf-secure KDF_1 can be instantiated with the nested combination of any dual-secure function, as explained in Section 2.2.2. We suggest to double the security parameter of the symmetric primitives for PQ security.

7.6 Offline Deniability

As explained in Section 7.2.3, although the combinations of SPQR and ACD19 or our eSM achieve strong privacy and authenticity in the PQ setting, it is still an open question what flavors of offline deniability can be achieved by the combined protocols in the PQ setting. To address this, we extend the game-based offline deniability for asynchronous DAKE scheme Σ [63] to its combination with an eSM scheme Π.

Our offline deniability experiment is depicted in Figure 7.7. For the notational purpose, we use $\overline{ipk}$, $\overline{ik}$, $\overline{prepk}$, and $\overline{prek}$ to denote the public and private keys that are generated by DAKE construction Σ. The keys generated by eSM construction Π are notated without overline. The difference to the original model in [63, Definition 11], also see Definition 61 in Section 7.3, is highlighted with blue color.

In addition to message senders and receivers, the deniability experiment includes three new roles: accuser, defendant, and judge. For any two-party conversation, we call a party "accuser", whose identifier is denoted by aid, if it wants to accuse that its honest conversation partner has communicated with it. Correspondingly, we call the accused honest partner "defendant", whose identifier is denoted by did. The role of the "judge" in the experiment is performed by the attacker. The goal of the experiment is to ensure that no attackers (i.e., the real-life judges) can distinguish the real conversation transcripts between accusers and defendants from the fake ones that are produced by the accusers alone, given all secrets of all parties.

The experiment initializes a dictionary $\mathcal{D}_{\mathsf{session}}$, which records the identity of the parties in each session, and a session counter n with 0. Next, long-term identity and medium-term pre- public/private key pairs of Σ and Π are generated for all honest parties and provided to the attacker (e.g., the judge). A challenge bit $\mathsf{b} \in \{0, 1\}$ is randomly sampled.

```
$\mathrm{Expr}^{\mathsf{deni}}_{\Sigma,\Pi,q_{\mathsf{P}},q_{\mathsf{M}},q_{\mathsf{S}}}(\mathcal{A})$:
$\mathcal{D}_{\mathsf{session}}[\cdot] \leftarrow \bot,\ n \leftarrow 0$
$\mathcal{L}_{\mathsf{all}}, \mathcal{L}^{\overline{ipk}}_{\mathsf{all}}, \mathcal{L}^{\overline{prepk}}_{\mathsf{all}} \leftarrow \emptyset$
for $u \in [q_{\mathsf{P}}]$
    $\mathcal{L}^{\overline{prek}}_{u} \leftarrow \emptyset$
    $\mathcal{L}^{prek}_{u} \leftarrow \emptyset$
    $(\overline{ipk}_u, \overline{ik}_u) \xleftarrow{\$} \Sigma.\mathsf{IdKGen}()$
    $(ipk_u, ik_u) \xleftarrow{\$} \Pi.\mathsf{IdKGen}()$
    $\mathcal{L}^{\overline{ipk}}_{\mathsf{all}} \xleftarrow{+} \{\overline{ipk}_u\}$
    $\mathcal{L}_{\mathsf{all}} \xleftarrow{+} (\overline{ipk}_u, \overline{ik}_u)$
    $\mathcal{L}_{\mathsf{all}} \xleftarrow{+} (ipk_u, ik_u)$
    for $\mathsf{ind} \in [q_{\mathsf{M}}]$
        $(\overline{prepk}^{\mathsf{ind}}_u, \overline{prek}^{\mathsf{ind}}_u) \xleftarrow{\$} \Sigma.\mathsf{PreKGen}()$
        $(prepk^{\mathsf{ind}}_u, prek^{\mathsf{ind}}_u) \xleftarrow{\$} \Pi.\mathsf{PreKGen}()$
        $\mathcal{L}^{\overline{prek}}_{u} \xleftarrow{+} \overline{prek}^{\mathsf{ind}}_u$
        $\mathcal{L}^{\overline{prepk}}_{\mathsf{all}} \xleftarrow{+} \overline{prepk}^{\mathsf{ind}}_u$
        $\mathcal{L}^{prek}_{u} \xleftarrow{+} prek^{\mathsf{ind}}_u$
        $\mathcal{L}_{\mathsf{all}} \xleftarrow{+} (\overline{prepk}_u, \overline{prek}_u)$
        $\mathcal{L}_{\mathsf{all}} \xleftarrow{+} (prepk_u, prek_u)$
$\mathsf{b} \xleftarrow{\$} \{0,1\}$
$\mathsf{b}' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\mathcal{L}_{\mathsf{all}})$
return $[\![\mathsf{b} = \mathsf{b}']\!]$
```

```
$\mathsf{Session\text{-}Start}(\mathsf{sid}, \mathsf{rid}, \mathsf{aid}, \mathsf{did}, \mathsf{ind})$:
require $\{\mathsf{aid}, \mathsf{did}\} = \{\mathsf{sid}, \mathsf{rid}\}$ and $\mathsf{sid} \neq \mathsf{rid}$
$n{+}{+},\ \mathcal{D}_{\mathsf{session}}[n] \leftarrow \{\mathsf{sid}, \mathsf{rid}\}$
if $\mathsf{b} = 0$ or $\mathsf{aid} = \mathsf{sid}$
    $\pi_{\mathsf{rid}}.\mathsf{role} \leftarrow \mathtt{resp},\ \pi_{\mathsf{rid}}.\mathsf{st}_{\mathsf{exec}} \leftarrow \mathsf{running}$
    $\pi_{\mathsf{sid}}.\mathsf{role} \leftarrow \mathtt{init},\ \pi_{\mathsf{sid}}.\mathsf{st}_{\mathsf{exec}} \leftarrow \mathsf{running}$
    $(\pi'_{\mathsf{rid}}, m) \xleftarrow{\$} \Sigma.\mathsf{Run}(\overline{ik}_{\mathsf{rid}}, \mathcal{L}^{\overline{prek}}_{\mathsf{rid}}, \mathcal{L}^{\overline{ipk}}_{\mathsf{all}}, \mathcal{L}^{\overline{prepk}}_{\mathsf{all}}, \pi_{\mathsf{rid}}, (\mathsf{create}, \mathsf{ind}))$
    $(\pi'_{\mathsf{sid}}, m') \xleftarrow{\$} \Sigma.\mathsf{Run}(\overline{ik}_{\mathsf{sid}}, \mathcal{L}^{\overline{prek}}_{\mathsf{sid}}, \mathcal{L}^{\overline{ipk}}_{\mathsf{all}}, \mathcal{L}^{\overline{prepk}}_{\mathsf{all}}, \pi_{\mathsf{sid}}, m)$
    $(K, T) \xleftarrow{\$} (\pi'_{\mathsf{sid}}.K, (m, m'))$
else
    $(K, T) \xleftarrow{\$} \Sigma.\mathsf{Fake}(\overline{ipk}_{\mathsf{sid}}, \overline{ik}_{\mathsf{rid}}, \mathcal{L}^{\overline{prek}}_{\mathsf{rid}}, \mathsf{ind})$
if $\mathsf{b} = 0$
    $\mathsf{st}^{n}_{\mathsf{sid}} \xleftarrow{\$} \Pi.\mathsf{eInit\text{-}B}(K),\ \mathsf{st}^{n}_{\mathsf{rid}} \xleftarrow{\$} \Pi.\mathsf{eInit\text{-}A}(K)$
else
    $\mathsf{st}^{n}_{\mathsf{Fake}} \xleftarrow{\$} \mathsf{Fake}^{\mathsf{eInit}}_{\Pi}(K, ipk_{\mathsf{did}}, ik_{\mathsf{aid}}, \mathcal{L}^{prek}_{\mathsf{aid}}, \mathsf{sid}, \mathsf{rid}, \mathsf{aid}, \mathsf{did})$
return $T$
$\mathsf{Session\text{-}Execute}(\mathsf{sid}, \mathsf{rid}, i, \mathsf{ind}, m)$:
require $\mathcal{D}_{\mathsf{session}}[i] = \{\mathsf{sid}, \mathsf{rid}\}$
if $\mathsf{b} = 0$
    $(\mathsf{st}^{i}_{\mathsf{sid}}, c) \xleftarrow{\$} \Pi.\mathsf{eSend}(\mathsf{st}^{i}_{\mathsf{sid}}, ipk_{\mathsf{rid}}, prepk^{\mathsf{ind}}_{\mathsf{rid}}, m)$
    $(\mathsf{st}^{i}_{\mathsf{rid}}, \_, \_, \_) \leftarrow \Pi.\mathsf{eRcv}(\mathsf{st}^{i}_{\mathsf{rid}}, ik_{\mathsf{rid}}, prek^{\mathsf{ind}}_{\mathsf{rid}}, c)$
else
    $(\mathsf{st}^{i}_{\mathsf{Fake}}, c) \xleftarrow{\$} \mathsf{Fake}^{\mathsf{eSend}}_{\Pi}(\mathsf{st}^{i}_{\mathsf{Fake}}, ipk_{\mathsf{rid}}, prepk^{\mathsf{ind}}_{\mathsf{rid}}, m, \mathsf{sid}, \mathsf{rid}, \mathsf{ind})$
return $c$
```

Figure 7.7: The offline deniability experiment for an attacker $\mathcal{A}$ against the combination of a DAKE scheme Σ and an eSM scheme Π. The experiment $\mathrm{Expr}^{\mathsf{deni}}_{\Sigma,\Pi,q_{\mathsf{P}},q_{\mathsf{M}},q_{\mathsf{S}}}$ is parameterized the maximal numbers of parties q_{P}, pre-keys per party q_{M}, and total sessions q_{S}. We highlight the difference to the offline deniability experiment for DAKE in Definition 61 with blue color.

The attacker (i.e., the judge) is given repeated access to the following two oracles: The Session-Start oracle initializes a session between a sender sid and a receiver rid and determines that the party $\mathsf{aid} \in \{\mathsf{sid}, \mathsf{rid}\}$ plays the role of accuser in this session and the other party $\mathsf{did} \in \{\mathsf{sid}, \mathsf{rid}\}$ plays the role of defendant in this session. This oracle executes a real session setup and real eSM initialization if $\mathsf{b} = 0$, and some fake algorithms that simulate the accuser's view if $\mathsf{b} = 1$. The Session-Execute forwards the interaction in an existing session one step: this oracle executes eSM algorithm for sending and receiving one message honestly if $\mathsf{b} = 0$, and some fake algorithms that simulate the accuser's view if $\mathsf{b} = 1$. The attacker wins if it can distinguish real conversation transcripts (i.e., $\mathsf{b} = 0$) from fake transcripts that simulate accusers' views (i.e., $\mathsf{b} = 1$). We say a full messaging protocol is offline deniable, if there exist fake algorithms that prevent all attackers from winning the offline deniability experiment in polynomial time. By this, we ensure that if a protocol is offline deniable, then no judge can decide whether a transcript given by the accuser is the real transcript of the conversation with the defendant or produced by the accuser alone.

Oracle $\mathsf{Session\text{-}Start}(\mathsf{sid}, \mathsf{rid}, \mathsf{aid}, \mathsf{did}, \mathsf{ind})$***:*** This oracle inputs are a sender identity sid, a receiver identity rid, an accuser identity aid, a defendant identity did, and a pre-key index ind. This oracle first checks whether the sender identity and the receiver identity are distinct and whether either the sender is the accuser and the receiver is the defendant or another way around. Next, the session counter n is incremented by 1 and the set of the sender identity sid and the receiver identity rid is set to $\mathcal{D}_{\mathsf{session}}[i]$. Then, it simulates the honest DAKE execution if the challenge bit is 0 or the accuser is the sender. Otherwise, it runs the fake algorithm $\Sigma.\mathsf{Fake}$. In both cases, a key K and a transcript T are derived. In the end, if the challenge bit is 0, then the oracle honestly runs $\Pi.\mathsf{eInit\text{-}A}(K)$ and $\Pi.\mathsf{eInit\text{-}B}(K)$ on the shared key K to produce the state $\mathsf{st}_{\mathsf{sid}}^n$ and $\mathsf{st}_{\mathsf{rid}}^n$. Otherwise, the oracle runs a function $\mathsf{Fake}_\Pi^{\mathsf{eInit}}(K, ipk_{\mathsf{did}}, ik_{\mathsf{aid}}, \mathcal{L}_{\mathsf{aid}}^{prek}, \mathsf{sid}, \mathsf{rid}, \mathsf{aid}, \mathsf{did})$ to produce a fake state $\mathsf{st}_{\mathsf{Fake}}^n$. The transcript T is returned.

Oracle $\mathsf{Session\text{-}Execute}(\mathsf{sid}, \mathsf{rid}, i, \mathsf{ind}, m)$***:*** This oracle inputs a sender identity sid, a receiver identity rid, a session index i, a pre-key index ind, and a message m. This oracle first checks whether the session between sid and rid has been established by requiring $\mathcal{D}_{\mathsf{session}}[i] = \{\mathsf{sid}, \mathsf{rid}\}$. Next, if the challenge bit is 0, this oracle simulates the honest transmission of message m. Otherwise, this oracle produces a ciphertext c by running a function $\mathsf{Fake}_\Pi^{\mathsf{eSend}}$ on the fake state $\mathsf{st}_{\mathsf{Fake}}^i$, the receiver's public identity key ipk_{rid}, pre-key $prepk_{\mathsf{rid}}^{\mathsf{ind}}$, the message m, and sender identity sid, the receiver identity rid, and a pre-key index ind. In both cases, the ciphertext c is returned.

We stress that our offline deniability model is a significant extension to the one for DAKE in [63]. First, our model also allows the attacker (e.g. the judge) to obtain *all* initial private secret of all parties, as in [63].

Second, while the model in [63] prevents an attacker from deciding the challenge bit b given the (output) shared keys and the transcripts of DAKE key establishments, our model prevents an attacker from deciding b given the transcripts of full conversations, which include the one of DAKE and the one of eSM inputting the shared key of DAKE. This extension follows the idea behind the simulation-based extension [175].

Third, the accuser in the model for DAKE in [63] must play the role of a responder `resp` (i.e., the receiver rid during the key establishment) rather than an initiator (i.e., the sender sid during the key establishment), since the $\Sigma.\mathsf{Fake}$ algorithm is only defined on the responder's behalf. The main reason behind is that all transcripts in a DAKE scheme are produced by the initiator alone. However, the responder producing no output during the key establishment might produce some transcripts afterwards. To capture this, our model also allows the accuser to be the initiator `init` in the whole conversation. In fact, our $\mathsf{Session\text{-}Execute}$ simulates the accuser's view (when $\mathsf{b} = 1$) by running the Fake_Π algorithm that simulates the stateful execution of either the initiator or the responder, depending on whether $\mathsf{aid} = \mathsf{sid}$ or rid in the corresponding Session-Start query[7].

Definition 69. *We say the composition of a* DAKE *scheme* Σ *and an* eSM *scheme* Π *is* $(t, \epsilon, q_{\mathsf{P}}, q_{\mathsf{M}}, q_{\mathsf{S}})$*-deniable, if two functions* $\mathsf{Fake}_{\Pi}^{\mathsf{eInit}}$ *and* $\mathsf{Fake}_{\Pi}^{\mathsf{eSend}}$ *exist such that the below defined advantage for any attacker* $\mathcal{A}$ *in time* t *is bounded by*

$$\mathsf{Adv}_{\Sigma,\Pi,q_{\mathsf{P}},q_{\mathsf{M}},q_{\mathsf{S}}}^{\mathsf{deni}}(\mathcal{A}) := |\mathrm{Expr}_{\Sigma,\Pi,q_{\mathsf{P}},q_{\mathsf{M}},q_{\mathsf{S}}}^{\mathsf{deni}}(\mathcal{A}) - \frac{1}{2}| \leq \epsilon$$

where q_{P}*,* q_{M}*, and* q_{S} *respectively denote the maximal number of parties, of pre-key per party, and total sessions in the* $\mathrm{Expr}_{\Sigma,\Pi,q_{\mathsf{P}},q_{\mathsf{M}},q_{\mathsf{S}}}^{\mathsf{deni}}$ *in Figure 7.7.*

Theorem 29. *Let* Σ *denote a* DAKE *scheme and* Π *denote our* eSM *construction in Section 7.5.2. If* Σ *is* (t, ϵ, q)*-deniable (with respect to any* q_{P}*,* q_{M}*) in terms of the Definition 61 , then the composition of* Σ *and* Π *is* $(t, \epsilon, q_{\mathsf{P}}, q_{\mathsf{M}}, q)$*-deniable.*

Proof Sketch. We define $\mathsf{Fake}_{\Pi}^{\mathsf{eInit}}$ algorithm as running both eInit-A and eInit-B upon the input K and storing all other inputs. We define $\mathsf{Fake}_{\Pi}^{\mathsf{eSend}}$ algorithm as honest execution of Π.eSend upon sender sid followed by Π.eRcv upon the receiver rid and the ciphertext of Π.eSend. If the attacker cannot distinguish the real DAKE transcripts and output keys from the fake ones, then it cannot distinguish the real DAKE and eSM (and therefore the full) transcripts from the fake ones. We give the full proof in Section 7.11.7. $\square$

7.7 Review of ACD19 and TR protocols

The* ACD19 *protocol [12, Section 5.1]: The ACD19 protocol is an instance of the SM scheme and can be further modularized into three building blocks: the *Continuous Key Agreement* (CKA), where the sender exchanges its randomness with the partner; the *Forward-Secure Authenticated Encryption with Associated Data* (FS-AEAD), where the sender sends messages to the recipient and updates the shared state in a deterministic manner, which provides forward secrecy and immediate decryption; the PRF-PRNG refreshes its inherent shared state by using the randomness of provided by CKA and initializes a new FS-AEAD thread, which provides the post-compromise security.

The ACD19 protocol is managed according to the epoch, which is used to describe how many interactions in a two-party communication channel have been processed. The behavior of a party (assume A) for sending messages is different when A enters a new epoch or not:

1. When a receiver A switches to sender and sends the first message in a new epoch, A first counts and remembers how many messages have been sent in the last epoch using the corresponding FS-AEAD thread, which is then erased. Next, A increments the inherent

[7]In our model, we restrict the behavior of the accuser, who acts as initiator, to be honest during the key establishment phase, see Line 24. We leave a stronger model without this restriction as future work.

epoch counter by 1. Then, A invokes the sending algorithm of the CKA component for exchanging the randomness with the partner B. The output of CKA algorithm in this epoch is also remembered locally. Afterwards, A refreshes the shared state using PRF-PRNG and initialize a new FS-AEAD thread for the new epoch.

2. Regardless of whether A is sending the first message in a new epoch (after executing the above step) or sending subsequent messages in the current epoch, A uses the current FS-AEAD thread for the encrypting real message with header: the number of messages sent two epoch earlier, the output of CKA in this epoch, the current epoch counter.

The receiving process is defined in the reverse way. When a sender (assume B) receives a message indicating the next epoch, B switches his role to receiver and enters the next epoch by incrementing the internal epoch counter. Notably, B parses and locally remembers the number of messages sent two epochs earlier from the received ciphertext and erases the FS-AEAD thread once these messages arrived at B.

Moreover, several different instantiations of CKA, FS-AEAD, and PRF-PRNG components are also given in [12].

The* TR *protocol [50, Section 5.1]: The Triple Ratchet (TR) is very close to the ACD19 construction in [12], except for the following two differences:

1. When a party switches its role from receiver to sender, it does not count and remember how many messages have been sent in the last epoch. Instead, this step is executed in the receiving algorithm when a party enters a new epoch and switches its role from sender to receiver.

2. The underlying CKA component must be instantiated with a customized CKA+ construction, which provides better privacy against randomness leakage but relies on a non-standard assumption and a random oracle. Note that CKA is a generic building block, while CKA+ is a concrete instantiation. The other building blocks such as FS-AEAD and PRF-PRNG can be instantiated with the constructions in [12].

For the interested readers, we also compare ACD19 and TR with our protocol in Section 7.10.

7.8 Review on Messaging Protocols with Various Optimal Security

The "optimal" protocols by Jäger and Stepanovs [118] and by Pöttering Rösler [154], the "sub-optimal" protocol by Durak and Vaudenay [85], and a novel protocol by Pijnenburg and Pöttering [153] (we call "ID-optimal"), all are PQ compatible. The "almost-optimal" protocol by Jost, Maurer, and Mularczyk [122] only has classically secure instantiation. Technically, they follow different ratcheting frameworks:

(1) "optimal" Jäger-Stepanovs protocol [118]: In the Jäger-Stepanovs protocol, all cryptographic building blocks except the hash functions, such as PKE and DS, are asymmetric and updatable. When Alice continuously sends messages to Bob, the next encryption key is deterministically derived from an encryption key included in the last reply from Bob and all past transcript since the last reply from Bob. On the one hand, this protocol enjoys high security guarantee against impersonation due to the asymmetric state. On the other hand, this protocol has no message-loss resilience, namely, if one message from Alice to Bob is lost, then Bob cannot decrypt subsequent messages anymore. In particular, no instantiation with constant bandwidth in the post-quantum setting is available.

(2) "optimal" Pöttering-Rösler protocol [154]: In the Pöttering-Rösler protocol, both asymmetric and symmetric primitives, including updatable KEM, DS, MAC are employed. When Alice sends messages to Bob, she first runs the encapsulations upon the one or more KEM public keys depending on her behavior. If Alice is sending a reply, then she needs to run the encapsulation upon all accumulated KEM public keys that are generated and signed by Bob. Otherwise, she only needs one KEM public key that was generated by herself when sending the previous message. After that, Alice derives the symmetric key for message encryption from the symmetric state and the encapsulated keys. This protocol enjoys *state healing* when continuously sending messages. Any unpredictable randomness at some point can heal Alice's state from corruption when she continuously sends messages. However, this protocol has no message-loss resilience: If one message is lost in the transmission, the both parties' symmetric states that are used for key update mismatch. This means, all subsequent messages cannot be correctly recovered by the recipient.

(3) "sub-optimal" Durak-Vaudenay protocol [85]: In contrast to the above two "optimal" approaches, the Durak-Vaudenay protocol does not employ any key updatable components and has a substantially better time complexity. When Alice sends messages to Bob, she samples several fragments of a symmetric key and encrypts them using signcryption with the accumulated sender keys, where the sender keys are generated either by herself or by Bob depending on whether Alice is continuously sending messages or sending a reply. The Durak-Vaudenay protocol is similar to Pöttering-Rösler but is less reliant on the state. Any randomness leakage corrupts the next message. Moreover, both the message and the receiver key that is used for receiving or sending next message, are encrypted under the symmetric key. This implies that the protocol does not have message-loss resilience: If one message is lost in the transmission (from either Alice or Bob), the communication session is aborted.

(4) "almost-optimal" Jost-Maurer-Mularczyk protocol [122]: The Jost-Maurer-Mularczyk protocol aims at stronger security than what is achieved by Signal, but slightly weaker than optimal security proposed in Jäger-Stepanovs' and Pöttering-Rösler's work, yet its efficiency is closer to that of Signal. The Jost-Maurer-Mularczyk protocol employs two customized novel schemes: healable and key-updating encryption (HkuPke) and key-updating signatures (KuSig). When Alice sends messages to Bob, Alice first samples two DS key pairs, while the one is used by Alice for sending next continuous message, the other is used by Bob for sending the reply. Next, Alice updates the key of HkuPke and encrypts the message as well as the private DS signing key for Bob. Then, Alice signs the transcript and her next DS verification key twice, by using KuSig and DS. Finally, the state is updated. Note that the sender has to send the next DS signing and verification keys to the partner. If one message is lost in the transmission (from either Alice or Bob), the receiver can neither verify the next message from the partner nor send a valid reply to the partner – the communication session becomes stuck.

Moreover, Jost-Maurer-Mularczyk's HkuPke construction uses a customized secretly key-updatable encryption (SkuPke), the only known instantiation of which relies on the Diffie-Hellman exchange, for which currently no PQ-secure instantiation is available.

(5) "ID-optimal" Pijnenburg-Pöttering protocol [153]: The Pijnenburg-Pöttering protocol aims to solve the weak forward secrecy caused by the immediate decryption by definition. In principle, the immediate decryption requires every receiver to be able to decrypt a ciphertext at the time of arrival. Thus, if an attacker can intercept a message and corrupt the receiver's state in the future, the attacker can always recover the plaintext from the intercepted ciphertext.

To solve this, the Pijnenburg-Poettering protocol employs three updatable mechanisms: Updatable Signature Schemes (USS), Key-Evolving KEM (KeKEM), and Key-Updatable KEM (KuKEM). Unlike all above protocols, while keys of the KuKEM and USS schemes are updated whenever a party switches the role from receiver to sender, the keys of KeKEM are updated every certain time interval. If a past message does not arrive at the receiver, the receiver still stores the corresponding decryption keys for the decryption at the time of message arrival, however, but only within a fixed length of time. After a pre-defined time interval, the corresponding decryption keys are expired and cleaned from the local state. By this, the compromise of a party's state does not cause the message leakage that is sent long time ago.

In particular, none of these protocols provide immediate decryption with constant-size overhead.

7.9 Security Model Comparison between our eSM and SM in [12]

Our eSM model extends the SM model [12], with the following main differences.

Extended Syntax. Compared to the original SM definition [12], eSM has two additional algorithms IdKGen and $\mathsf{PreKGen}$: IdKGen outputs the public-private identity key, which is fixed once generated, and $\mathsf{PreKGen}$ outputs pre-key pairs, which are updated regularly (similar to $\mathsf{X3DH}$). The generated identity and pre-key pairs both are used in the eSend and eRcv algorithms for sending and receiving messages.

More Expected Security Properties. Our eSM is expected to preserve all basic properties of the SM schemes in [12], including correctness, immediate decryption, FS, and PCS. Moreover, our eSM targets the stronger authenticity and privacy than SM in [12]. In particular, the authenticity and privacy in [12] hold only when neither parties' states are compromised. Instead, we aim for stronger authenticity and privacy against more fine-grained state compromise. This potentially indicates that our eSM achieves stronger randomness leakage/failures property. Finally, our eSM also aims at two new properties: state compromise/failures and PPR, which are not captured by SM in [12].

Stronger Security Model. Our eSM model is more complicated than the SM model [12] from many aspects. First, our eSM model needs more variables that are related to the identity keys and pre-keys, which are excluded in [12], such as $\mathsf{safe}_\mathtt{P}^{\mathsf{idK}}$, $\mathcal{L}_\mathtt{P}^{\mathsf{rev}}$, and $n_\mathtt{P}$, for $\mathtt{P} \in \{\mathtt{A}, \mathtt{B}\}$. We also import two new sets $\mathsf{allTrans}$ and $\mathsf{allChall}$ to simplify the security analysis of the benefits obtained from using the identity keys and pre-keys. Besides, we use two lists $\mathcal{L}_\mathtt{A}^{\mathsf{cor}}$ and $\mathcal{L}_\mathtt{B}^{\mathsf{cor}}$ to capture the state corruption of either party instead of using a single counter. While splitting the single state corruption variable into two helps our model to capture our strong privacy and strong authentication, using lists but not a counter additionally simplifies the definition of the safe state predicate.

Second, we define two new safe predicates $\mathsf{safe}_\mathtt{P}^{\mathsf{preK}}$ and $\mathsf{safe\text{-}st}_\mathtt{P}$, which respectively capture the safety of the the pre-key and session state. The $\mathsf{safe\text{-}ch}_\mathtt{P}$ and $\mathsf{safe\text{-}inj}_\mathtt{P}$ predicates were introduced in [12]. However, our eSM model defines them in a different way: Compared to [12], our $\mathsf{safe\text{-}ch}_\mathtt{P}$ predicates additionally input a randomness quality, a epoch number, and a pre-key index. While the $\mathsf{safe\text{-}ch}_\mathtt{P}$ predicate in [12] equals the condition 1, our new conditions 2, 3, and 4 respectively capture the strong privacy, state compromise/failures, and PPR security properties. Moreover, our $\mathsf{safe\text{-}inj}_\mathtt{P}$ additionally inputs an epoch number t.

We stress that our safe requirements are more relaxed and allow to reveal more information than in [12] (even when removing the usage of identity keys and pre-keys). In particular, if a safe predicate in the SM security model in [12] is true, then the one in our eSM model is true, but the reserve direction does not always hold.

Third, our eSM model has one new helper function $\mathsf{corruption\text{-}update}$. The other four helper functions in our eSM model are introduced in [12], but are defined with slight differences due to our new notations.

Finally, our eSM model includes 8 new oracles that are not included in SM [12]. The new oracles are related to the identity keys and pre-keys. Besides, the other 8 oracles for message transmissions are identical to the one in SM model, except for the notation differences. The only oracles that have huge differences with the ones in SM model are state corruption oracles: While our corrupt oracles requires any of three conditions holds: (1) the chall does not include the record produced by the partner $\neg$P, (2) the flag in the record is good and P's identity key is safe, and (3) the flag in the record is good and P's pre-key corresponding to the pre-key index in the record is safe, the ones in SM model only require the condition (1). After that, the corruption oracle in our eSM model adds all records rec $\in$ trans, which are produced by $\neg$P at an unsafe epoch t (but not all epochs as in [12]), into the compromise set comp.

Compared to [12], the corruption oracles in our model can be queried under weaker requirements, providing the attacker with more information. Moreover, our corruption oracles set fewer records into the compromise set, which enables the attacker to forge ciphertexts for more epochs.

Conclusion. Even without taking the use of identity keys and pre-keys into account, our security model is strictly stronger than the one in [12].

7.10 Comparison of our eSM Construction with ACD19 and TR

Although our eSM construction in Section 7.5.2, the ACD19 in [12], and the TR construction in [50], all satisfy immediate decryption with constant bandwidth consumption, their designs differ in many details.

***Comparison between our* eSM *construction and* ACD19:** The ACD19 protocol in [12, Section 5.1] employs three underlying modules: CKA, FS-AEAD, and PRF-PRNG. While the CKA employs the asymmetric cryptographic primitives, such as KEM or Diffie-Hellman exchange, the FS-AEAD and PRF-PRNG only employ symmetric cryptographic primitives, such as AEAD, PRF, PRG. In particular, the FS-AEAD deterministically derives the symmetric keys for encrypting messages and decrypting ciphertexts from the state, which is shared by both parties. Besides, they provide several CKA instantiations and all of them sample the asymmetric key pairs only using the ephemeral randomness. Moreover, their construction does not rely on any material outside the session state. Thus, it is obvious that the leakage of either state will trigger the loss of the privacy and authenticity.

Compared to the ACD19, our eSM construction has the differences mainly from following three aspects: First, the asymmetric primitives are used in every sending or receiving execution. In particular, our construction uses the KEM and DS keys across our asymmetric ratchet (ar) and unidirectional ratchet (ur) frameworks. Although this stops the further modularization of our eSM construction, the deployment of the KEM and DS provides better performance in terms of the strong privacy and strong authenticity, since the leakage of sender's (resp. receiver's) state does not indicate the compromise of the decapsulation key (resp. signing key) and preserves the privacy (resp. authenticity).

Second, our construction makes use of the identity keys and pre-keys, which also provide benefits in terms of strong privacy, state compromise/failure, and PPR. If the corruption of a device's full state without secure environment is not noticed by the owner (which is the common real-world scenario), the privacy for subsequent messages from the partner is lost until the corruption party sends a reply. The use of pre-key provides mitigation in this scenario as the pre-key is updated every certain period in the back-end without the active behavior of the corrupted party. Moreover, if the device has a secure environment such as an HSM, storing identity keys into the HSM provides even stronger security guarantees, as we explained in Section 7.4.3.

Finally, our construction implicitly uses three kinds of NAXOS-like tricks for strong privacy. (1) First, the symmetric root key together with ephemeral randomness is used for deriving new shared state when sending the first message in each epoch, this is same as in ACD19. (2) Second, the NAXOS string $\mathsf{st}.nxs$ (in the sender's state) together with the ephemeral randomness is used for improving the key generation when sending the first message in each epoch. (3) Third, the unidirectional ratchet keys (derived from the shared state) together with the ephemeral randomness are used to derive the real message keys. We stress the second and third NAXOS tricks provide additional benefits to our construction when comparing with ACD19. On the one hand, bad randomness quality of a party when sending the first message in a new epoch will cause leakage of the private KEM key in ACD19, but not in our construction. In this case, the corruption of the partner in the next epoch will cause the loss of privacy in ACD19, but not in our construction, due to the second NAXOS trick. On the other hand, the message keys are derived from not only the mere state but also ephemeral randomness. The third NAXOS trick together with the usage of identity keys and pre-keys provide stronger privacy against state corruption attacks.

As an aside, we observe that the CKA instantiation based on LWE (Frodo) does *not* provide correctness: CKA-correctness requires both parties to always output the same key, even if the attacker controls the randomness. Since LWE based Frodo includes an error that needs to be reconciled during the decapsulation, the attacker can always pick bad randomness to prevent the correct reconciliation. Instead, our construction is provably correct in the post-quantum setting, if the underlying KEM satisfies strong correctness, as explained in Section 7.5.3.

***Comparison between our* eSM *construction and* TR**: The TR construction in [50, Section 5.1] is very close to the one in ACD19 except for two differences: (1) The FS-Stop function of the underlying FS-AEAD components is invoked when receiving the first message in a new epoch but not sending. (2) The underlying CKA component must be instantiated with a new customized CKA+ construction based on a Diffie-Hellman exchange. The state of CKA+ component does not merely rely on the randomness but also on the past state. This can be seen as a variant of the NAXOS trick.

Compared to the TR construction, our eSM construction mainly differs in four aspects: First, our construction employs generic KEMs aiming at post-quantum compatibility, while TR makes use of a concrete Diffie-Hellman exchange, which is vulnerable to quantum attacks.

Second, while TR and our constructions both use the root key for a NAXOS trick, the NAXOS trick for improving privacy of the KEM key pairs is different. While TR uses a tailored CKA+ construction assuming a non-standard StDH and random oracles, our construction uses a local NAXOS string $\mathsf{st}.nxs$ only assuming the dual security of the function KDF_2, the generic constructions of which based solely on standard assumptions are given in [29].

Third, TR and our construction both prevent an attacker from corrupting the receiver in the current epoch and forging a ciphertext corresponding to the previous epoch to the partner by erasing a party's state for sending messages once a message from the partner for the next epoch arrives. Note that this attack is effective against ACD19, as the attacker can in the current epoch corrupt the FS-AEAD thread corresponding to the previous epoch and use it to encrypt the forged message. The only difference Due to the immediate decryption property, the forged ciphertext must be correctly decrypted. The TR construction prevents this attack by invoking FS-Stop function when receiving the first message in a new epoch to erase the chain key for sending in the previous epoch. In contrast, our construction prevents this attack by erasing both the chain key and the KEM encapsulation key for sending in the old epoch in the eRcv-Max function.

Fourth, the remaining benefits of our construction in comparison to ACD19 also apply to the comparison with TR, including strong privacy, strong authenticity, PPR, the resilience to a novel forgery attack.

7.11 Full Proofs

7.11.1 Our Lemmas

Lemma 2. *Let* Π *be an* eSM *scheme that is*

- $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \epsilon_{\Pi}^{\mathsf{CORR}})$-CORR *secure,*
- $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \epsilon_{\Pi}^{\mathsf{AUTH}})$-AUTH *secure, and*
- $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \epsilon_{\Pi}^{\mathsf{PRIV}})$-PRIV *secure*

Then, it is also $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \epsilon)$-eSM *secure, where*

$$\epsilon \leq \epsilon_{\mathsf{eSM}}^{\mathsf{CORR}} + q_{\mathsf{ep}}(\epsilon_{\mathsf{eSM}}^{\mathsf{AUTH}} + \epsilon_{\mathsf{eSM}}^{\mathsf{PRIV}})$$

Lemma 3. *Let* Π *denote our* eSM *construction in Section 7.5.2. If the underlying* KEM*,* DS*, and* SKE *are respectively* δ_{KEM}*,* δ_{DS}*,* δ_{SKE}*-strongly correct*[8] *in time* t*, then* Π *is* $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \epsilon_{\mathsf{eSM}}^{\mathsf{CORR}})$-CORR *secure for* $\triangle_{\mathsf{eSM}} = 2$*, such that*

$$\epsilon_{\mathsf{eSM}}^{\mathsf{CORR}} \leq (q_{\mathsf{ep}} + q)\delta_{\mathsf{DS}} + 3(q_{\mathsf{ep}} + q)\delta_{\mathsf{KEM}} + q\delta_{\mathsf{SKE}}$$

Lemma 4. *Let* Π *denote our* eSM *construction in Section 7.5.2. If the underlying* KEM *is* $\epsilon_{\mathsf{KEM}}^{\mathsf{ind\text{-}cca}}$*-secure,* SKE *is* $\epsilon_{\mathsf{SKE}}^{\mathsf{ind\text{-}1cca}}$*-secure,* KDF_1 *is* $\epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}}$*-secure*[9]*,* KDF_2 *is* $\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}}$ *secure,* KDF_3 *is* $\epsilon_{\mathsf{KDF}_3}^{\mathsf{prf}}$*-secure,* KDF_4 *is* $\epsilon_{\mathsf{KDF}_4}^{\mathsf{prg}}$*-secure,* KDF_5 *is* $\epsilon_{\mathsf{KDF}_5}^{\mathsf{dual}}$*-secure, in time* t*, then* Π *is* $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \epsilon_{\mathsf{eSM}}^{\mathsf{PRIV}})$-PRIV *secure for* $\triangle_{\mathsf{eSM}} = 2$*, such that*

$$\begin{aligned}\epsilon_{\mathsf{eSM}}^{\mathsf{PRIV}} \leq & q_{\mathsf{M}} q_{\mathsf{ep}} q \epsilon_{\mathsf{KEM}}^{\mathsf{ind\text{-}cca}} + q_{\mathsf{M}} q \epsilon_{\mathsf{SKE}}^{\mathsf{ind\text{-}1cca}} + q_{\mathsf{M}} q_{\mathsf{ep}} q \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} \\ & + q_{\mathsf{ep}}^2 q \epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}} + q_{\mathsf{ep}} q \epsilon_{\mathsf{KDF}_3}^{\mathsf{prf}} + q^2 \epsilon_{\mathsf{KDF}_4}^{\mathsf{prg}} + (q_{\mathsf{M}} q_{\mathsf{ep}} + 1) q \epsilon_{\mathsf{KDF}_5}^{\mathsf{dual}}\end{aligned}$$

Lemma 5. *Let* Π *denote our* eSM *construction in Section 7.5.2. If the underlying* DS *is* $\epsilon_{\mathsf{DS}}^{\mathsf{suf\text{-}cma}}$*-secure,* KEM *is* $\epsilon_{\mathsf{KEM}}^{\mathsf{ind\text{-}cca}}$*-secure,* SKE *is* $\epsilon_{\mathsf{SKE}}^{\mathsf{ind\text{-}1cca}}$*-secure,* KDF_1 *is* $\epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}}$*-secure,* KDF_2 *is* $\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}}$ *secure,* KDF_3 *is* $\epsilon_{\mathsf{KDF}_3}^{\mathsf{prf}}$*-secure,* KDF_4 *is* $\epsilon_{\mathsf{KDF}_4}^{\mathsf{prg}}$*-secure,* KDF_5 *is* $\epsilon_{\mathsf{KDF}_5}^{\mathsf{dual}}$*-secure, in time* t*, then* Π *is* $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \epsilon_{\mathsf{eSM}}^{\mathsf{AUTH}})$-AUTH *secure for* $\triangle_{\mathsf{eSM}} = 2$*, such that*

$$\begin{aligned}\epsilon_{\mathsf{eSM}}^{\mathsf{AUTH}} \leq & \epsilon_{\mathsf{DS}}^{\mathsf{suf\text{-}cma}} + q_{\mathsf{ep}} q_{\mathsf{M}} \epsilon_{\mathsf{KEM}}^{\mathsf{ind\text{-}cca}} + 2q \epsilon_{\mathsf{SKE}}^{\mathsf{ind\text{-}1cca}} \\ & + q_{\mathsf{ep}} q_{\mathsf{M}} \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + q_{\mathsf{ep}}(q_{\mathsf{ep}} + 1) \epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}} + q_{\mathsf{ep}} \epsilon_{\mathsf{KDF}_3}^{\mathsf{prf}} \\ & + q \epsilon_{\mathsf{KDF}_4}^{\mathsf{prg}} + (q_{\mathsf{ep}} q_{\mathsf{M}} + q) \epsilon_{\mathsf{KDF}_5}^{\mathsf{dual}}\end{aligned}$$

7.11.2 Proof of Theorem 28

Proof. Our proof is divided into two steps: First, we introduce four lemmas in Section 7.11.1. Lemma 2 reduces the eSM security to the simplified security notions, the full proof of which is given in Section 7.11.3. Lemma 3, Lemma 4, and Lemma 5 respectively proves the simplified correctness, privacy, and authenticity of our eSM construction in Section 7.5.2, the full proof of which are given in Section 7.11.4, Section 7.11.5, and Section 7.11.6. Second, the proof is concluded by combining the above four lemmas together. □

[8] By strongly correct, we mean that the schemes are conventionally correct for all randomness. See Section 2.2.3, Section 2.2.5, and Section 2.2.6 for more details.

[9] By 3prf security, we mean that a function is indistinguishable from a random function with respect to any of the three inputs. See Section 2.2.2 for mode details.

7.11.3 Proof of Lemma 2

Proof. The proof is conducted by case distinction. Let $\mathcal{A}$ denote an attacker that breaks $\mathrm{Expr}_{\Pi,\triangle_{\mathsf{eSM}}}^{\mathsf{eSM}}$ security of an eSM scheme Π with respect to the parameter $\triangle_{\mathsf{eSM}}$. Recall that the advantage of $\mathcal{A}$ in winning $\mathrm{Expr}_{\Pi,\triangle_{\mathsf{eSM}}}^{\mathsf{eSM}}$ experiment is defined as:

$$\begin{aligned}\mathsf{Adv}_{\Pi,\triangle_{\mathsf{eSM}}}^{\mathsf{eSM}}(\mathcal{A}) = \max \Big(& \Pr[\mathrm{Expr}_{\Pi,\triangle_{\mathsf{eSM}}}^{\mathsf{eSM}}(\mathcal{A}) = (1,0,0)], \\ & \Pr[\mathrm{Expr}_{\Pi,\triangle_{\mathsf{eSM}}}^{\mathsf{eSM}}(\mathcal{A}) = (0,1,0)], \\ & |\Pr[\mathrm{Expr}_{\Pi,\triangle_{\mathsf{eSM}}}^{\mathsf{eSM}}(\mathcal{A}) = (0,0,1)] - \frac{1}{2}| \Big)\end{aligned}$$

Below, we respectively measure $\Pr[\mathrm{Expr}_{\Pi,\triangle_{\mathsf{eSM}}}^{\mathsf{eSM}}(\mathcal{A}) = (1,0,0)]$, $\Pr[\mathrm{Expr}_{\Pi,\triangle_{\mathsf{eSM}}}^{\mathsf{eSM}}(\mathcal{A}) = (0,1,0)]$, and $|\Pr[\mathrm{Expr}_{\Pi,\triangle_{\mathsf{eSM}}}^{\mathsf{eSM}}(\mathcal{A}) = (0,0,1)] - \frac{1}{2}|$ in the following Case 1, 2, and 3.

Case 1. We compute the probability $\Pr[\mathrm{Expr}_{\Pi,\triangle_{\mathsf{eSM}}}^{\mathsf{eSM}}(\mathcal{A}) = (1,0,0)]$, i.e., $\mathcal{A}$ wins via the winning predicate $\mathsf{win}^{\mathsf{corr}}$ by reduction. Namely, if $\mathcal{A}$ can win $\mathrm{Expr}_{\Pi,\triangle_{\mathsf{eSM}}}^{\mathsf{eSM}}$ experiment of the eSM construction Π with a parameter $\triangle_{\mathsf{eSM}}$, then there exists an attacker $\mathcal{B}_1$ that breaks simplified CORR security of the eSM construction Π with the same parameter $\triangle_{\mathsf{eSM}}$. Let $\mathcal{C}_1$ denote the challenger in the $\mathrm{Expr}_{\Pi,\triangle_{\mathsf{eSM}}}^{\mathsf{CORR}}$ experiment. At the beginning, the attacker $\mathcal{B}_1$ samples a challenge bit $\mathsf{b} \in \{0,1\}$ uniformly at random. Then, $\mathcal{B}_1$ invokes $\mathcal{A}$ and answers the queries from $\mathcal{A}$ as follows. Note that all safe predicates in eSM and CORR experiments are identical, $\mathcal{B}_1$ can always compute the safe predicates by itself, according to $\mathcal{A}$'s previous queries.

- $\mathcal{O}_{\mathsf{NewIdKey\text{-}A}}(r)$ and $\mathcal{O}_{\mathsf{NewIdKey\text{-}B}}(r)$: $\mathcal{B}_1$ simply forwards them to $\mathcal{C}_1$ followed by forwarding replies from $\mathcal{C}_1$ to $\mathcal{A}$.
- $\mathcal{O}_{\mathsf{NewPreKey\text{-}A}}(r)$ and $\mathcal{O}_{\mathsf{NewPreKey\text{-}B}}(r)$: $\mathcal{B}_1$ simply forwards them to $\mathcal{C}_1$ followed by forwarding replies from $\mathcal{C}_1$ to $\mathcal{A}$.
- $\mathcal{O}_{\mathsf{RevIdKey\text{-}A}}$ and $\mathcal{O}_{\mathsf{RevIdKey\text{-}B}}$: $\mathcal{B}_1$ sets $\mathsf{safe}_{\mathtt{A}}^{\mathsf{idK}}$ or $\mathsf{safe}_{\mathtt{B}}^{\mathsf{idK}}$ (according the invoked oracle) to false and runs $\mathsf{corruption\text{-}update}()$. For each record in the $\mathsf{allChall}$ set, $\mathcal{B}_1$ then checks whether the safe challenge predicate for all of the records holds. If one of them is false, $\mathcal{B}_1$ undoes the actions in this query and exists the oracle invocation. In particular, $\mathcal{B}_1$ resets the safe identity predicate to true. Then, the attacker $\mathcal{B}_1$ simply forwards the queries to $\mathcal{C}_1$ followed by forwarding replies from $\mathcal{C}_1$ to $\mathcal{A}$.
- $\mathcal{O}_{\mathsf{RevPreKey\text{-}A}}(\mathsf{ind})$ and $\mathcal{O}_{\mathsf{RevPreKey\text{-}B}}(\mathsf{ind})$: $\mathcal{B}_1$ adds the ind into the pre-key reveal list, according to the invoked oracle and runs $\mathsf{corruption\text{-}update}()$. For each record in the $\mathsf{allChall}$ set, $\mathcal{B}_1$ then checks whether the safe challenge predicate for all of the records holds. If one of them is false, $\mathcal{B}_1$ undoes the actions in this query and exists the oracle invocation. In particular, $\mathcal{B}_1$ removes the pre-key counter ind from the pre-key reveal list. Then, the attacker $\mathcal{B}_1$ simply forwards the queries to $\mathcal{C}_1$ followed by forwarding replies from $\mathcal{C}_1$ to $\mathcal{A}$.

- $\mathcal{O}_{\textsf{Corrupt-A}}$ and $\mathcal{O}_{\textsf{Corrupt-B}}$: Let $\texttt{P}$ denote the party, whose session state the attacker is trying to corrupt. $\mathcal{B}_1$ adds the corresponding epoch counter $t_{\texttt{P}}$ into the session state corruption list $\mathcal{L}_{\texttt{P}}^{\textsf{cor}}$ and runs $\textsf{corruption-update}()$. Next, $\mathcal{B}_1$ checks whether there exists a record including $(\neg\texttt{P}, \textsf{ind}, \textsf{flag}) \in \textsf{chall}$. If such element does not exist, or, such element exists but either of the following conditions holds,

  – $\textsf{flag} = \textsf{good}$ and $\textsf{safe}_{\texttt{P}}^{\textsf{idK}}$
  – $\textsf{flag} = \textsf{good}$ and $\textsf{safe}_{\texttt{P}}^{\textsf{preK}}(\textsf{ind})$

  If one of them is $\textsf{false}$, $\mathcal{B}_1$ undoes the actions in this query and exists the oracle invocation. In particular, $\mathcal{B}_1$ removes the epoch counter $t_{\texttt{P}}$ from the session state corruption list. Then, the attacker $\mathcal{B}_1$ simply forwards the queries to $\mathcal{C}_1$ followed by forwarding replies from $\mathcal{C}_1$ to $\mathcal{A}$.

- $\mathcal{O}_{\textsf{Transmit-A}}(\textsf{ind}, m, r)$ and $\mathcal{O}_{\textsf{Transmit-B}}(\textsf{ind}, m, r)$: $\mathcal{B}_1$ simply forwards them to $\mathcal{C}_1$ followed by forwarding replies from $\mathcal{C}_1$ to $\mathcal{A}$.

- $\mathcal{O}_{\textsf{Challenge-A}}(\textsf{ind}, m_0, m_1, r)$ and $\mathcal{O}_{\textsf{Challenge-B}}(\textsf{ind}, m_0, m_1, r)$: We first consider the case for answering $\mathcal{O}_{\textsf{Challenge-A}}(\textsf{ind}, m_0, m_1, r)$. The attacker $\mathcal{B}_1$ first computes $\textsf{flag} = [\![r = \perp]\!]$. Namely, $\textsf{flag} = \textsf{true}$ if and only if r is $\perp$. Then, $\mathcal{B}_1$ checks whether the predicate $\textsf{safe-ch}_{\texttt{A}}(\textsf{flag}, t_{\texttt{A}}, \textsf{ind})$ is true, according to $\mathcal{A}$'s previous queries. If the safe predicates is false, or, the input messages m_0 and m_1 have the distinct length, $\mathcal{B}_1$ simply aborts the oracle. Otherwise, $\mathcal{B}_1$ queries $\mathcal{O}_{\textsf{Transmit-A}}(\textsf{ind}, m_{\textsf{b}}, r)$ to $\mathcal{C}_1$ for a ciphertext c. Then, $\mathcal{B}_1$ adds the record $\textsf{record}(\texttt{A}, \textsf{ind}, \textsf{flag}, t_{\texttt{A}}, i_{\texttt{A}}, m_{\textsf{b}}, c)$ into its own $\textsf{allChall}$ and $\textsf{chall}$. Finally, $\mathcal{B}_1$ returns c to $\mathcal{A}$.

  The step for answering $\mathcal{O}_{\textsf{Challenge-B}}(\textsf{ind}, m_0, m_1, r)$ is similar to above step except that the functions and variables related to $\texttt{A}$ are replaced by the ones to $\texttt{B}$ and vice versa.

- $\mathcal{O}_{\textsf{Deliver-A}}(c)$ and $\mathcal{O}_{\textsf{Deliver-B}}(c)$: $\mathcal{B}_1$ first checks whether there exists an element $(t, i, c) \in \textsf{chall}$ for any t and i. If such element exists, the attacker $\mathcal{B}_1$ simply returns $(t, i, \perp)$ to $\mathcal{A}$. Otherwise, $\mathcal{B}_1$ simply forwards the queries to $\mathcal{C}_1$, followed by forwarding replies from $\mathcal{C}_1$ to $\mathcal{A}$. After that, $\mathcal{B}_1$ removes any element including (t, i, c) from the challenge set $\textsf{chall}$.

- $\mathcal{O}_{\textsf{Inject-A}}(\textsf{ind}, c)$ and $\mathcal{O}_{\textsf{Inject-B}}(\textsf{ind}, c)$: $\mathcal{B}_1$ simply forwards them to $\mathcal{C}_1$ followed by forwarding replies from $\mathcal{C}_1$ to $\mathcal{A}$.

Note that if the attacker $\mathcal{A}$ wins via the winning predicate $\textsf{win}^{\textsf{corr}}$, the winning predicate $\textsf{win}^{\textsf{auth}}$ in the $\mathcal{O}_{\textsf{Inject-A}}$ and $\mathcal{O}_{\textsf{Inject-B}}$ is never set to $\textsf{true}$, which implies either $m' = \perp$ or $(\texttt{B}, t', i') \in \textsf{comp}$, where t' and i' can be efficiently computed from the input ciphertext c. This means, the reduced injection oracles are identical to the original injection oracles from $\mathcal{A}$'s view. Moreover, all other oracles are honestly simulated. This means, $\mathcal{B}_1$ wins if and only if $\mathcal{A}$ wins. Thus, we have that

$$\Pr[\mathrm{Expr}_{\Pi,\triangle_{\textsf{eSM}}}^{\textsf{eSM}}(\mathcal{A}) = (1,0,0)] \leq \textsf{Adv}_{\Pi,\triangle_{\textsf{eSM}}}^{\textsf{CORR}}(\mathcal{B}_1) \leq \epsilon_{\Pi}^{\textsf{CORR}}$$

Furthermore, if $\mathcal{A}$ runs in time t, so does $\mathcal{B}_1$.

Case 2. We compute the probability $\Pr[\mathrm{Expr}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (0,1,0)]$, i.e., $\mathcal{A}$ wins via the winning predicate $\mathsf{win}^{\mathsf{auth}}$ by reduction.

Namely, if $\mathcal{A}$ can win $\mathrm{Expr}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ experiment of a eSM construction Π with a parameter $\triangle_{\mathsf{eSM}}$, then there exists an attacker $\mathcal{B}_2$ that breaks simplified AUTH security of the eSM construction Π with the same parameter $\triangle_{\mathsf{eSM}}$. Let $\mathcal{C}_2$ denote the challenger in the $\mathrm{Expr}^{\mathsf{AUTH}}_{\Pi,\triangle_{\mathsf{eSM}}}$ experiment. At the beginning, the attacker $\mathcal{B}_2$ samples a challenge bit $\mathsf{b} \in \{0,1\}$ and an epoch $t^\star \in [q_{\mathsf{ep}}]$ uniformly at random. Next, $\mathcal{B}_2$ sends $t^\star$ to its challenger $\mathcal{C}_2$. Then, $\mathcal{B}_2$ invokes $\mathcal{A}$ and answers the queries from $\mathcal{A}$ as follows. Note that all safe predicates in eSM and AUTH experiments are identical, $\mathcal{B}_2$ can always compute the safe predicates by itself, according to $\mathcal{A}$'s previous queries.

- $\mathcal{O}_{\mathsf{NewIdKey\text{-}A}}(r)$ and $\mathcal{O}_{\mathsf{NewIdKey\text{-}B}}(r)$: $\mathcal{B}_2$ simply forwards them to $\mathcal{C}_2$ followed by forwarding replies from $\mathcal{C}_2$ to $\mathcal{A}$.
- $\mathcal{O}_{\mathsf{NewPreKey\text{-}A}}(r)$ and $\mathcal{O}_{\mathsf{NewPreKey\text{-}B}}(r)$: $\mathcal{B}_2$ simply forwards them to $\mathcal{C}_2$ followed by forwarding replies from $\mathcal{C}_2$ to $\mathcal{A}$.
- $\mathcal{O}_{\mathsf{RevIdKey\text{-}A}}$ and $\mathcal{O}_{\mathsf{RevIdKey\text{-}B}}$: $\mathcal{B}_2$ sets $\mathsf{safe}^{\mathsf{idK}}_{\mathtt{A}}$ or $\mathsf{safe}^{\mathsf{idK}}_{\mathtt{B}}$ (according the invoked oracle) to false and runs $\mathsf{corruption\text{-}update}()$. For each record in the $\mathsf{allChall}$ set, $\mathcal{B}_2$ then checks whether the safe challenge predicate for all of the records holds. If one of them is false, $\mathcal{B}_2$ undoes the actions in this query and exists the oracle invocation. In particular, $\mathcal{B}_2$ resets the safe identity predicate to true. Then, the attacker $\mathcal{B}_2$ simply forwards the queries to $\mathcal{C}_2$ followed by forwarding replies from $\mathcal{C}_2$ to $\mathcal{A}$.
- $\mathcal{O}_{\mathsf{RevPreKey\text{-}A}}(\mathsf{ind})$ and $\mathcal{O}_{\mathsf{RevPreKey\text{-}B}}(\mathsf{ind})$: $\mathcal{B}_2$ adds ind into the pre-key reveal list, according to the invoked oracle and runs $\mathsf{corruption\text{-}update}()$. For each record in the $\mathsf{allChall}$ set, $\mathcal{B}_2$ then checks whether the safe challenge predicate for all of the records holds. If one of them is false, $\mathcal{B}_2$ undoes the actions in this query and exists the oracle invocation. In particular, $\mathcal{B}_2$ removes the pre-key counter ind from the pre-key reveal list. Then, the attacker $\mathcal{B}_2$ simply forwards the queries to $\mathcal{C}_2$ followed by forwarding replies from $\mathcal{C}_2$ to $\mathcal{A}$.
- $\mathcal{O}_{\mathsf{Corrupt\text{-}A}}$ and $\mathcal{O}_{\mathsf{Corrupt\text{-}B}}$: Let P denote the party, whose session state the attacker is trying to corrupt. $\mathcal{B}_2$ adds the corresponding epoch counter t_{P} into the session state corruption list $\mathcal{L}^{\mathsf{cor}}_{\mathsf{P}}$ and runs $\mathsf{corruption\text{-}update}()$. Next, $\mathcal{B}_2$ checks whether there exists a record including $(\neg\mathsf{P}, \mathsf{ind}, \mathsf{flag}) \in \mathsf{chall}$. If such element does not exist, or, such element exists but either of the following conditions holds,
  - $\mathsf{flag} = \mathsf{good}$ and $\mathsf{safe}^{\mathsf{idK}}_{\mathsf{P}}$
  - $\mathsf{flag} = \mathsf{good}$ and $\mathsf{safe}^{\mathsf{preK}}_{\mathsf{P}}(\mathsf{ind})$

If one of them is false, $\mathcal{B}_2$ undoes the actions in this query and exists the oracle invocation. In particular, $\mathcal{B}_2$ removes the epoch counter $t_{\mathtt{P}}$ from the session state corruption list. Then, the attacker $\mathcal{B}_2$ simply forwards the queries to $\mathcal{C}_2$ followed by forwarding replies from $\mathcal{C}_2$ to $\mathcal{A}$.

- $\mathcal{O}_{\mathsf{Transmit\text{-}A}}(\mathsf{ind}, m, r)$ and $\mathcal{O}_{\mathsf{Transmit\text{-}B}}(\mathsf{ind}, m, r)$: $\mathcal{B}_2$ simply forwards them to $\mathcal{C}_2$ followed by forwarding replies from $\mathcal{C}_2$ to $\mathcal{A}$.
- $\mathcal{O}_{\mathsf{Challenge\text{-}A}}(\mathsf{ind}, m_0, m_1, r)$ and $\mathcal{O}_{\mathsf{Challenge\text{-}B}}(\mathsf{ind}, m_0, m_1, r)$: We first consider the case for answering $\mathcal{O}_{\mathsf{Challenge\text{-}A}}(\mathsf{ind}, m_0, m_1, r)$. The attacker $\mathcal{B}_2$ first computes $\mathsf{flag} = [\![r = \bot]\!]$. Namely, $\mathsf{flag} = \mathsf{true}$ if and only if r is $\bot$. Then, $\mathcal{B}_2$ checks whether the predicate $\mathsf{safe\text{-}ch}_{\mathtt{A}}(\mathsf{flag}, t_{\mathtt{A}}, \mathsf{ind})$ is true, according to $\mathcal{A}$'s previous queries. If the safe predicates is false, or, the input messages m_0 and m_1 have the distinct length, $\mathcal{B}_2$ simply aborts the oracle. Otherwise, $\mathcal{B}_2$ queries $\mathcal{O}_{\mathsf{Transmit\text{-}A}}(\mathsf{ind}, m_{\mathsf{b}}, r)$ to $\mathcal{C}_2$ for a ciphertext c. Then, $\mathcal{B}_2$ adds the record $\mathsf{record}(\mathtt{A}, \mathsf{ind}, \mathsf{flag}, t_{\mathtt{A}}, i_{\mathtt{A}}, m_{\mathsf{b}}, c)$ into its own $\mathsf{allChall}$ and chall. Finally, $\mathcal{B}_2$ returns c to $\mathcal{A}$.

  The step for answering $\mathcal{O}_{\mathsf{Challenge\text{-}B}}(\mathsf{ind}, m_0, m_1, r)$ is similar to above step except that the functions and variables related to $\mathtt{A}$ are replaced by the ones to $\mathtt{B}$ and vice versa.
- $\mathcal{O}_{\mathsf{Deliver\text{-}A}}(c)$ and $\mathcal{O}_{\mathsf{Deliver\text{-}B}}(c)$: $\mathcal{B}_2$ first checks whether there exists an element $(t, i, c) \in \mathsf{chall}$ for any t and i. If such element exists, the attacker $\mathcal{B}_2$ simply returns $(t, i, \bot)$ to $\mathcal{A}$. Otherwise, $\mathcal{B}_2$ simply forwards the queries to $\mathcal{C}_2$, followed by forwarding replies from $\mathcal{C}_2$ to $\mathcal{A}$. After that, $\mathcal{B}_2$ removes any element including (t, i, c) from the challenge set chall.
- $\mathcal{O}_{\mathsf{Inject\text{-}A}}(\mathsf{ind}, c)$ and $\mathcal{O}_{\mathsf{Inject\text{-}B}}(\mathsf{ind}, c)$: $\mathcal{B}_2$ simply forwards them to $\mathcal{C}_2$ followed by forwarding replies from $\mathcal{C}_2$ to $\mathcal{A}$.

Note that if the attacker $\mathcal{A}$ wins via the winning predicate $\mathsf{win}^{\mathsf{auth}}$, the winning predicate $\mathsf{win}^{\mathsf{corr}}$ in the $\mathcal{O}_{\mathsf{Deliver\text{-}A}}(c)$ and $\mathcal{O}_{\mathsf{Deliver\text{-}B}}(c)$ is never set to true. This means, the deliver oracles in CORR experiment is identical to the original deliver oracles from $\mathcal{A}$'s view. Note also that the winning predicate $\mathsf{win}^{\mathsf{auth}}$ is never set to false once it has been set to true.

Assume that attacker $\mathcal{B}_2$ guesses the epoch $t^\star$ correctly, such that $\mathcal{A}$ triggers the flip of $\mathsf{win}^{\mathsf{auth}}$ by querying $\mathcal{O}_{\mathsf{Inject\text{-}A}}(\mathsf{ind}, c)$ or $\mathcal{O}_{\mathsf{Inject\text{-}B}}(\mathsf{ind}, c)$ for a ciphertext c corresponding to epoch $t^\star$, which happens with probability $\frac{1}{q_{\mathsf{ep}}}$. For all previous queries $\mathcal{O}_{\mathsf{Inject\text{-}A}}(\mathsf{ind}, c)$ and $\mathcal{O}_{\mathsf{Inject\text{-}B}}(\mathsf{ind}, c)$, where c does not correspond to the epoch $t^\star$, the flip of $\mathsf{win}^{\mathsf{auth}}$ from false to true will not be triggered. In this case, our reduced injection oracle correctly simulates the behavior of the original injection oracles. For all previous queries $\mathcal{O}_{\mathsf{Inject\text{-}A}}(\mathsf{ind}, c)$ and $\mathcal{O}_{\mathsf{Inject\text{-}B}}(\mathsf{ind}, ct)$, where c corresponds to the epoch $t^\star$, our reduced injection oracle simulates the identical behavior of the original injection oracles.

Note that all other oracles are honestly simulated. The attacker $\mathcal{B}_2$ wins if and only if $\mathcal{A}$ wins and the guess $t^\star$ is correctly. Note also that the event $\mathcal{A}$ wins and the number that $\mathcal{B}_2$ guesses are independent. Thus, we have that

$$\Pr[\mathrm{Expr}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (1,0,0)] \leq q_{\mathsf{ep}}\mathsf{Adv}^{\mathsf{CORR}}_{\Pi,\triangle_{\mathsf{eSM}}} \leq q_{\mathsf{ep}}\epsilon^{\mathsf{AUTH}}_{\Pi}$$

Moreover, if $\mathcal{A}$ runs in time t, so does $\mathcal{B}_2$.

Case 3. We compute the probability $|\Pr[\mathrm{Expr}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (0,0,1)] - \frac{1}{2}|$, i.e., $\mathcal{A}$ wins via the winning predicate $\mathsf{win}^{\mathsf{priv}}$ by hybrid games. Let G_j denote the simulation of **Game** j.

Game 0. This game is identical to the $\mathrm{Expr}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ experiment. Thus, we have that

$$\Pr[\mathsf{G}_0(\mathcal{A}) = (0,0,1)] = \Pr[\mathrm{Expr}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (0,0,1)]$$

Game i $(1 \leq j \leq q_{\mathsf{ep}})$. This game is identical to **Game** $(j-1)$ except the following modifications:

- When the attacker queries $\mathcal{O}_{\mathsf{Challenge\text{-}A}}(\mathsf{ind}, m_0, m_1, r)$ at epoch j, the challenger first checks whether $\mathsf{ind} \leq n_{\mathsf{B}}$ and $|m_0| = |m_1|$ and aborts if the condition does not hold. Then, the challenger samples a random message $\bar{m}$ of the length $|m_0|$ and runs $\mathcal{O}_{\mathsf{Challenge\text{-}A}}(\mathsf{ind}, \bar{m}, \bar{m}, r)$ instead of $\mathcal{O}_{\mathsf{Challenge\text{-}A}}(m_0, m_1, r)$. Finally, the challenger returns the produced ciphertext c to $\mathcal{A}$.

It is easy to observe that in **Game** q_{ep} all challenge ciphertexts are encrypted independent of the challenge bit. Thus, the attacker $\mathcal{A}$ can output the bit b' only by randomly guessing, which indicates that

$$\Pr[\mathsf{G}_{q_{\mathsf{ep}}}(\mathcal{A}) = (0,0,1)] = \frac{1}{2}$$

Let E denote the event that the attacker can distinguish any two adjacent hybrid games. We have that

$$|\Pr[\mathsf{G}_{j-1}(\mathcal{A}) = (0,0,1)] - \Pr[\mathsf{G}_j(\mathcal{A}) = (0,0,1)]| \leq \Pr[E]$$

Moreover, note that the modifications in every hybrid game j is independent of the behavior in hybrid game $(j-1)$. Thus, we have that

$$\begin{aligned}
&|\Pr[\mathsf{G}_0(\mathcal{A}) = (0,0,1)] - \Pr[\mathsf{G}_{q_{\mathsf{ep}}}(\mathcal{A}) = (0,0,1)]| \\
\leq &|\sum_{j=1}^{q_{\mathsf{ep}}} \Pr[\mathsf{G}_{j-1}(\mathcal{A}) = (0,0,1)] - \Pr[\mathsf{G}_j(\mathcal{A}) = (0,0,1)]| \\
\leq &\sum_{j=1}^{q_{\mathsf{ep}}} |\Pr[\mathsf{G}_{j-1}(\mathcal{A}) = (0,0,1)] - \Pr[\mathsf{G}_j(\mathcal{A}) = (0,0,1)]| \\
\leq &q_{\mathsf{ep}} \Pr[E]
\end{aligned}$$

Below, we analyze the probability of the occurrence of the event E by reduction. Namely, if $\mathcal{A}$ can distinguish any two adjacent games **Game** $(j-1)$ and **Game** j, then there exists an attacker $\mathcal{B}_3$ that breaks simplified PRIV security of the eSM construction Π with the same parameter $\triangle_{\mathsf{eSM}}$. Let $\mathcal{C}_3$ denote the challenger in the $\mathrm{Expr}^{\mathsf{PRIV}}_{\Pi,\triangle_{\mathsf{eSM}}}$ experiment. At the beginning, the attacker $\mathcal{B}_3$ sends the epoch j to its challenger $\mathcal{C}_3$ and samples a bit $\bar{\mathsf{b}} \in \{0,1\}$ uniformly at random. Then, $\mathcal{B}_3$ invokes $\mathcal{A}$ and answers the queries from $\mathcal{A}$ as follows. Note that all safe predicates in **Game** $(j-1)$, **Game** j, and PRIV experiments are identical, $\mathcal{B}_3$ can always compute the safe predicates by itself, according to $\mathcal{A}$'s previous queries.

- $\mathcal{O}_{\mathsf{NewIdKey\text{-}A}}(r)$ and $\mathcal{O}_{\mathsf{NewIdKey\text{-}B}}(r)$: $\mathcal{B}_3$ simply forwards them to $\mathcal{C}_3$ followed by forwarding replies from $\mathcal{C}_3$ to $\mathcal{A}$.
- $\mathcal{O}_{\mathsf{NewPreKey\text{-}A}}(r)$ and $\mathcal{O}_{\mathsf{NewPreKey\text{-}B}}(r)$: $\mathcal{B}_3$ simply forwards them to $\mathcal{C}_3$ followed by forwarding replies from $\mathcal{C}_3$ to $\mathcal{A}$.
- $\mathcal{O}_{\mathsf{RevIdKey\text{-}A}}$ and $\mathcal{O}_{\mathsf{RevIdKey\text{-}B}}$: $\mathcal{B}_3$ sets $\mathsf{safe}^{\mathsf{idK}}_{\mathsf{A}}$ or $\mathsf{safe}^{\mathsf{idK}}_{\mathsf{B}}$ (according the invoked oracle) to false and runs $\mathsf{corruption\text{-}update}()$. For each record in the $\mathsf{allChall}$ set, $\mathcal{B}_3$ then checks whether the safe challenge predicate for all of the records holds. If one of them is false, $\mathcal{B}_3$ undoes the actions in this query and exists the oracle invocation. In particular, $\mathcal{B}_3$ resets the safe identity predicate to true. Then, the attacker $\mathcal{B}_3$ simply forwards the queries to $\mathcal{C}_3$ followed by forwarding replies from $\mathcal{C}_3$ to $\mathcal{A}$.
- $\mathcal{O}_{\mathsf{RevPreKey\text{-}A}}(\mathsf{ind})$ and $\mathcal{O}_{\mathsf{RevPreKey\text{-}B}}(\mathsf{ind})$: $\mathcal{B}_3$ adds ind into the pre-key reveal list, according to the invoked oracle and runs $\mathsf{corruption\text{-}update}()$. For each record in the $\mathsf{allChall}$ set, $\mathcal{B}_3$ then checks whether the safe challenge predicate for all of the records holds. If one of them is false, $\mathcal{B}_3$ undoes the actions in this query and exists the oracle invocation. In particular, $\mathcal{B}_3$ removes the pre-key counter ind from the pre-key reveal list. Then, the attacker $\mathcal{B}_3$ simply forwards the queries to $\mathcal{C}_3$ followed by forwarding replies from $\mathcal{C}_3$ to $\mathcal{A}$.
- $\mathcal{O}_{\mathsf{Corrupt\text{-}A}}$ and $\mathcal{O}_{\mathsf{Corrupt\text{-}B}}$: Let P denote the party, whose session state the attacker is trying to corrupt. $\mathcal{B}_3$ adds the corresponding epoch counter t_{P} into the session state corruption list $\mathcal{L}^{\mathsf{cor}}_{\mathsf{P}}$ and runs $\mathsf{corruption\text{-}update}()$. Next, $\mathcal{B}_3$ checks whether there exists a record including $(\neg\mathsf{P}, \mathsf{ind}, \mathsf{flag}) \in \mathsf{chall}$. If such element does not exist, or, such element exists but either of the following conditions holds,

  – $\mathsf{flag} = \mathsf{good}$ and $\mathsf{safe}^{\mathsf{idK}}_{\mathsf{P}}$
  – $\mathsf{flag} = \mathsf{good}$ and $\mathsf{safe}^{\mathsf{preK}}_{\mathsf{P}}(\mathsf{ind})$

  If one of them is false, $\mathcal{B}_3$ undoes the actions in this query and exists the oracle invocation. In particular, $\mathcal{B}_3$ removes the epoch counter t_{P} from the session state corruption list. Then, the attacker $\mathcal{B}_3$ simply forwards the queries to $\mathcal{C}_3$ followed by forwarding replies from $\mathcal{C}_3$ to $\mathcal{A}$.

- $\mathcal{O}_{\textsf{Transmit-A}}(\textsf{ind}, m, r)$ and $\mathcal{O}_{\textsf{Transmit-B}}(\textsf{ind}, m, r)$: $\mathcal{B}_3$ simply forwards them to $\mathcal{C}_3$ followed by forwarding replies from $\mathcal{C}_3$ to $\mathcal{A}$.
- $\mathcal{O}_{\textsf{Challenge-A}}(\textsf{ind}, m_0, m_1, r)$ and $\mathcal{O}_{\textsf{Challenge-B}}(\textsf{ind}, m_0, m_1, r)$: These oracles are answered according to one of the following cases. Here, we only explain the behavior for answering $\mathcal{O}_{\textsf{Challenge-A}}$ for simplicity. The behavior for answering $\mathcal{O}_{\textsf{Challenge-B}}$ can be defined analogously.
  - $[t_{\texttt{A}} < j]$: When the attacker $\mathcal{A}$ queries $\mathcal{O}_{\textsf{Challenge-A}}(\textsf{ind}, m_0, m_1, r)$ at epoch $t_{\texttt{A}} < j$, the $\mathcal{B}_3$ first computes $\textsf{flag} \leftarrow [\![r = \perp]\!]$. Next, $\mathcal{B}_3$ checks whether $\textsf{safe-ch}_{\texttt{A}}(\textsf{flag}, t_{\texttt{A}}, \textsf{ind}) = \textsf{true}$, $\textsf{ind} \leq n_{\texttt{B}}$, and $|m_0| = |m_1|$ and aborts if any condition does not hold. Otherwise, $\mathcal{B}_3$ samples a random message $\bar{m}$ of the length $|m_0|$ and queries $\mathcal{O}_{\textsf{Transmit-A}}(\textsf{ind}, \bar{m}, r)$ for a ciphertext c. Finally, the $\mathcal{B}_3$ adds the record $\textsf{rec} = (\texttt{A}, \textsf{ind}, \textsf{flag}, t_{\texttt{A}}, i_{\texttt{A}}, \bar{m}, c)$ into both $\textsf{allChall}$ and $\textsf{chall}$, followed by returning the ciphertext c to $\mathcal{A}$.
  - $[t_{\texttt{A}} = j]$: When the attacker $\mathcal{A}$ queries $\mathcal{O}_{\textsf{Challenge-A}}(\textsf{ind}, m_0, m_1, r)$ at epoch $t_{\texttt{A}} = j$, the $\mathcal{B}_3$ first computes $\textsf{flag} \leftarrow [\![r = \perp]\!]$. Next, $\mathcal{B}_3$ checks whether $\textsf{safe-ch}_{\texttt{A}}(\textsf{flag}, t_{\texttt{A}}, \textsf{ind}) = \textsf{true}$, $\textsf{ind} \leq n_{\texttt{B}}$, and $|m_0| = |m_1|$ and aborts if any condition does not hold. Otherwise, $\mathcal{B}_3$ samples a random message $\bar{m}$ of the length $|m_0|$ and queries $\mathcal{O}_{\textsf{Challenge-A}}(\textsf{ind}, m_{\bar{\textsf{b}}}, \bar{m}, r)$ for a ciphertext c. Finally, the $\mathcal{B}_3$ adds the record $\textsf{rec} = (\texttt{A}, \textsf{ind}, \textsf{flag}, t_{\texttt{A}}, i_{\texttt{A}}, \_, c)$ into both $\textsf{allChall}$ and $\textsf{chall}$, followed by returning the ciphertext c to $\mathcal{A}$.
  - $[t_{\texttt{A}} > j]$: When the attacker $\mathcal{A}$ queries $\mathcal{O}_{\textsf{Challenge-A}}(\textsf{ind}, m_0, m_1, r)$ at epoch $t_{\texttt{A}} > j$, the $\mathcal{B}_3$ first computes $\textsf{flag} \leftarrow [\![r = \perp]\!]$. Next, $\mathcal{B}_3$ checks whether $\textsf{safe-ch}_{\texttt{A}}(\textsf{flag}, t_{\texttt{A}}, \textsf{ind}) = \textsf{true}$, $\textsf{ind} \leq n_{\texttt{B}}$, and $|m_0| = |m_1|$, and aborts if any condition does not hold. Otherwise, $\mathcal{B}_3$ queries $\mathcal{O}_{\textsf{Transmit-A}}(\textsf{ind}, m_{\bar{\textsf{b}}}, r)$ for a ciphertext c. Finally, the $\mathcal{B}_3$ adds the record $\textsf{rec} = (\texttt{A}, \textsf{ind}, \textsf{flag}, t_{\texttt{A}}, i_{\texttt{A}}, m_{\bar{\textsf{b}}}, c)$ into both $\textsf{allChall}$ and $\textsf{chall}$, followed by returning the ciphertext c to $\mathcal{A}$.
- $\mathcal{O}_{\textsf{Deliver-A}}(c)$ and $\mathcal{O}_{\textsf{Deliver-B}}(c)$: $\mathcal{B}_3$ first checks whether there exists an element $(t, i, c) \in \textsf{chall}$ for any t and i. If such element exists, the attacker $\mathcal{B}_3$ simply returns $(t, i, \perp)$ to $\mathcal{A}$. Otherwise, $\mathcal{B}_3$ simply forwards the queries to $\mathcal{C}_3$, followed by forwarding replies from $\mathcal{C}_3$ to $\mathcal{A}$. After that, $\mathcal{B}_3$ removes any element including (t, i, c) from the challenge set $\textsf{chall}$.
- $\mathcal{O}_{\textsf{Inject-A}}(\textsf{ind}, c)$ and $\mathcal{O}_{\textsf{Inject-B}}(\textsf{ind}, c)$: $\mathcal{B}_3$ simply forwards them to $\mathcal{C}_3$ followed by forwarding replies from $\mathcal{C}_3$ to $\mathcal{A}$.

Note that if the attacker $\mathcal{A}$ wins via the winning predicate $\textsf{win}^{\textsf{priv}}$, the winning predicate $\textsf{win}^{\textsf{corr}}$ in the $\mathcal{O}_{\textsf{Deliver-A}}$ and $\mathcal{O}_{\textsf{Deliver-B}}$ and $\textsf{win}^{\textsf{auth}}$ in the $\mathcal{O}_{\textsf{Inject-A}}$ and $\mathcal{O}_{\textsf{Inject-B}}$ is never set to $\textsf{true}$. This means, the deliver oracles and injection oracles in $\textsf{PRIV}$ experiment is identical to the original ones from $\mathcal{A}$'s view.

Note that all other oracles are honestly simulated. If the challenge bit $\textsf{b}$ in the $\textsf{PRIV}$ experiment is 0, then $\mathcal{B}_3$ perfectly simulates **Game** $(j-1)$ to $\mathcal{A}$. If the challenge bit $\textsf{b}$

in the PRIV experiment is 1, then $\mathcal{B}_3$ perfectly simulates **Game** j to $\mathcal{A}$. This means, the attacker $\mathcal{B}_3$ wins if and only if $\mathcal{A}$ can distinguish the adjacent hybrid games **Game** $(j-1)$ and **Game** j, which is defined as the occurrence of event E. Thus, we have that

$$\Pr[E] \leq \mathsf{Adv}^{\mathsf{PRIV}}_{\Pi,\triangle_{\mathsf{eSM}}} \leq \epsilon^{\mathsf{PRIV}}_{\mathsf{eSM}}$$

Combing the equations above, we have that:

$$\begin{aligned}
&|\Pr[\mathrm{Expr}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (0,0,1)] - \frac{1}{2}| \\
=&|\Pr[\mathsf{G}_0(\mathcal{A}) = (0,0,1)] - \Pr[\mathsf{G}_{q_{\mathsf{ep}}}(\mathcal{A})]| \\
\leq& q_{\mathsf{ep}} \Pr[E] \leq q_{\mathsf{ep}}\epsilon^{\mathsf{PRIV}}_{\mathsf{eSM}}
\end{aligned}$$

Moreover, if $\mathcal{A}$ runs in time t, so does $\mathcal{B}_2$.

Conclusion. The proof is concluded by

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) =& \max\Big(\Pr[\mathrm{Expr}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (1,0,0)], \\
& \Pr[\mathrm{Expr}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (0,1,0)], \\
& |\Pr[\mathrm{Expr}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (0,0,1)] - \frac{1}{2}|\Big) \\
\leq& \max\Big(\epsilon^{\mathsf{CORR}}_{\Pi}, q_{\mathsf{ep}}\epsilon^{\mathsf{AUTH}}_{\Pi}, q_{\mathsf{ep}}\epsilon^{\mathsf{PRIV}}_{\Pi}\Big) \\
\leq& \epsilon^{\mathsf{CORR}}_{\Pi} + q_{\mathsf{ep}}(\epsilon^{\mathsf{AUTH}}_{\Pi} + \epsilon^{\mathsf{PRIV}}_{\Pi})
\end{aligned}$$

□

7.11.4 Proof of Lemma 3

Proof. The proof is given by a sequence of games. Let Adv_j denote the attacker $\mathcal{A}$'s advantage in winning **Game** j.

Game 0. This game is identical to the $\mathrm{Expr}^{\mathsf{CORR}}_{\Pi,\triangle_{\mathsf{eSM}}}$. Thus, we have that

$$\mathsf{Adv}_0 = \epsilon^{\mathsf{CORR}}_{\mathsf{eSM}}$$

Game 1. In this game, if the attacker queries $\mathcal{O}_{\mathsf{Inject\text{-}A}}(\mathsf{ind}, c)$ and $\mathcal{O}_{\mathsf{Inject\text{-}B}}(\mathsf{ind}, c)$ with c corresponding to position $(t^\star, i^\star)$ such that $t^\star \leq \min(t_{\mathtt{A}}, t_{\mathtt{B}}) - 2$, the challenger immediately returns $(t^\star, i^\star, \perp)$.

Note that the oracles are defined symmetric for party $\mathtt{A}$ and $\mathtt{B}$. Without the loss of generality, we only explain the case for $\mathcal{O}_{\mathsf{Inject\text{-}A}}(\mathsf{ind}, c)$ and $t^\star$ is even. The case for $\mathcal{O}_{\mathsf{Inject\text{-}B}}$ and $t^\star$ is odd can be given analogously.

In fact, recall that the eRcv algorithm is executed in $\mathcal{O}_{\mathsf{Inject\text{-}A}}(\mathsf{ind}, c)$ oracle only if the following conditions hold

1. $(\mathsf{B}, c) \notin \mathsf{trans}$
2. $\mathsf{ind} \leq n_{\mathtt{A}}$
3. $\mathsf{safe\text{-}inj}_{\mathtt{A}}(t_{\mathtt{B}}) = \mathsf{true}$ **and** $\mathsf{safe\text{-}inj}_{\mathtt{A}}(t_{\mathtt{A}}) = \mathsf{true}$ which are equivalent to $\mathsf{safe\text{-}st}_{\mathsf{B}}(t_{\mathtt{B}}) = \mathsf{true}$ **and** $\mathsf{safe\text{-}st}_{\mathsf{B}}(t_{\mathtt{A}}) = \mathsf{true}$
4. $(t^\star, i^\star) \in \mathsf{comp}$, where $(t^\star, i^\star)$ is the position of the input ciphertext c

Recall that $(t^\star, i^\star) \in \mathsf{comp}$ means that a ciphertext at this position has been produced by a party, which implies that $t^\star \leq \max(t_{\mathtt{A}}, t_{\mathtt{B}})$. Moreover, a ciphertext is added into comp only when

1. in the $\mathcal{O}_{\mathsf{Corrupt\text{-}A}}$ oracle, if $\mathsf{safe\text{-}st}(t^\star) = \mathsf{false}$ holds.
2. in the $\mathcal{O}_{\mathsf{Corrupt\text{-}B}}$ oracle at epoch $t_{\mathsf{B}} = t^\star$, which means $\mathsf{safe\text{-}st}_{\mathsf{B}}(t^\star) = \mathsf{false}$
3. in the $\mathcal{O}_{\mathsf{Transmit\text{-}B}}$ oracle, if $\mathsf{safe\text{-}inj}_{\mathtt{A}}(t^\star) = \mathsf{safe\text{-}st}_{\mathsf{B}}(t^\star) = \mathsf{false}$ holds
4. in the $\mathcal{O}_{\mathsf{RevIdKey\text{-}A}}$, $\mathcal{O}_{\mathsf{RevIdKey\text{-}B}}$, $\mathcal{O}_{\mathsf{RevPreKey\text{-}A}}$, $\mathcal{O}_{\mathsf{RevPreKey\text{-}B}}$ oracles, if $\mathsf{safe\text{-}inj}_{\mathtt{A}}(t^\star) = \mathsf{safe\text{-}st}_{\mathsf{B}}(t^\star) = \mathsf{false}$

In all of the above cases, we know that $\mathsf{safe\text{-}st}_{\mathsf{B}}(t^\star) = \mathsf{false}$. Note that the conditions $\mathsf{safe\text{-}st}_{\mathsf{B}}(t_{\mathsf{B}}) = \mathsf{false}$ **and** $\mathsf{safe\text{-}st}_{\mathsf{B}}(t_{\mathtt{A}}) = \mathsf{false}$ must hold at the same time. This means, $t^\star \leq \min(t_{\mathtt{A}}, t_{\mathtt{B}}) - 2$. Thus, **Game** 0 and **Game** 1 are identical from the attacker's view. Thus, we have that

$$\mathsf{Adv}_0 = \mathsf{Adv}_1$$

In particular, this also means that both parties have already received at least one message in the epoch $t^\star$ and have produced the root keys before the $\mathcal{O}_{\mathsf{Inject\text{-}A}}$ and $\mathcal{O}_{\mathsf{Inject\text{-}B}}$ for ciphertexts corresponding $t^\star$ are queried.

Game 2. This game is identical to **Game** 1 except the following modification:

1. Whenever the challenger executes $\mathcal{O}_{\mathsf{Transmit\text{-}A}}$ and $\mathcal{O}_{\mathsf{Transmit\text{-}B}}$ to enter a new epoch $t^\star$, the challenger records the root key $rk' \leftarrow \mathsf{st}.rk$ produced during the oracle. When $\mathcal{O}_{\mathsf{Deliver\text{-}A}}$ or $\mathcal{O}_{\mathsf{Deliver\text{-}B}}$ is invoked on the first ciphertext that corresponds to the epoch $t^\star$, the challenger replaces the derivation of the root key rk by the recorded rk'.

The gap between **Game** 1 and **Game** 2 can be analyzed by a sequence of hybrid games, where each hybrid only replace the root key at one epoch. Note that if the receiver executes the eRcv algorithm for the first message in a new epoch. The new $\mathsf{st}.rk$ is derived only when the output of $\mathsf{DS.Vrfy}$ in Line 29 is true, which happens except probability δ_{DS}. Note also that the $\mathcal{O}_{\mathsf{Deliver\text{-}A}}$ and $\mathcal{O}_{\mathsf{Deliver\text{-}B}}$ oracles are used to simulate the transmission of the original data that were produced. The honest KEM ciphertexts are delivered to the receiver and will be decrypted using the corresponding private keys in Line 31. All of them are correctly recovered except probability at most $3\delta_{\mathsf{KEM}}$. If both parties' local root keys are identical, which is true due to the previous hybrid game, the root keys of both parties in this epoch are also identical in this hybrid game. Note that there are at most q_{ep} epochs. Thus, we have that

$$\mathsf{Adv}_1 \leq \mathsf{Adv}_2 + q_{\mathsf{ep}}(\delta_{\mathsf{DS}} + 3\delta_{\mathsf{KEM}})$$

Game 3. This game is identical to **Game** 2 except the following modification:

1. Whenever the challenger executes $\mathcal{O}_{\mathsf{Transmit\text{-}A}}$ and $\mathcal{O}_{\mathsf{Transmit\text{-}B}}$, the challenger records the message key $mk' \leftarrow mk$ produced during the oracle together with the position. When $\mathcal{O}_{\mathsf{Deliver\text{-}A}}$ or $\mathcal{O}_{\mathsf{Deliver\text{-}B}}$ is invoked on a ciphertext, the challenger searches the mk at the location of the input c, followed by replacing the derivation of the message key mk by the recorded mk'.

This game is similar to **Game** 2. The only difference is that the challenger runs q hybrid games but not q_{ep}, where q denotes the maximal queries that $\mathcal{A}$ can make. Thus, we can easily have that

$$\mathsf{Adv}_2 \leq \mathsf{Adv}_3 + q(\delta_{\mathsf{DS}} + 3\delta_{\mathsf{KEM}})$$

Game 4. This game s identical to **Game** 3 except the following modification:

1. Whenever the challenger executes $\mathcal{O}_{\mathsf{Transmit\text{-}A}}(\mathsf{ind}, m, r)$ and $\mathcal{O}_{\mathsf{Transmit\text{-}B}}(\mathsf{ind}, m, r)$, the challenger records the message m produced during the oracle together with the position. When $\mathcal{O}_{\mathsf{Deliver\text{-}A}}$ or $\mathcal{O}_{\mathsf{Deliver\text{-}B}}$ is invoked on a ciphertext, the challenger searches the message m' at the location of the input c, followed by replacing the recovery of the message m by the recorded m'.

This game is similar to **Game** 3. The only difference is that the challenger runs q hybrid games on the scheme SKE which is deterministic and δ_{SKE}-correct. Similarly, we can easily have that

$$\mathsf{Adv}_3 \leq \mathsf{Adv}_4 + q\delta_{\mathsf{SKE}}$$

Final Analysis of Game 4: Now, whenever $\mathcal{O}_{\textsf{Deliver-A}}$ or $\mathcal{O}_{\textsf{Deliver-B}}$ is delivered, the original messages are always correctly recovered and output with the correct position, which means the attacker never wins. Thus, we have that

$$\mathsf{Adv}_5 = 0$$

The following equation concludes the proof.

$$\begin{aligned}\epsilon_{\mathsf{eSM}}^{\mathsf{CORR}} &\leq q_{\mathsf{ep}}(\delta_{\mathsf{DS}} + 3\delta_{\mathsf{KEM}}) + q(\delta_{\mathsf{DS}} + 3\delta_{\mathsf{KEM}} + \delta_{\mathsf{SKE}}) \\ &= (q_{\mathsf{ep}} + q)\delta_{\mathsf{DS}} + 3(q_{\mathsf{ep}} + q)\delta_{\mathsf{KEM}} + q\delta_{\mathsf{SKE}}\end{aligned}$$

□

7.11.5 Proof of Lemma 4

Proof. The proof is given by a sequence of games. Let Adv_j denote the attacker $\mathcal{A}$'s advantage in winning Game j. At the beginning of the experiment, the attacker $\mathcal{A}$ outputs a target epoch $t^\star$, such that it only queries challenge oracles in this epoch. Without loss of generality, we assume $t^\star$ is odd, i.e., $\mathtt{A}$ is the message sender. The case for $t^\star$ is even can be given analogously.

Game 0. This game is identical to the $\mathrm{Expr}^{\mathsf{PRIV}}_{\Pi,\triangle_{\mathsf{eSM}}}$. Thus, we have that

$$\mathsf{Adv}_0 = \epsilon^{\mathsf{PRIV}}_{\mathsf{eSM}}$$

Game 1. This game is identical to **Game** 0 except the following modifications:

1. At the beginning of the game, in addition to the target epoch $t^\star$, the attacker has to output a target message index $i^\star$.
2. The challenge oracle $\mathcal{O}_{\mathsf{Challenge\text{-}A}}$ can only be queried for encrypting $i^\star$-th message (i.e., $i_{\mathtt{A}} = i^\star - 1$ before the query and $i_{\mathtt{A}} = i^\star$ after the query) in $t_{\mathtt{A}} = t^\star$.

We analyze the gap between **Game** 0 and **Game** 1 by hybrid games. Note that $\mathcal{A}$ can query oracles at most q times. There are at most q messages can be encrypted in the target epoch.

Game 1.0. This game is identical to **Game** 0. Thus, we have that

$$\mathsf{Adv}_{1.0} = \mathsf{Adv}_0$$

Game 1.j, $1 \le j \le q$. This game is identical to **Game** 1.$(j-1)$ except the following modification:

1. If $\mathcal{A}$ sends challenge oracle $\mathcal{O}_{\mathsf{Challenge\text{-}A}}(\mathsf{ind}, m_0, m_1, r)$ for encrypting j-th message. The challenger first checks whether m_0 and m_1 have the same length and aborts if the condition does not hold. Then, the challenge samples a random message $\bar{m}$ of the length m_0 and runs $\mathcal{O}_{\mathsf{Challenge\text{-}A}}(\mathsf{ind}, \bar{m}, \bar{m}, r)$ instead of $\mathcal{O}_{\mathsf{Challenge\text{-}A}}(\mathsf{ind}, m_0, m_1, r)$. Finally, the challenger returns the produced ciphertext c to $\mathcal{A}$.

It is easy to observe that all challenge ciphertexts are encrypted independent of the challenge bit in **Game** 1.q. Thus, the attacker can guess the challenge bit only by randomly guessing in **Game** 1.q, which implies that

$$\mathsf{Adv}_{1.q} = 0$$

Let E denote the event that the attacker $\mathcal{A}$ can distinguish any two adjacent hybrid games. Note that the modification in every hybrid game j is independent of the behavior in hybrid game $(j-1)$. Thus, we have that

$$\mathsf{Adv}_{1.0} = \mathsf{Adv}_{1.0} - \mathsf{Adv}_{1.q} \le q \Pr[E]$$

We compute the probability of the occurrence of the event E by reduction. If $\mathcal{A}$ can distinguish any **Game** 1.$(j-1)$ and **Game** 1.j, then we can construct an attacker $\mathcal{B}_1$ that breaks **Game** 1. The attacker $\mathcal{B}_1$ is executed as follows:

1. When $\mathcal{A}$ outputs an epoch $t^\star$, $\mathcal{B}$ outputs $(t^\star, j)$. Meanwhile, $\mathcal{B}_1$ samples a random bit $\bar{\mathsf{b}} \in \{0,1\}$ uniformly at random.
2. When $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Challenge\text{-}A}}$, $\mathcal{B}$ answers according one of the following case:
   - $[i_{\mathtt{A}} < j-1]$: When the attacker queries $\mathcal{O}_{\mathsf{Challenge\text{-}A}}(\mathsf{ind}, m_0, m_1, r)$ when $i_{\mathtt{A}} < j-1$, i.e., for encrypting messages before j-th message. $\mathcal{B}_1$ first computes $\mathsf{flag} \leftarrow [\![r = \perp]\!]$. Next $\mathcal{B}_1$ checks whether $\mathsf{safe\text{-}ch}_{\mathtt{A}}(\mathsf{flag}, t_{\mathtt{A}}, \mathsf{ind})$, $\mathsf{ind} \le n_{\mathtt{B}}$, and m_0 and m_1 have the same length. If any condition does not hold, $\mathcal{B}_1$ simply aborts. Otherwise, $\mathcal{B}_1$ samples a random message $\bar{m}$ of the length m_0 and queries $\mathcal{O}_{\mathsf{Transmit\text{-}A}}(\mathsf{ind}, \bar{m}, r)$ for a ciphertext c. Finally, $\mathcal{B}_1$ adds the corresponding record into both $\mathsf{allChall}$ and chall, followed by returning the ciphertext c to $\mathcal{A}$.
   - $[i_{\mathtt{A}} = j-1]$: When the attacker queries $\mathcal{O}_{\mathsf{Challenge\text{-}A}}(\mathsf{ind}, m_0, m_1, r)$ when $i_{\mathtt{A}} = j-1$, i.e., for encrypting j-th message. $\mathcal{B}_1$ first computes $\mathsf{flag} \leftarrow [\![r = \perp]\!]$. Next $\mathcal{B}_1$ checks whether $\mathsf{safe\text{-}ch}_{\mathtt{A}}(\mathsf{flag}, t_{\mathtt{A}}, \mathsf{ind})$, $\mathsf{ind} \le n_{\mathtt{B}}$, and m_0 and m_1 have the same length. If any condition does not hold, $\mathcal{B}_1$ simply aborts. Otherwise, $\mathcal{B}_1$ samples a random message $\bar{m}$ of the length m_0 and queries $\mathcal{O}_{\mathsf{Challenge\text{-}A}}(\mathsf{ind}, m_{\bar{\mathsf{b}}}, \bar{m}, r)$ for a ciphertext c. Finally, $\mathcal{B}_1$ adds the corresponding record into both $\mathsf{allChall}$ and chall, followed by returning the ciphertext c to $\mathcal{A}$.
   - $[i_{\mathtt{A}} > j-1]$: When the attacker queries $\mathcal{O}_{\mathsf{Challenge\text{-}A}}(\mathsf{ind}, m_0, m_1, r)$ when $i_{\mathtt{A}} > j-1$, i.e., for encrypting messages after j-th message. $\mathcal{B}_1$ first computes $\mathsf{flag} \leftarrow [\![r = \perp]\!]$. Next $\mathcal{B}_1$ checks whether $\mathsf{safe\text{-}ch}_{\mathtt{A}}(\mathsf{flag}, t_{\mathtt{A}}, \mathsf{ind})$, $\mathsf{ind} \le n_{\mathtt{B}}$, and m_0 and m_1 have the same length. If either condition does not hold, $\mathcal{B}_1$ simply aborts. Otherwise, $\mathcal{B}_1$ queries $\mathcal{O}_{\mathsf{Transmit\text{-}A}}(\mathsf{ind}, m_{\bar{\mathsf{b}}}, r)$ for a ciphertext c. Finally, $\mathcal{B}_1$ adds the corresponding record into both $\mathsf{allChall}$ and chall, followed by returning the ciphertext c to $\mathcal{A}$.
3. To answer all other oracles, $\mathcal{B}_1$ first checks whether the safe predicate requirements in individual oracles hold. If so, $\mathcal{B}_1$ simply forward the queries to challenger and returns the reply to $\mathcal{A}$. If not, $\mathcal{B}_1$ simply aborts.

Note that all other oracles are honestly simulated except for $\mathcal{O}_{\mathsf{Challenge\text{-}A}}$. If the challenge bit b in **Game** 1 is 0, then $\mathcal{B}_1$ perfectly simulates **Game** 1.$(j-1)$ to $\mathcal{A}$. If the challenge bit b in **Game** 1 is 1, then $\mathcal{B}_1$ perfectly simulates **Game** 1.j to $\mathcal{A}$. Thus, if $\mathcal{A}$ can distinguish any adjacent two hybrid games, $\mathcal{B}_1$ wins **Game** 1, which implies $\Pr[E] \le \mathsf{Adv}_1$, and further

$$\mathsf{Adv}_0 = \mathsf{Adv}_{1.0} \le q \Pr[E] \le q\mathsf{Adv}_1$$

Game 2. Let $\mathsf{ind}^\star$ denote the index of $\mathit{prepk}_{\mathtt{B}}$ that is used to encrypt $i^\star$'s message in epoch $t^\star$. Let $\mathsf{flag}^\star$ denote the random quality in the target challenge oracle. In this game, $\mathcal{A}$ wins immediately, if at the end of experiment $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \Big(\mathsf{flag}^\star = \mathsf{good}$ **and** $\mathsf{safe}_{\mathtt{B}}^{\mathsf{idK}}\Big) = \Big(\mathsf{flag}^\star = \mathsf{good}$ **and** $\mathsf{safe}_{\mathtt{B}}^{\mathsf{preK}}(\mathsf{ind}^\star)\Big) = \mathsf{false}$.

Note that before the challenge query, the safe predicate $\mathsf{safe\text{-}ch}_{\mathtt{A}}(\mathsf{flag}, t^\star, \mathsf{ind}^\star)$ must hold, i.e.,

$$\Big(\mathsf{safe\text{-}st}_{\mathtt{A}}(t^\star) \textbf{ and } \mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star)\Big) \textbf{ or } \Big(\mathsf{flag}^\star = \mathsf{good} \textbf{ and } \mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star)\Big) \textbf{ or } \Big(\mathsf{flag}^\star = \mathsf{good} \textbf{ and } \mathsf{safe}_{\mathtt{B}}^{\mathsf{idK}}\Big) \textbf{ or } \Big(\mathsf{flag}^\star = \mathsf{good} \textbf{ and } \mathsf{safe}_{\mathtt{B}}^{\mathsf{preK}}(\mathsf{ind}^\star)\Big)$$

This means, at least one of the following conditions must hold at the time of query of $\mathcal{O}_{\mathsf{Challenge\text{-}A}}$.

1. $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \mathsf{true}$
2. $\Big(\mathsf{flag}^\star = \mathsf{good}$ **and** $\mathsf{safe}_{\mathtt{B}}^{\mathsf{idK}}\Big) = \mathsf{true}$
3. $\Big(\mathsf{flag}^\star = \mathsf{good}$ **and** $\mathsf{safe}_{\mathtt{B}}^{\mathsf{preK}}(\mathsf{ind}^\star)\Big) = \mathsf{true}$

When querying identity keys or pre-keys oracles, the oracle aborts if it will triggers the safe challenge predicate $\mathsf{safe\text{-}ch}_{\mathtt{A}}(\mathsf{flag}^\star, t^\star, \mathsf{ind}^\star)$ to false. When querying corruption oracles, the violation of $\mathsf{safe\text{-}st}_{\mathtt{B}}$ must indicate $\Big(\mathsf{flag}^\star = \mathsf{good}$ **and** $\mathsf{safe}_{\mathtt{B}}^{\mathsf{idK}}\Big)$ or $\Big(\mathsf{flag}^\star = \mathsf{good}$ **and** $\mathsf{safe}_{\mathtt{B}}^{\mathsf{preK}}(\mathsf{ind}^\star)\Big)$. Thus, at least one of the above conditions must hold even at the end of experiment

This means, $\mathcal{A}$ cannot gain any additional advantage in winning **Game** 2, which implies that

$$\mathsf{Adv}_1 = \mathsf{Adv}_2$$

Below, we analyze the advantage Adv_2 into three cases, whether $\Big(\mathsf{flag}^\star = \mathsf{good}$ **and** $\mathsf{safe}_{\mathtt{B}}^{\mathsf{idK}}\Big) = \mathsf{true}$ or $\Big(\mathsf{flag}^\star = \mathsf{good}$ **and** $\mathsf{safe}_{\mathtt{B}}^{\mathsf{preK}}(\mathsf{ind}^\star)\Big) = \mathsf{true}$ or $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \mathsf{true}$ holds at the end of the experiment.

Case 1: $\Big(\mathsf{flag}^\star = \mathsf{good}$ **and** $\mathsf{safe}_{\mathtt{B}}^{\mathsf{idK}}\Big) = \mathsf{true}$

In this case, $\Big(\mathsf{flag}^\star = \mathsf{good}$ **and** $\mathsf{safe}_{\mathtt{B}}^{\mathsf{idK}}\Big) = \mathsf{true}$ holds at the end of the experiment, thus also holds at the time of challenge oracle $\mathcal{O}_{\mathsf{Challenge\text{-}A}}$ query. We use Adv_j^{C1} to denote $\mathcal{A}$'s advantage in winning **Game** j in this case. In the remaining of this case analysis, we focus on the epoch $t^\star$ and the message index $i^\star$.

Game C1.3. This game is identical to **Game** 2 except the following modification:

1. The challenger additionally samples a random key $k' \in \mathcal{K}$, where $\mathcal{K}$ denote the key space of the underlying KEM.

2. $(\mathsf{upd}^{\mathsf{ar}}, \mathsf{upd}^{\mathsf{ur}}) \leftarrow \mathsf{KDF}_1(k_1, k_2, k_3)$ in Line 16 in Figure 7.6 is replaced by $(\mathsf{upd}^{\mathsf{ar}}, \mathsf{upd}^{\mathsf{ur}}) \leftarrow \mathsf{KDF}_1(k_1, k', k_3)$
3. $k_2 \leftarrow \mathsf{KEM.Decaps}(ik, c_2)$ in Line 31 in Figure 7.6 is replaced by $k_2 \leftarrow k'$

If $\mathcal{A}$ can distinguish **Game** 2 and **Game** C1.3, then we can construct an attacker $\mathcal{B}_2$ that breaks IND-CCA security of underlying KEM. The attacker $\mathcal{B}_2$ receives a public key pk, a challenge ciphertext $c^\star$, and a key $k^\star$, and simulates the game as follows:

1. $\mathcal{A}$ outputs $(t^\star, i^\star)$ at the beginning of the game.
2. When $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{NewIdKey\text{-}B}}(r)$, checks whether $r = \perp$. If $r \neq \perp$, then $\mathcal{B}_2$ returns pk to $\mathcal{A}$.
3. When $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Challenge\text{-}A}}(\mathsf{ind}^\star, m_0, m_1, r)$ for encrypting $i^\star$'s message in the epoch $t^\star$, $\mathcal{B}_2$ aborts if $r \neq \perp$. Then, $\mathcal{B}_2$ honestly runs $\mathcal{O}_{\mathsf{Challenge\text{-}A}}$ except replacing $(\mathsf{upd}^{\mathsf{ar}}, \mathsf{upd}^{\mathsf{ur}}) \leftarrow \mathsf{KDF}_1(k_1, k_2, k_3)$ in Line 16 in Figure 7.6 by $(\mathsf{upd}^{\mathsf{ar}}, \mathsf{upd}^{\mathsf{ur}}) \leftarrow \mathsf{KDF}_1(k_1, k^\star, k_3)$
4. When $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Deliver\text{-}B}}(c)$ oracle, where c is output by $\mathcal{O}_{\mathsf{Challenge\text{-}A}}$ oracles, $\mathcal{B}_2$ honestly runs the eRcv algorithm except directly using $k^\star$ at the place of k_2 instead of running decapsulation algorithm.
5. When $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Inject\text{-}B}}(\mathsf{ind}, c)$ oracle for a pre-key index ind and a ciphertext c, $\mathcal{B}_2$ forwards c to its decapsulation oracle for a key k, followed by use this key in the place of the decapsulated k_2 to run eRcv algorithm.
6. All other oracles are honestly simulated.

Note that if the challenge bit in the IND-CCA security experiment equals 0, then $\mathcal{B}_2$ simulates **Game** 2 to $\mathcal{A}$. If the challenge bit in the IND-CCA security experiment equals 1, then $\mathcal{B}_2$ simulates **Game** C1.3 to $\mathcal{A}$. $\mathcal{B}_2$ wins if and only if $\mathcal{A}$ can distinguish **Game** 2 and **Game** C1.3. Thus, we have that

$$\mathsf{Adv}_2^{C1} \leq \mathsf{Adv}_3^{C1} + \epsilon_{\mathsf{KEM}}^{\mathsf{ind\text{-}cca}}$$

Game C1.4. This game is identical to **Game** C1.3 except the following modifications:

1. The challenger additionally samples a random update value $\widetilde{\mathsf{upd}}^{\mathsf{ur}} \in \{0,1\}^\lambda$
2. $mk \leftarrow \mathsf{KDF}_5(urk, \mathsf{upd}^{\mathsf{ur}})$ in Line 22 and 40 in Figure 7.6 is replaced by $mk \leftarrow \mathsf{KDF}_5(urk, \widetilde{\mathsf{upd}}^{\mathsf{ur}})$

If $\mathcal{A}$ can distinguish **Game** C1.3 and **Game** C1.4, then we can construct an attacker $\mathcal{B}_3$ that breaks 3prf security of underlying KDF_1. Note that the random key k' is sampled random in **Game** C1.3. $\mathcal{B}_3$ can easily query k_1, k_3 to its oracle on the second input, and use the reply in the place of $(\mathsf{upd}^{\mathsf{ar}}, \mathsf{upd}^{\mathsf{ur}})$. If the oracle simulates KDF_1, then $\mathcal{B}_3$ simulates **Game** C1.3 to $\mathcal{A}$. If the oracle simulates a random function, then $\mathcal{B}_3$ simulates **Game** C1.4. Thus, we have that

$$\mathsf{Adv}_3^{C1} \leq \mathsf{Adv}_4^{C1} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}}$$

Game C1.5. This game is identical to **Game** C1.4 except the following modifications:

1. The challenger additionally samples a random message key $\widetilde{mk} \in \{0,1\}^\lambda$
2. $c' \leftarrow \mathsf{SKE.Enc}(mk, m)$ in Line 22 and 40 in Figure 7.6 is replaced by $c' \leftarrow \mathsf{SKE.Enc}(\widetilde{mk}, m)$

Similar to the game above, if $\mathcal{A}$ can distinguish **Game** C1.4 and **Game** C1.5, then we can construct an attacker $\mathcal{B}_4$ that breaks swap security of underlying KDF_5. Note that the random update value $\widetilde{\mathsf{upd}}^{\mathsf{ur}}$ is sampled random in **Game** C1.4. $\mathcal{B}_4$ can easily query urk to its oracle and use the reply in the place of mk. If the oracle simulates KDF_5, then $\mathcal{B}_4$ simulates **Game** C1.3 to $\mathcal{A}$. If the oracle simulates a random function, then $\mathcal{B}_3$ simulates **Game** C1.5. Thus, we have that

$$\mathsf{Adv}_4^{C1} \leq \mathsf{Adv}_5^{C1} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{swap}} \leq \mathsf{Adv}_5^{C1} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{dual}}$$

Game Final Analysis for Case 1: In the end, we compute $\mathcal{A}$'s advantage in winning **Game** C1.5 by reduction. If $\mathcal{A}$ can win **Game** C1.5, then we can construct an attacker $\mathcal{B}_5$ that breaks $\mathsf{IND\text{-}1CCA}$ security of the underlying SKE. The reduction is simulated as follows:

1. $\mathcal{A}$ outputs $(t^\star, i^\star)$ at the beginning of the game.
2. $\mathcal{B}$ samples a random bit $\bar{\mathsf{b}} \xleftarrow{\$} \{0,1\}$.
3. When $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Challenge\text{-}A}}(\mathsf{ind}^\star, m_0, m_1, r)$ for encrypting $i^\star$'s message in the epoch $t^\star$, $\mathcal{B}_5$ aborts if $r \neq \perp$ or m_0 and m_1 have different length. Next, $\mathcal{B}_5$ samples a random message $\bar{m}$ of length $|m_0|$.Then, $\mathcal{B}_5$ queries its challenger on $(\bar{m}, m_{\bar{\mathsf{b}}})$ and receives a ciphertext $c^\star$. After that, $\mathcal{B}_5$ honestly runs $\mathcal{O}_{\mathsf{Challenge\text{-}A}}$ except replacing $c' \leftarrow \mathsf{SKE.Enc}(mk, m)$ in Line 22 and 40 in Figure 7.6 by $c' \leftarrow c^\star$.
4. When $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Deliver\text{-}B}}(c)$ oracle such that c includes $t^\star$, $i^\star$, and $c^\star$, $\mathcal{B}_5$ honestly simulates $\mathcal{O}_{\mathsf{Deliver\text{-}B}}$ except for outputting $m' = \perp$.
5. When $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Inject\text{-}B}}(\mathsf{ind}, c)$ oracle for a pre-key index ind and a ciphertext corresponds to the position $(t^\star, i^\star)$, $\mathcal{B}_5$ forwards c to its decapsulation oracle for a message m', followed by outputting $(t^\star, i^\star, m')$
6. All other oracles are honestly simulated.

Note that if the forgery via $\mathcal{O}_{\mathsf{Inject\text{-}B}}$ is accepted, then the attacker cannot win via $\mathsf{win}^{\mathsf{priv}}$ predicate since a natural eSM scheme does not accept two messages at the same position. So, $\mathcal{B}_5$ perfectly simulate **Game** C1.5 to $\mathcal{A}$ and wins if and only if $\mathcal{A}$ wins. Thus, we have that

$$\mathsf{Adv}_5^{C1} \leq \epsilon_{\mathsf{SKE}}^{\mathsf{ind\text{-}1cca}}$$

To sum up, we have that

$$\mathsf{Adv}_2^{C1} \leq \epsilon_{\mathsf{SKE}}^{\mathsf{ind\text{-}1cca}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{dual}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KEM}}^{\mathsf{ind\text{-}cca}}$$

Case 2: $\Big(\mathsf{flag}^\star = \mathsf{good}$ **and** $\mathsf{safe}_{\mathtt{B}}^{\mathsf{preK}}(\mathsf{ind}^\star)\Big) = \mathsf{true}$

In this case, $\Big(\mathsf{flag}^\star = \mathsf{good}$ **and** $\mathsf{safe}_{\mathtt{B}}^{\mathsf{preK}}(\mathsf{ind}^\star)\Big) = \mathsf{true}$ holds at the end of the experiment, thus also holds at the time of challenge oracle $\mathcal{O}_{\mathsf{Challenge\text{-}A}}$ query. We use Adv_j^{C2} to denote $\mathcal{A}$'s advantage in winning **Game** j in this case. In the remaining of this case analysis, we focus on the epoch $t^\star$ and the message index $i^\star$.

Game C2.3 In this game, the challenger guesses the index of the pre-key $\mathsf{ind}^\star$ by randomly guessing at the beginning of the experiment. If the guess is wrong, the challenger aborts and let $\mathcal{A}$ immediately win. Note that there are at most q_{M} in the experiment, the challenger can guess correctly with probability $\frac{1}{q_{\mathsf{M}}}$. Thus, we have that

$$\mathsf{Adv}_2^{C2} \le q_{\mathsf{M}} \mathsf{Adv}_3^{C2}$$

Game C2.4, C2.5, C2.6. These games are defined similar to **Game** C1.3, C1.4, C1.5. The only difference is to apply the modification not to $\mathtt{B}$'s identity key but $\mathtt{B}$'s $\mathsf{ind}^\star$-th pre-key. The proof can be easily given in a similar way and we have that

$$\mathsf{Adv}_3^{C2} \le \epsilon_{\mathsf{SKE}}^{\mathsf{ind\text{-}1cca}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{dual}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KEM}}^{\mathsf{ind\text{-}cca}}$$

To sum up, we have that

$$\mathsf{Adv}_2^{C2} \le q_{\mathsf{M}}(\epsilon_{\mathsf{SKE}}^{\mathsf{ind\text{-}1cca}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{dual}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KEM}}^{\mathsf{ind\text{-}cca}})$$

Case 3: $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \mathsf{true}$

In this case, $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \mathsf{true}$ holds at the end of the experiment, thus also holds at the time of challenge oracle $\mathcal{O}_{\mathsf{Challenge\text{-}A}}$ query. We further split this case into two subcases: when $\mathcal{A}$ queries the challenge oracle at $\mathcal{O}_{\mathsf{Challenge\text{-}A}}$ for encrypting $i^\star$'s message at epoch $t^\star$ whether $\Big(\mathsf{flag}^\star = \mathsf{good}$ **and** $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star)\Big)$ holds, see Case 3.1, or, $\Big(\mathsf{safe\text{-}st}_{\mathtt{A}}(t^\star)$ **and** $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star)\Big)$ holds, see Case 3.2.

Case 3.1: $\Big(\mathsf{flag}^\star = \mathsf{good}$ **and** $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star)\Big)$

Game C3.1.3 This game is identical to **Game** 2 except the following modification:

1. Whenever $\mathtt{P} \in \{\mathtt{A}, \mathtt{B}\}$ is trying to sending the first message in a new epoch $t+1$ (i.e. $\mathtt{P} = \mathtt{A}$ if t even and $\mathtt{P} = \mathtt{B}$ if t odd) and the execution $\mathcal{L}_{\mathtt{P}}^{\mathsf{cor}} \xleftarrow{+} t+1$ in Line 80 in the $\mathsf{ep\text{-}mgmt}$ helper function in Figure 7.6 is not triggered, then the challenger replaces $r \xleftarrow{\$} \{0,1\}^\lambda$, $(\mathsf{st}_{\mathtt{P}}.nxs, r^{\mathsf{KEM}}, r^{\mathsf{DS}}) \leftarrow \mathsf{KDF}_2(\mathsf{st}_{\mathtt{P}}.nxs, r)$ executed in the following eSend algorithm in Line 18 in Figure 7.6 by $\mathsf{st}_{\mathtt{P}}.nxs \xleftarrow{\$} \{0,1\}^\lambda$, $r^{\mathsf{KEM}} \xleftarrow{\$} \{0,1\}^\lambda$, $r^{\mathsf{DS}} \xleftarrow{\$} \{0,1\}^\lambda$.

We analyze $\mathcal{A}$'s advantage in winning **Game** C3.1.3 by hybrid games.

Game $\mathsf{hy}.0$: This game is identical to **Game** 2. Thus, we have that

$$\mathsf{Adv}_2^{C3.1} = \mathsf{Adv}_{\mathsf{hy}.0}$$

Game $\mathsf{hy}.j, (1 \leq j \leq q_{\mathsf{ep}})$: This game is identical to game **Game** $\mathsf{hy}.(j-1)$ except that:

1. When entering epoch j from $j-1$, if the execution $\mathcal{L}_{\mathsf{P}}^{\mathsf{cor}} \xleftarrow{+} j$ in Line 80 in the $\mathsf{ep\text{-}mgmt}$ helper function in Figure 7.6 is not triggered for $\mathtt{P} = \mathtt{A}$ if j odd and $\mathtt{P} = \mathtt{B}$ if j even, then in the following eSend algorithm, the challenger replaces $r \xleftarrow{\$} \{0,1\}^\lambda$, $(\mathsf{st}_{\mathsf{P}}.nxs, r^{\mathsf{KEM}}, r^{\mathsf{DS}}) \leftarrow \mathsf{KDF}_2(\mathsf{st}_{\mathsf{P}}.nxs, r)$ executed in Line 18 in Figure 7.6 by $\mathsf{st}_{\mathsf{P}}.nxs \xleftarrow{\$} \{0,1\}^\lambda$, $r^{\mathsf{KEM}} \xleftarrow{\$} \{0,1\}^\lambda$, $r^{\mathsf{DS}} \xleftarrow{\$} \{0,1\}^\lambda$.

It is obvious that **Game** $\mathsf{hy}.q_{\mathsf{ep}}$ is identical to **Game** C3.1.3. Thus, we have that

$$\mathsf{Adv}_3^{C3.1} = \mathsf{Adv}_{\mathsf{hy}.q_{\mathsf{ep}}}$$

Let E denote the event that $\mathcal{A}$ can distinguish any adjacent hybrid games **Game** $\mathsf{hy}.(j-1)$ and **Game** $\mathsf{hy}.j$. Note that the modification in every hybrid game is independent of the behavior of the previous game. Thus, we have that

$$\mathsf{Adv}_2^{C3.1} - \mathsf{Adv}_3^{C3.1} \leq q_{\mathsf{ep}} \Pr[E]$$

Below, we compute the probability of the occurrence of event E by case distinction. Note that the execution $\mathcal{L}_{\mathsf{P}}^{\mathsf{cor}} \xleftarrow{+} j$ in **Game** $\mathsf{hy}.j$ indicates that **Game** $\mathsf{hy}.(j-1)$ is identical to **Game** $\mathsf{hy}.j$. Below, we only consider the case for that the execution $\mathcal{L}_{\mathsf{P}}^{\mathsf{cor}} \xleftarrow{+} j$ is not triggered. Note also that $\mathcal{L}_{\mathsf{P}}^{\mathsf{cor}} \xleftarrow{+} j$ is not triggered only when $\mathsf{safe\text{-}ch}_{\mathsf{P}}(\mathsf{flag}, j-1, \mathsf{ind}^\star)$, which further implies that one of the following conditions must hold: (1) $\mathsf{safe\text{-}st}_{\mathsf{P}}(j-1)$ or (2) $\mathsf{flag} = \mathsf{good}$. Then, we consider each of the two cases.

Case $\mathsf{safe\text{-}st}_{\mathsf{P}}(j-1)$**:** First, $\mathsf{safe\text{-}st}_{\mathsf{P}}(j-1)$ means $(j-1), (j-2) \notin \mathcal{L}_{\mathsf{P}}^{\mathsf{cor}}$. Moreover, $(j-1) \notin \mathcal{L}_{\mathsf{P}}^{\mathsf{cor}}$ indicates that (1) the execution $\mathcal{L}_{\mathsf{P}}^{\mathsf{cor}} \xleftarrow{+} (j-2)$ in **Game** $\mathsf{hy}.(j-2)$ is not triggered, and (2) the state corruption on $\mathtt{P}$ is not invoked during epoch $(j-1)$ and $(j-2)$. According to hybrid game **Game** $\mathsf{hy}.(j-2)$, the value $\mathsf{st}_{\mathsf{P}}.nxs$ sampled uniformly at random during sending the first message in epoch $(j-2)$. In other words, $\mathsf{st}_{\mathsf{P}}.nxs$ is uniformly at random from the attacker's view when entering epoch j from $(j-1)$. During sending the first message in epoch j, $r \xleftarrow{\$} \{0,1\}^\lambda$, $(\mathsf{st}_{\mathsf{P}}.nxs, r^{\mathsf{KEM}}, r^{\mathsf{DS}}) \leftarrow \mathsf{KDF}_2(\mathsf{st}_{\mathsf{P}}.nxs, r)$ is executed in Line 18 in Figure 7.6. By the prf security of KDF_2, it is easy to know that if $\mathcal{A}$ can distinguish **Game** $\mathsf{hy}.(j-1)$ and **Game** $\mathsf{hy}.j$, then there must exist an attacker that distinguish the keyed KDF_2 and a random function. Thus, it holds that

$$\Pr[E] \leq \epsilon_{\mathsf{KDF}_2}^{\mathsf{prf}}$$

Case $\mathsf{flag} = \mathsf{good}$**:** This means, the first message in epoch $j-2$ is computed using fresh randomness. In particular, this means, $r \xleftarrow{\$} \{0,1\}^\lambda$, $(\mathsf{st}_\mathtt{P}.nxs, r^{\mathsf{KEM}}, r^{\mathsf{DS}}) \leftarrow \mathsf{KDF}_2(\mathsf{st}_\mathtt{P}.nxs, r)$ is executed in Line 18 in Figure 7.6 uses fresh randomness r. It is easy to know that $\mathsf{st}_\mathtt{P}.nxs$ after sending the first message in epoch $(j-2)$ is distinguishable from a random string, due to the swap-security of KDF_2.

Thus, we have that

$$\Pr[E] \le \epsilon^{\mathsf{swap}}_{\mathsf{KDF}_2}$$

From above two cases, we know that

$$\Pr[E] \le \max\left(\epsilon^{\mathsf{prf}}_{\mathsf{KDF}_2} + \epsilon^{\mathsf{swap}}_{\mathsf{KDF}_2}\right) \le \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_2}$$

To sum up, we have that

$$\mathsf{Adv}^{C3.1}_2 \le q_{\mathsf{ep}} \Pr[E] + \mathsf{Adv}^{C3.1}_3 \le \mathsf{Adv}^{C3.1}_3 + q_{\mathsf{ep}}\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_2}$$

Game C3.1.5, C3.1.6, C3.1.7. Note that $\mathsf{safe\text{-}st}_\mathtt{B}(t^\star)$ means that $t^\star, (t^\star - 1) \notin \mathcal{L}^{\mathsf{cor}}_\mathtt{B}$. This implies that both following conditions must hold:

1. $\mathsf{st}_\mathtt{P}.nxs \xleftarrow{\$} \{0,1\}^\lambda$, $r^{\mathsf{KEM}} \xleftarrow{\$} \{0,1\}^\lambda$, $r^{\mathsf{DS}} \xleftarrow{\$} \{0,1\}^\lambda$ are executed when $\mathtt{B}$ was entering $t^\star - 1$.
2. The corruption oracle $\mathcal{O}_{\mathsf{Corrupt\text{-}B}}$ is not queried during $t^\star$ and $(t^\star - 1)$.

Furthermore, the KEM key pair in $\mathsf{st}_\mathtt{B}$ generated in epoch $t^\star - 1$ for $\mathtt{A}$ to encrypt messages in $t^\star$ is not leaked. Applying a similar game hopping to the KEM key pair in the state, as to the identity key pairs in **Game** 1.3, 1.4, 1.5, we can easily have that

$$\mathsf{Adv}^{C3.1}_3 \le \epsilon^{\mathsf{ind\text{-}1cca}}_{\mathsf{SKE}} + \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}}$$

Combing the above statements, we have that

$$\mathsf{Adv}^{C3.1}_2 \le \epsilon^{\mathsf{ind\text{-}1cca}}_{\mathsf{SKE}} + \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}} + q_{\mathsf{ep}}\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_2}$$

Case 3.2: $\left(\mathsf{safe\text{-}st}_\mathtt{A}(t^\star) \textbf{ and } \mathsf{safe\text{-}st}_\mathtt{B}(t^\star)\right)$

Game C3.2.3 This game is identical to **Game** 2 except the following modification:

1. Whenever $\mathtt{P} \in \{\mathtt{A}, \mathtt{B}\}$ is trying to sending the first message in a new epoch $t+1$ (i.e. $\mathtt{P} = \mathtt{A}$ if t even and $\mathtt{P} = \mathtt{B}$ if t odd) and the execution $\mathcal{L}^{\mathsf{cor}}_\mathtt{P} \xleftarrow{+} t+1$ in Line 80 in the $\mathsf{ep\text{-}mgmt}$ helper function in Figure 7.6 is not triggered, then the challenger replaces $(\mathsf{st}.rk, \mathsf{st}.ck^{\mathsf{st}.t}) \leftarrow \mathsf{KDF}_3(\mathsf{st}.rk, \mathsf{upd}^{\mathsf{ar}})$ executed in the following eSend algorithm in Line 21 in Figure 7.6 by $\mathsf{st}_\mathtt{P}.rk \xleftarrow{\$} \{0,1\}^\lambda$ and $\mathsf{st}.ck^{\mathsf{st}.t} \xleftarrow{\$} \{0,1\}^\lambda$, followed by storing $(t+1, \mathsf{st}_\mathtt{P}.rk, \mathsf{st}.ck^{t+1}, \mathsf{st}.\mathsf{prtr})$.

2. if there exist a locally stored tuple $(t', rk, ck, \mathsf{prtr})$ and the eRcv is invoked to entering epoch t' with ciphertext including prtr, the challenger replaces $(\mathsf{st}.rk, \mathsf{st}.ck^{\mathsf{st}.t}) \leftarrow \mathsf{KDF}_3(\mathsf{st}.rk, \mathsf{upd}^{\mathsf{ar}})$ executed in the eRcv algorithm in Line 32 in Figure 7.6 by $\mathsf{st}.rk \leftarrow rk$, $\mathsf{st}.ck^{\mathsf{st}.t} \leftarrow ck$.

We analyze $\mathcal{A}$'s advantage in winning **Game** C3.2.3 by hybrid games.

Game $\mathsf{hy}.0$: This game is identical to **Game** 2. Thus, we have that

$$\mathsf{Adv}_2^{C3.2} = \mathsf{Adv}_{\mathsf{hy}.0}$$

Game $\mathsf{hy}.j, (1 \leq j \leq q_{\mathsf{ep}})$: This game is identical to game **Game** $\mathsf{hy}.(j-1)$ except that:

1. When $\mathtt{P} \in \{\mathtt{A}, \mathtt{B}\}$ is trying to send the first message in a new epoch j (i.e. $\mathtt{P} = \mathtt{A}$ if j odd and $\mathtt{P} = \mathtt{B}$ if t even) and the execution $\mathcal{L}_{\mathtt{P}}^{\mathsf{cor}} \xleftarrow{+} j$ in Line 80 in the $\mathsf{ep\text{-}mgmt}$ helper function in Figure 7.6 is not triggered, then the challenger replaces $(\mathsf{st}.rk, \mathsf{st}.ck^{j}) \leftarrow \mathsf{KDF}_3(\mathsf{st}.rk, \mathsf{upd}^{\mathsf{ar}})$ executed in the following eSend algorithm in Line 21 in Figure 7.6 by $\mathsf{st}_{\mathtt{P}}.rk \xleftarrow{\$} \{0,1\}^{\lambda}$ and $\mathsf{st}.ck^{j} \xleftarrow{\$} \{0,1\}^{\lambda}$, followed by storing $(j, \mathsf{st}_{\mathtt{P}}.rk, \mathsf{st}.ck^{j}, \mathsf{st}.\mathsf{prtr})$.
2. if there exist a locally stored tuple $(t', rk, ck, \mathsf{prtr})$ and the eRcv is invoked to entering epoch t' with ciphertext including prtr, the challenger replaces $(\mathsf{st}.rk, \mathsf{st}.ck^{j}) \leftarrow \mathsf{KDF}_3(\mathsf{st}.rk, \mathsf{upd}^{\mathsf{ar}})$ executed in the eRcv algorithm in Line 32 in Figure 7.6 by $\mathsf{st}.rk \leftarrow rk$, $\mathsf{st}.ck^{j} \leftarrow ck$.

It is obvious that **Game** $\mathsf{hy}.q_{\mathsf{ep}}$ is identical to **Game** C3.1.3. Thus, we have that

$$\mathsf{Adv}_3^{C3.2} = \mathsf{Adv}_{\mathsf{hy}.q_{\mathsf{ep}}}$$

Let E denote the event that $\mathcal{A}$ can distinguish any adjacent hybrid games **Game** $\mathsf{hy}.(j-1)$ and **Game** $\mathsf{hy}.j$. Note that the modification in every hybrid game is independent of the behavior of the previous game. Thus, we have that

$$\mathsf{Adv}_2^{C3.2} - \mathsf{Adv}_3^{C3.2} \leq q_{\mathsf{ep}} \Pr[E]$$

Below, we compute the probability of the occurrence of event E by case distinction. Note that the execution $\mathcal{L}_{\mathtt{P}}^{\mathsf{cor}} \xleftarrow{+} j$ in **Game** $\mathsf{hy}.j$ indicates that **Game** $\mathsf{hy}.(j-1)$ is identical to **Game** $\mathsf{hy}.j$. Below, we only consider the case for that the execution $\mathcal{L}_{\mathtt{P}}^{\mathsf{cor}} \xleftarrow{+} j$ is not triggered. Note also that $\mathcal{L}_{\mathtt{P}}^{\mathsf{cor}} \xleftarrow{+} j$ is not triggered only when $\mathsf{safe\text{-}ch}_{\mathtt{P}}(\mathsf{flag}, j-1, \mathsf{ind})$, which further implies that one of the following conditions must hold:

1. $\Big(\mathsf{safe\text{-}st}_{\mathtt{P}}(j-1) \textbf{ and } \mathsf{safe\text{-}st}_{\neg\mathtt{P}}(j-1)\Big)$
2. $\Big(\mathsf{flag} = \mathsf{good} \textbf{ and } \mathsf{safe\text{-}st}_{\neg\mathtt{P}}(j-1)\Big)$

3. $\left(\mathsf{flag} = \mathsf{good} \textbf{ and } \mathsf{safe}^{\mathsf{idK}}_{\neg\mathsf{P}}\right)$

4. $\left(\mathsf{flag} = \mathsf{good} \textbf{ and } \mathsf{safe}^{\mathsf{preK}}_{\neg\mathsf{P}}(\mathsf{ind})\right)$

Then, we consider each of the four cases:

Case $\left(\mathsf{safe\text{-}st_P}(j-1) \textbf{ and } \mathsf{safe\text{-}st}_{\neg\mathsf{P}}(j-1)\right)$**:** Recall that $\mathsf{safe\text{-}st_P}(j-1)$ and $\mathsf{safe\text{-}st}_{\neg\mathsf{P}}(j-1)$ means $(j-1),(j-2) \notin \mathcal{L}^{\mathsf{cor}}_{\mathtt{A}}, \mathcal{L}^{\mathsf{cor}}_{\mathtt{B}}$. This indicates that (1) the execution $\mathcal{L}^{\mathsf{cor}}_{\mathsf{P}} \xleftarrow{+} (j-1)$ in **Game** $\mathsf{hy}.(j-1)$ is not triggered, and (2) the state corruption on both party is not invoked during epoch $(j-1)$. (3) the first message that $\mathtt{P}$ receives in the epoch $(j-1)$ is not forged by the attacker. According to hybrid game **Game** $\mathsf{hy}.(j-1)$, the value $\mathsf{st_P}.rk$ sampled uniformly at random during sending the first message in epoch $(j-1)$. In other words, $\mathsf{st_P}.rk$ is uniformly at random from the attacker's view when entering epoch j from $(j-1)$. During sending the first message in epoch j, $(\mathsf{st}.rk, \mathsf{st}.ck^j) \leftarrow \mathsf{KDF}_3(\mathsf{st}.rk, \mathsf{upd}^{\mathsf{ar}})$ is executed in the eSend algorithm in Line 21 in Figure 7.6. By the prf security of KDF_3, it is easy to know that if $\mathcal{A}$ can distinguish **Game** $\mathsf{hy}.(j-1)$ and **Game** $\mathsf{hy}.j$, then there must exist an attacker that distinguish the keyed KDF_3 and a random function. Thus, it holds that

$$\Pr[E] \leq \epsilon^{\mathsf{prf}}_{\mathsf{KDF}_3}$$

Case $\left(\mathsf{flag} = \mathsf{good} \textbf{ and } \mathsf{safe\text{-}st}_{\neg\mathsf{P}}(j-1)\right)$ **:** This case can be analyze in the following games. Here, we only sketch the idea, since they are very similar to **Game** C3.1.3, **Game** C1.3, **Game** C1.4, and **Game** C1.5. First, similar to analysis in **Game** C3.1.3, we know that KEM public key stored in $\mathsf{st}_{\neg\mathsf{P}}$ and will be used by $\mathtt{P}$ in epoch j is sampled uniformly at random except probability $q_{\mathsf{ep}}\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_2}$. Next, similar to **Game** C1.3, we know that the encapsulated key is indistinguishable from a random key except probability $\epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}}$ due to the $\mathsf{IND\text{-}CCA}$ security of the underlying KEM. Then, similar to **Game** C1.4, we know that the update value $\mathsf{upd}^{\mathsf{ar}}$ is indistinguishable from a random string in $\{0,1\}^\lambda$ except probability $\epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1}$ due to the $\mathsf{3prf}$ security of the KDF_1. Finally, similar to **Game** C1.5, the root key $\mathsf{st}.rk$ and the chain key $\mathsf{st}.ck^j$ are indistinguishable from random strings except probability $\epsilon^{\mathsf{swap}}_{\mathsf{KDF}_5} \leq \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5}$ due to the swap-security (and the dual-security) of the function KDF_5. Thus, we have that

$$\Pr[E] \leq q_{\mathsf{ep}}\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_2} + \epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5}$$

Case $\left(\mathsf{flag} = \mathsf{good} \textbf{ and } \mathsf{safe}^{\mathsf{idK}}_{\neg\mathsf{P}}\right)$**:** This case can be analyze in the following games. Here, we only sketch the idea, since they are very similar to **Game** C1.3, **Game** C1.4, and **Game** C1.5. First, similar to **Game** C1.3, we know that the encapsulated key is indistinguishable from a random key except probability $\epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}}$ due to the $\mathsf{IND\text{-}CCA}$ security of the underlying KEM. Then, similar to **Game** C1.4, we know that the update value $\mathsf{upd}^{\mathsf{ar}}$ is indistinguishable from a random string in $\{0,1\}^\lambda$ except probability $\epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1}$

due to the $\mathsf{3prf}$ security of the KDF_1. Finally, similar to **Game** C1.5, the root key $\mathsf{st}.rk$ and the chain key $\mathsf{st}.ck^j$ are indistinguishable from random strings except probability $\epsilon^{\mathsf{swap}}_{\mathsf{KDF}_5} \leq \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5}$ due to the swap-security (and the dual-security) of the function KDF_5. Thus, we have that

$$\Pr[E] \leq \epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5}$$

Case $\Big(\mathsf{flag} = \mathsf{good}$ **and** $\mathsf{safe\text{-}st}_{\neg\mathsf{P}}(j-1)\Big)$ **:** This case can be analyze in the following games. Here, we only sketch the idea, since they are very similar to **Game** C2.3, **Game** C2.4, **Game** C2.5, and **Game** C2.6. First, similar to analysis in **Game** C2.3, the challenger first guesses the medium-term pre-key that will be used for sending the first message in epoch j, which can be guessed correctly with probability at least $\frac{1}{q_{\mathsf{M}}}$. Next, similar to **Game** C2.4, we know that the encapsulated key is indistinguishable from a random key except probability $\epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}}$ due to the $\mathsf{IND\text{-}CCA}$ security of the underlying KEM. Then, similar to **Game** C2.5, we know that the update value $\mathsf{upd}^{\mathsf{ar}}$ is indistinguishable from a random string in $\{0,1\}^\lambda$ except probability $\epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1}$ due to the $\mathsf{3prf}$ security of the KDF_1. Finally, similar to **Game** C2.6, the root key $\mathsf{st}.rk$ and the chain key $\mathsf{st}.ck^j$ are indistinguishable from random strings except probability $\epsilon^{\mathsf{swap}}_{\mathsf{KDF}_5} \leq \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5}$ due to the swap-security (and the dual-security) of the function KDF_5.

Thus, we have that

$$\Pr[E] \leq q_{\mathsf{M}}(\epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5})$$

From above two cases, we know that

$$\begin{aligned}\Pr[E] \leq \max\Big(&\epsilon^{\mathsf{prf}}_{\mathsf{KDF}_3}, q_{\mathsf{ep}}\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_2} + \epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5},\\ &\epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5}, q_{\mathsf{M}}(\epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5})\Big)\\ \leq \max\Big(&\epsilon^{\mathsf{prf}}_{\mathsf{KDF}_3}, q_{\mathsf{ep}}\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_2} + \epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5},\\ &q_{\mathsf{M}}(\epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5})\Big)\end{aligned}$$

This means, it holds that

$$\begin{aligned}\mathsf{Adv}^{C3.2}_2 \leq \mathsf{Adv}^{C3.2}_3 + q_{\mathsf{ep}}\max\Big(&\epsilon^{\mathsf{prf}}_{\mathsf{KDF}_3}, q_{\mathsf{ep}}\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_2} + \epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}}\\ &+ \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5}, q_{\mathsf{M}}(\epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5})\Big)\end{aligned}$$

Game C3.2.4. This game is identical to **Game** 3.2.3 except the following modification:

1. For running $\mathtt{A}$'s eSend at $t^\star$, the execution $(\mathsf{st}.ck^{t^\star}, urk) \leftarrow \mathsf{KDF}_4(\mathsf{st}.ck^{t^\star})$ in Line 22 in Figure 7.6 is replaced by $\mathsf{st}.ck^{t^\star} \xleftarrow{\$} \{0,1\}^\lambda$, $urk \xleftarrow{\$} \{0,1\}^\lambda$. After that, the challenger stored $(\mathsf{st}.ck^{t^\star}, urk)$ into a local list.
2. For running $\mathtt{B}$'s eRcv at $t^\star$ the execution $(\mathsf{st}.ck^{t^\star}, urk) \leftarrow \mathsf{KDF}_4(\mathsf{st}.ck^{t^\star})$ in Line 38 is replaced by the tuple $(\mathsf{st}.ck^{t^\star}, urk)$ in the local list for the corresponding message index.

The advantage gap of $\mathcal{A}$ in winning **Game** C3.2.3 and **Game** C3.2.4 can be computed by hybrid games. Recall that $\mathcal{A}$ can query oracles at most q times, the maximum of the message index is q.

Game $\mathsf{hy}.0$: This game is identical to **Game** C3.2.3. Thus, we have that

$$\mathsf{Adv}_3^{C3.2} = \mathsf{Adv}_{\mathsf{hy}.0}$$

Game $\mathsf{hy}.j, (1 \leq j \leq q)$: This game is identical to game **Game** $\mathsf{hy}.(j-1)$ except that:

1. For running $\mathtt{A}$'s j-th eSend at $t^\star$, the execution $(\mathsf{st}.ck^{t^\star}, urk) \leftarrow \mathsf{KDF}_4(\mathsf{st}.ck^{t^\star})$ in Line 22 in Figure 7.6 is replaced by $\mathsf{st}.ck^{t^\star} \xleftarrow{\$} \{0,1\}^\lambda$, $urk \xleftarrow{\$} \{0,1\}^\lambda$. After that, the challenger stored $(\mathsf{st}.ck^{t^\star}, urk)$ into a local list.
2. For running $\mathtt{B}$'s eRcv on a ciphertext corresponds to the position $(t^\star, j)$, the execution $(\mathsf{st}.ck^{t^\star}, urk) \leftarrow \mathsf{KDF}_4(\mathsf{st}.ck^{t^\star})$ in Line 38 is replaced by the tuple $(\mathsf{st}.ck^{t^\star}, urk)$ in the local list for the corresponding message index j.

It is obvious that **Game** $\mathsf{hy}.q$ is identical to **Game** C3.2.4. So, we have that $\mathsf{Adv}_4^{C3.2} = \mathsf{Adv}_{\mathsf{hy}.q}$. The gap between every two adjacent hybrid games can be reduced to the prg security of KDF_4. Namely, if the attacker can distinguish **Game** $\mathsf{hy}.(j-1)$ from **Game** $\mathsf{hy}.j$, then there must exist an attacker can distinguish the real KDF_4 and a random number generator. Thus, we can easily have that

$$\mathsf{Adv}_3^{C3.2} \leq \mathsf{Adv}_4^{C3.2} + q\epsilon_{\mathsf{KDF}_4}^{\mathsf{prg}}$$

Game C3.2.5. This game is identical to **Game** C3.2.4 except the following modifications:

1. The challenger additionally samples a random message key $\widetilde{mk} \in \{0,1\}^\lambda$ for the position $(t^\star, i^\star)$
2. $c' \leftarrow \mathsf{SKE.Enc}(mk, m)$ in Line 22 and 40 in Figure 7.6 is replaced by $c' \leftarrow \mathsf{SKE.Enc}(\widetilde{mk}, m)$

Note that the unidirectional ratchet key urk is sampled random in **Game** C3.2.4. Similar to the game **Game** C1.5, if $\mathcal{A}$ can distinguish **Game** C3.2.4 and **Game** C3.2.5, then we can construct an attacker that breaks prf security (and therefore the dual security) of underlying KDF_5. Thus, we have that

$$\mathsf{Adv}_4^{C3.2} \leq \mathsf{Adv}_5^{C3.2} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{prf}} \leq \mathsf{Adv}_5^{C3.2} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{dual}}$$

Game Final Analysis for Case C3.2:

Similar to the final analysis for **Game** C1, if the attacker $\mathcal{A}$ can distinguish the challenge bit in **Game** C3.2.5, then there exists an attacker that breaks $\mathsf{IND\text{-}1CCA}$ security of the underlying SKE. Thus, we can easily have that

$$\mathsf{Adv}_5^{C3.2} \leq \epsilon_{\mathsf{SKE}}^{\mathsf{ind\text{-}1cca}}$$

To sum up, we have that

$$\mathsf{Adv}_2^{C3.2} \leq q_{\mathsf{ep}} \max\Big(\epsilon^{\mathsf{prf}}_{\mathsf{KDF}_3}, q_{\mathsf{ep}}\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_2} + \epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5}, q_{\mathsf{M}}(\epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5})\Big) + q\epsilon^{\mathsf{prg}}_{\mathsf{KDF}_4} + \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5} + \epsilon^{\mathsf{ind\text{-}1cca}}_{\mathsf{SKE}}$$

Combining all statements above, the proof is concluded by

$$\begin{aligned}
\epsilon^{\mathsf{PRIV}}_{\mathsf{eSM}} &\leq q \max(\mathsf{Adv}_2^{C1}, \mathsf{Adv}_2^{C2}, \mathsf{Adv}_2^{C3.1}, \mathsf{Adv}_2^{C3.2}) \\
&\leq q \max\Bigg(\epsilon^{\mathsf{ind\text{-}1cca}}_{\mathsf{SKE}} + \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}}, \\
&\quad q_{\mathsf{M}}(\epsilon^{\mathsf{ind\text{-}1cca}}_{\mathsf{SKE}} + \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}}), \\
&\quad \epsilon^{\mathsf{ind\text{-}1cca}}_{\mathsf{SKE}} + \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}} + q_{\mathsf{ep}}\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_2}, \\
&\quad q_{\mathsf{ep}} \max\Big(\epsilon^{\mathsf{prf}}_{\mathsf{KDF}_3}, q_{\mathsf{ep}}\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_2} + \epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5}, q_{\mathsf{M}}(\epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5})\Big) \\
&\qquad + q\epsilon^{\mathsf{prg}}_{\mathsf{KDF}_4} + \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5} + \epsilon^{\mathsf{ind\text{-}1cca}}_{\mathsf{SKE}}\Bigg) \\
&\leq q\Bigg(q_{\mathsf{M}}\epsilon^{\mathsf{ind\text{-}1cca}}_{\mathsf{SKE}} + q_{\mathsf{ep}}(\epsilon^{\mathsf{prf}}_{\mathsf{KDF}_3} + q_{\mathsf{ep}}\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_2}) + q_{\mathsf{M}}q_{\mathsf{ep}}(\epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5}) \\
&\quad + q\epsilon^{\mathsf{prg}}_{\mathsf{KDF}_4} + \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5}\Bigg) \\
&\leq q_{\mathsf{M}}q_{\mathsf{ep}}q\epsilon^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}} + q_{\mathsf{M}}q\epsilon^{\mathsf{ind\text{-}1cca}}_{\mathsf{SKE}} + q_{\mathsf{M}}q_{\mathsf{ep}}q\epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} \\
&\quad + q^2_{\mathsf{ep}}q\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_2} + q_{\mathsf{ep}}q\epsilon^{\mathsf{prf}}_{\mathsf{KDF}_3} + q^2\epsilon^{\mathsf{prg}}_{\mathsf{KDF}_4} + (q_{\mathsf{M}}q_{\mathsf{ep}} + 1)q\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5}
\end{aligned}$$

□

7.11.6 Proof of Lemma 5

Proof. The proof is given by a sequence of games. Let Adv_i denote the attacker $\mathcal{A}$'s advantage in winning Game i. At the beginning of the experiment, the attacker $\mathcal{A}$ outputs a target epoch $t^\star$, such that it only queries the injection oracles inputting ciphertexts corresponding to in this epoch. Without loss of generality, we assume $t^\star$ is even, i.e., $\mathtt{A}$ is the message receiver. The case for $t^\star$ is even can be given analogously. Note also that the attacker $\mathcal{A}$ can immediately win when it successfully triggers the winning predicate $\mathsf{win}^{\mathsf{auth}}$ turning form false to true. So, we only consider the case that $\mathcal{A}$ successfully forges a ciphertext only once.

Game 0. This game is identical to the $\mathrm{Expr}^{\mathsf{AUTH}}_{\Pi,\triangle_{\mathsf{eSM}}}$. Thus, we have that

$$\mathsf{Adv}_0 = \epsilon^{\mathsf{AUTH}}_{\mathsf{eSM}}$$

Game 1. This game is identical to **Game** 0 except the following modifications:

1. If the attacker queries $\mathcal{O}_{\mathsf{Inject\text{-}A}}(\mathsf{ind}, c)$ with c corresponding epoch $t^\star$ and a message index $i^\star$ such that $t^\star \leq t_{\mathtt{A}} - 2$ and $(\mathtt{B}, t^\star, i^\star) \notin \mathsf{trans}$, the challenger immediately aborts the oracle and outputs $(t^\star, i, \perp)$.

Note that a record is not included in the transcript set for the previous epochs, only when

1. this record is delivered
2. no sender has produced any message in the previous epoch $t^\star$ with message index $i^\star$

The first case can be easily excluded, since a natural eSM scheme never accepts two messages at the same position. For the second case, note that $\mathtt{B}$ produces messages only with continuous message indices. $\mathtt{B}$ didn't produce the message with message index $i^\star$ means that $i^\star$ exceeds the maximal message length that $\mathtt{B}$ has produced in the epoch $t^\star$. Since in eSM $\mathtt{A}$ has received all maximal message length in all previous epochs (see Line 30 in Figure 7.6) and will aborts the eRcv execution if i exceeds the maximal message length in the corresponding epoch (see Line 26 in Figure 7.6). This game is identical to **Game** 0 from $\mathcal{A}$'s view. Thus, we have that

$$\mathsf{Adv}_1 = \mathsf{Adv}_0$$

Note that the attacker can win only when it queries $\mathcal{O}_{\mathsf{Inject\text{-}A}}(\mathsf{ind}, c)$ such that all of the following conditions hold

1. c corresponds to epoch $t^\star$
2. $(\mathtt{B}, c) \notin \mathsf{trans}$
3. $\mathsf{ind} \leq n_{\mathtt{A}}$

4. $\mathsf{safe\text{-}inj}_{\mathtt{A}}(t_{\mathtt{A}}) = \mathsf{safe\text{-}st}_{\mathtt{B}}(t_{\mathtt{A}})$ and $\mathsf{safe\text{-}inj}_{\mathtt{A}}(t_{\mathtt{B}}) = \mathsf{safe\text{-}st}_{\mathtt{B}}(t_{\mathtt{B}})$
5. $m' \neq \perp$
6. $(\mathtt{B}, t^\star, i^\star) \notin \mathsf{comp}$

where $(\mathsf{st}_{\mathtt{A}}, t^\star, i^\star, m') \leftarrow \mathsf{eRcv}(\mathsf{st}_{\mathtt{A}}, ik_{\mathtt{A}}, prepk_{\mathtt{A}}^{\mathsf{ind}}, c)$

In particular, $(\mathtt{B}, t^\star, i^\star) \notin \mathsf{comp}$ but $(\mathtt{B}, t^\star, i^\star) \in \mathsf{trans}$ means that

1. $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \mathsf{true}$ holds at the time of $\mathtt{B}$ sending message corresponding to the position $(t^\star, i^\star)$, and
2. if $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \mathsf{false}$, $\mathcal{O}_{\mathsf{Corrupt\text{-}A}}$ cannot be queried
3. If $\mathcal{O}_{\mathsf{Corrupt\text{-}A}}$ is queried at epoch $t^\star$, then $\mathcal{O}_{\mathsf{Corrupt\text{-}B}}$ cannot be queried.
4. $\mathcal{O}_{\mathsf{Corrupt\text{-}B}}$ can be queried only after the ciphertext corresponding to $(t^\star, i^\star)$ has been honestly generated.
5. After the leakage of identity keys or pre-keys, $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \mathsf{false}$

So, at most one of $\mathcal{O}_{\mathsf{Corrupt\text{-}A}}$ and $\mathcal{O}_{\mathsf{Corrupt\text{-}B}}$ at epoch $t^\star$, but not both.

We separate the analysis for $t^\star \geq t_{\mathtt{A}} - 1$, see Case 1, or $t^\star \leq t_{\mathtt{A}} - 2$, see Case 2.

Case 1. $t^\star \geq t_{\mathtt{A}} - 1$

In this case, the attacker queries $\mathcal{O}_{\mathsf{Inject\text{-}A}}(\mathsf{ind}, c)$ for some pre-key index ind and ciphertext c under the condition that $\mathsf{safe\text{-}st}_{\mathtt{B}}(t_{\mathtt{B}}) = \mathsf{true}$. This means, $t_{\mathtt{B}}, (t_{\mathtt{B}} - 1) \notin \mathcal{L}_{\mathtt{B}}^{\mathsf{cor}}$.

Game C1.2 This game is identical to **Game** 1 except the following modification:

1. Until epoch $t^\star$, whenever $\mathtt{P} \in \{\mathtt{A}, \mathtt{B}\}$ is trying to sending the first message in a new epoch $t+1$ (i.e. $\mathtt{P} = \mathtt{A}$ if t even and $\mathtt{P} = \mathtt{B}$ if t odd) and the execution $\mathcal{L}_{\mathtt{P}}^{\mathsf{cor}} \xleftarrow{+} t+1$ in Line 80 in the $\mathsf{ep\text{-}mgmt}$ helper function in Figure 7.6 is not triggered, then the challenger replaces $r \xleftarrow{\$} \{0,1\}^\lambda$, $(\mathsf{st}_{\mathtt{P}}.nxs, r^{\mathsf{KEM}}, r^{\mathsf{DS}}) \leftarrow \mathsf{KDF}_2(\mathsf{st}_{\mathtt{P}}.nxs, r)$ executed in the following eSend algorithm in Line 18 in Figure 7.6 by $\mathsf{st}_{\mathtt{P}}.nxs \xleftarrow{\$} \{0,1\}^\lambda$, $r^{\mathsf{KEM}} \xleftarrow{\$} \{0,1\}^\lambda$, $r^{\mathsf{DS}} \xleftarrow{\$} \{0,1\}^\lambda$.

We analyze $\mathcal{A}$'s advantage in winning **Game** C1.2 by hybrid games.

Game hy.0: This game is identical to **Game** 1. Thus, we have that

$$\mathsf{Adv}_1^{C1.1} = \mathsf{Adv}_{\mathsf{hy}.0}$$

Game hy.j, $(1 \leq j \leq q_{\mathsf{ep}})$: This game is identical to game **Game** hy.$(j-1)$ except that:

1. When entering epoch j from $j-1$, if the execution $\mathcal{L}_{\mathtt{P}}^{\mathsf{cor}} \xleftarrow{+} j$ in Line 80 in the $\mathsf{ep\text{-}mgmt}$ helper function in Figure 7.6 is not triggered for $\mathtt{P} = \mathtt{A}$ if j odd and $\mathtt{P} = \mathtt{B}$ if j even, then in the following eSend algorithm, the challenger replaces $r \xleftarrow{\$} \{0,1\}^\lambda$, $(\mathsf{st}_{\mathtt{P}}.nxs, r^{\mathsf{KEM}}, r^{\mathsf{DS}}) \leftarrow \mathsf{KDF}_2(\mathsf{st}_{\mathtt{P}}.nxs, r)$ executed in Line 18 in Figure 7.6 by $\mathsf{st}_{\mathtt{P}}.nxs \xleftarrow{\$} \{0,1\}^\lambda$, $r^{\mathsf{KEM}} \xleftarrow{\$} \{0,1\}^\lambda$, $r^{\mathsf{DS}} \xleftarrow{\$} \{0,1\}^\lambda$.

It is obvious that **Game** $\mathsf{hy}.q_{\mathsf{ep}}$ is identical to **Game** C1.2. Thus, we have that

$$\mathsf{Adv}_2^{C1} = \mathsf{Adv}_{\mathsf{hy}.q_{\mathsf{ep}}}$$

Let E denote the event that $\mathcal{A}$ can distinguish any adjacent hybrid games **Game** $\mathsf{hy}.(j-1)$ and **Game** $\mathsf{hy}.j$. Note that the modification in every hybrid game is independent of the behavior of the previous game. Thus, we have that

$$\mathsf{Adv}_1^{C1} - \mathsf{Adv}_2^{C1} \leq q_{\mathsf{ep}} \Pr[E]$$

Below, we compute the probability of the occurrence of event E by case distinction. Note that the execution $\mathcal{L}_{\mathsf{P}}^{\mathsf{cor}} \overset{+}{\leftarrow} j$ in **Game** $\mathsf{hy}.j$ indicates that **Game** $\mathsf{hy}.(j-1)$ is identical to **Game** $\mathsf{hy}.j$. Below, we only consider the case for that the execution $\mathcal{L}_{\mathsf{P}}^{\mathsf{cor}} \overset{+}{\leftarrow} j$ is not triggered. Note also that $\mathcal{L}_{\mathsf{P}}^{\mathsf{cor}} \overset{+}{\leftarrow} j$ is not triggered only when $\mathsf{safe\text{-}ch_P}(\mathsf{flag}, j-1, \mathsf{ind})$ for some pre-key index ind, which further implies that one of the following conditions must hold: (1) $\mathsf{safe\text{-}st_P}(j-1)$ or (2) $\mathsf{flag} = \mathsf{good}$. Then, we consider each of the two cases.

Case $\mathsf{safe\text{-}st_P}(j-1)$**:** First, $\mathsf{safe\text{-}st_P}(j-1)$ means $(j-1), (j-2) \notin \mathcal{L}_{\mathsf{P}}^{\mathsf{cor}}$. Moreover, $(j-1) \notin \mathcal{L}_{\mathsf{P}}^{\mathsf{cor}}$ indicates that (1) the execution $\mathcal{L}_{\mathsf{P}}^{\mathsf{cor}} \overset{+}{\leftarrow} (j-2)$ in **Game** $\mathsf{hy}.(j-2)$ is not triggered, and (2) the state corruption on P is not invoked during epoch $(j-1)$ and $(j-2)$. According to hybrid game **Game** $\mathsf{hy}.(j-2)$, the value $\mathsf{st_P}.nxs$ sampled uniformly at random during sending the first message in epoch $(j-2)$. In other words, $\mathsf{st_P}.nxs$ is uniformly at random from the attacker's view when entering epoch j from $(j-1)$. During sending the first message in epoch j, $r \overset{\$}{\leftarrow} \{0,1\}^{\lambda}$, $(\mathsf{st_P}.nxs, r^{\mathsf{KEM}}, r^{\mathsf{DS}}) \leftarrow \mathsf{KDF}_2(\mathsf{st_P}.nxs, r)$ is executed in Line 18 in Figure 7.6. By the prf security of KDF_2, it is easy to know that if $\mathcal{A}$ can distinguish **Game** $\mathsf{hy}.(j-1)$ and **Game** $\mathsf{hy}.j$, then there must exist an attacker that distinguish the keyed KDF_2 and a random function. Thus, it holds that

$$\Pr[E] \leq \epsilon_{\mathsf{KDF}_2}^{\mathsf{prf}}$$

Case $\mathsf{flag} = \mathsf{good}$**:** This means, the first message in epoch $j-2$ is computed using fresh randomness. In particular, this means, $r \overset{\$}{\leftarrow} \{0,1\}^{\lambda}$, $(\mathsf{st_P}.nxs, r^{\mathsf{KEM}}, r^{\mathsf{DS}}) \leftarrow \mathsf{KDF}_2(\mathsf{st_P}.nxs, r)$ is executed in Line 18 in Figure 7.6 uses fresh randomness r. It is easy to know that $\mathsf{st_P}.nxs$ after sending the first message in epoch $(j-2)$ is distinguishable from a random string, due to the swap-security of KDF_2.

Thus, we have that

$$\Pr[E] \leq \epsilon_{\mathsf{KDF}_2}^{\mathsf{swap}}$$

From above two cases, we know that

$$\Pr[E] \leq \max\left(\epsilon_{\mathsf{KDF}_2}^{\mathsf{prf}} + \epsilon_{\mathsf{KDF}_2}^{\mathsf{swap}}\right) \leq \epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}}$$

To sum up, we have that

$$\mathsf{Adv}_1^{C1} \leq q_{\mathsf{ep}} \Pr[E] + \mathsf{Adv}_2^{C1} \leq \mathsf{Adv}_2^{C1} + q_{\mathsf{ep}} \epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}}$$

Final Analysis for Case C1.

Note that $t_{\mathtt{A}} - 1 \leq t^\star$ and that $t^\star$ even. Then, there are following seven cases:

1. $t_{\mathtt{A}}$ is even: $t_{\mathtt{A}} = t_{\mathtt{B}} = t^\star$
2. $t_{\mathtt{A}}$ is odd: $t^\star = t_{\mathtt{A}} - 1$, $t_{\mathtt{B}} = t_{\mathtt{A}} - 1$
3. $t_{\mathtt{A}}$ is odd: $t^\star = t_{\mathtt{A}} - 1$, $t_{\mathtt{B}} = t_{\mathtt{A}}$
4. $t_{\mathtt{A}}$ is odd: $t^\star = t_{\mathtt{A}} - 1$, $t_{\mathtt{B}} = t_{\mathtt{A}} + 1$
5. $t_{\mathtt{A}}$ is odd: $t^\star = t_{\mathtt{A}} + 1$, $t_{\mathtt{B}} = t_{\mathtt{A}} - 1$
6. $t_{\mathtt{A}}$ is odd: $t^\star = t_{\mathtt{A}} + 1$, $t_{\mathtt{B}} = t_{\mathtt{A}}$
7. $t_{\mathtt{A}}$ is odd: $t^\star = t_{\mathtt{A}} + 1$, $t_{\mathtt{B}} = t_{\mathtt{A}} + 1$

In all of above seven cases, $t^\star$ and $t_{\mathtt{B}}$ are not two epochs apart. Moreover, by $\mathsf{safe\text{-}st}_{\mathtt{B}}(t_{\mathtt{A}})$ amd $\mathsf{safe\text{-}st}_{\mathtt{B}}(t_{\mathtt{B}})$, we know that the $\mathcal{A}$ has to forge at least one signature against a pair of uncorrupted and freshly generated key pair, due to **Game** C1.2. To make a successful injection query, $\mathcal{A}$ has to either keep the pre-transcript and forge a signature for the pre-transcript or forge a signature for a new pre-transcript, which violates the SUF-CMA security of the underlying DS scheme. Thus, we can have that

$$\mathsf{Adv}_2^{C1} \leq \epsilon_{\mathsf{DS}}^{\mathsf{suf\text{-}cma}}$$

To sum up, we have that

$$\mathsf{Adv}_1^{C1} \leq \epsilon_{\mathsf{DS}}^{\mathsf{suf\text{-}cma}} + q_{\mathsf{ep}} \epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}}$$

Case 2. $t^\star \leq t_{\mathtt{A}} - 2$

In this case, $\mathcal{A}$ aims to forge a ciphertext in a past epoch. By **Game** 1, we know that $(t^\star, i^\star) \in \mathsf{trans}$, where $i^\star$ denotes the message index corresponding to the forged ciphertext.

Game C2.2 This game is identical to **Game** 1 except the following modification:

1. The challenger directly outputs $(t^\star, i, \perp)$ for answering any $\mathcal{O}_{\mathsf{Inject\text{-}A}}(\mathsf{ind}, c)$ if $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \mathsf{true}$, where $(t^\star, i)$ is the position of c.

Note that $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \mathsf{true}$ holds at the time of $\mathtt{B}$ sending message corresponding to the position $(t^\star, i^\star)$ for some $i^\star$. This means, $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \mathsf{true}$ when $\mathtt{B}$ was switch from receiver to sender when entering epoch $t^\star$. Similar to the analysis in **Game** C1.2, we know that the signing keys are randomly sampled except probability at most $q_{\mathsf{ep}} \epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}}$. If $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \mathsf{true}$ at the time of any $\mathcal{O}_{\mathsf{Inject\text{-}A}}$ query, the signing key has not been corrupted.

Similar to the final analysis of Game C1.2, if $\mathcal{A}$ can forge a ciphertext, then we can construct another attacker that invokes $\mathcal{A}$ to break the SUF-CMA security of DS. Thus, we have that

$$\mathsf{Adv}_1^{C2} \leq \mathsf{Adv}_2^{C2} + \epsilon_{\mathsf{DS}}^{\mathsf{suf\text{-}cma}} + q_{\mathsf{ep}}\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}}$$

In the games below, we assume that $\mathsf{safe\text{-}st_B}(t^\star) = \mathsf{false}$ when $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Inject\text{-}A}}$. Recall that $\mathcal{O}_{\mathsf{Corrupt\text{-}B}}$ can be queried only after the ciphertext corresponding to $(t^\star, i^\star)$ has been honestly generated. This also means that the unidirectional ratchet key urk for encrypting and decrypting the ciphertext corresponding position $(t^\star, i^\star)$ has been removed from the state $\mathsf{st_B}$. Moreover, if $\mathcal{O}_{\mathsf{Corrupt\text{-}B}}$ is queried, then $\mathcal{O}_{\mathsf{Corrupt\text{-}A}}$ cannot be queried.

Game C2.3 This game is identical to **Game** C2.2 except the following modification:

1. Until epoch $t^\star$. Whenever $\mathtt{P} \in \{\mathtt{A}, \mathtt{B}\}$ is trying to sending the first message in a new epoch $t+1$ (i.e. $\mathtt{P} = \mathtt{A}$ if t even and $\mathtt{P} = \mathtt{B}$ if t odd) and the execution $\mathcal{L}_{\mathsf{P}}^{\mathsf{cor}} \xleftarrow{+} t+1$ in Line 80 in the ep-mgmt helper function in Figure 7.6 is not triggered, then the challenger replaces $(\mathsf{st}.rk, \mathsf{st}.ck^{\mathsf{st}.t}) \leftarrow \mathsf{KDF}_3(\mathsf{st}.rk, \mathsf{upd}^{\mathsf{ar}})$ executed in the following eSend algorithm in Line 21 in Figure 7.6 by $\mathsf{st_P}.rk \xleftarrow{\$} \{0,1\}^\lambda$ and $\mathsf{st}.ck^{\mathsf{st}.t} \xleftarrow{\$} \{0,1\}^\lambda$, followed by storing $(t+1, \mathsf{st_P}.rk, \mathsf{st}.ck^{t+1}, \mathsf{st.prtr})$.
2. if there exist a locally stored tuple $(t', rk, ck, \mathsf{prtr})$ and the eRcv is invoked to entering epoch t' with ciphertext including prtr, the challenger replaces $(\mathsf{st}.rk, \mathsf{st}.ck^{\mathsf{st}.t}) \leftarrow \mathsf{KDF}_3(\mathsf{st}.rk, \mathsf{upd}^{\mathsf{ar}})$ executed in the eRcv algorithm in Line 32 in Figure 7.6 by $\mathsf{st}.rk \leftarrow rk$, $\mathsf{st}.ck^{\mathsf{st}.t} \leftarrow ck$.

The analysis of this game is identical to **Game** C3.2.3 in Section 7.11.5. We can easily know that

$$\begin{aligned}\mathsf{Adv}_2^{C2} \leq & \mathsf{Adv}_3^{C2} + q_{\mathsf{ep}} \max\Big(\epsilon_{\mathsf{KDF}_3}^{\mathsf{prf}}, q_{\mathsf{ep}}\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}} + \epsilon_{\mathsf{KEM}}^{\mathsf{ind\text{-}cca}} \\ & + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{dual}}, q_{\mathsf{M}}(\epsilon_{\mathsf{KEM}}^{\mathsf{ind\text{-}cca}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{dual}})\Big)\end{aligned}$$

Game C2.4 This game is identical to **Game** C2.3 except the following modification until $\mathcal{O}_{\mathsf{Corrupt\text{-}B}}$ is invoked:

1. For running $\mathtt{A}$'s eSend at $t^\star$, the execution $(\mathsf{st}.ck^{t^\star}, urk) \leftarrow \mathsf{KDF}_4(\mathsf{st}.ck^{t^\star})$ in Line 22 in Figure 7.6 is replaced by $\mathsf{st}.ck^{t^\star} \xleftarrow{\$} \{0,1\}^\lambda$, $urk \xleftarrow{\$} \{0,1\}^\lambda$. After that, the challenger stored $(\mathsf{st}.ck^{t^\star}, urk)$ into a local list.
2. For running $\mathtt{B}$'s eRcv at $t^\star$ the execution $(\mathsf{st}.ck^{t^\star}, urk) \leftarrow \mathsf{KDF}_4(\mathsf{st}.ck^{t^\star})$ in Line 38 is replaced by the tuple $(\mathsf{st}.ck^{t^\star}, urk)$ in the local list for the corresponding message index.

The advantage gap of $\mathcal{A}$ in winning **Game** C3.2.3 and **Game** C3.2.4 can be computed by hybrid games and reduced to the prg security of KDF_4. Note that $\mathcal{A}$ can query at most q, we can easily have that

$$\mathsf{Adv}_3^{C2} \leq \mathsf{Adv}_4^{C2} + q\epsilon_{\mathsf{KDF}_4}^{\mathsf{prg}}$$

Game C2.5. In this game, the challenger guesses the message index $i^\star$ that $\mathcal{A}$ wants to attack. Note that $\mathcal{A}$ can query at most q times oracles. The challenger guesses correctly with probability at least $\frac{1}{q}$. Thus, we have that

$$\mathsf{Adv}_4^{C2} \leq q\mathsf{Adv}_5^{C2}$$

Game C2.6. This game is identical to **Game** C2.5 except the following modifications:

1. The challenger additionally samples a random message key $\widetilde{mk} \in \{0,1\}^\lambda$ for the position $(t^\star, i^\star)$
2. If the pre-key index ind equals the one for producing ciphertext at position $(t^\star, i^\star)$ and the KEM ciphertext are same as produced before, the challenger replaces $c' \leftarrow \mathsf{SKE.Enc}(mk, m)$ in Line 22 and 40 in Figure 7.6 by $c' \leftarrow \mathsf{SKE.Enc}(\widetilde{mk}, m)$. Otherwise, the challenger samples another random key $\widetilde{mk}' \in \{0,1\}^\lambda$ for decrypting ciphertext at location $(t^\star, i^\star)$.

Note that the unidirectional ratchet key urk is sampled random in **Game** C2.4. If $\mathcal{A}$ can distinguish **Game** C2.5 and **Game** C2.6, then we can construct an attacker that breaks prf security (and therefore the dual security) of underlying KDF_5. Thus, we have that

$$\mathsf{Adv}_5^{C2} \leq \mathsf{Adv}_6^{C2} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{prf}} \leq \mathsf{Adv}_6^{C2} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{dual}}$$

Game C2.7. This game is identical to **Game** C2.6 except the following modifications:

1. If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Inject\text{-}A}}(\mathsf{ind}, c)$ such that
   (a) c corresponds to the position $(t^\star, i^\star)$
   (b) ind does not equal the one for producing the ciphertext at position $(t^\star, i^\star)$ or the KEM ciphertexts included in c do not equal the ones in the original ciphertext at position $(t^\star, i^\star)$

   then the challenger simply returns $(t^\star, i^\star, \perp)$

The gap between **Game** C2.6 and **Game** C2.7 can be reduced to the $\mathsf{IND\text{-}1CCA}$ security of SKE. The reduction simulates **Game** C2.6 honestly except for the $\mathcal{O}_{\mathsf{Inject\text{-}A}}(\mathsf{ind}, c)$ that is described above. In this case, the reduction forwards the symmetric key ciphertext to its decryption oracle for a reply m'. Then, the reduction returns $(t^\star, i^\star, m')$ to $\mathcal{A}$. If the challenge bit is 0, then the reduction simulates **Game** C2.6 honestly, otherwise, it simulates **Game** C2.7. Thus, if $\mathcal{A}$ can distinguish **Game** C2.6 and **Game** C2.7, then the reduction can easily distinguish the challenge bit. Thus, we have that

$$\mathsf{Adv}_6^{C2} \leq \mathsf{Adv}_7^{C2} + \epsilon_{\mathsf{SKE}}^{\mathsf{ind\text{-}1cca}}$$

Game C2.8. This game is identical to **Game** C2.7 except the following modifications:

1. If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Inject\text{-}A}}(\mathsf{ind}, c)$ such that
   (a) c corresponds to the position $(t^\star, i^\star)$
   (b) ind equals the one for producing the ciphertext at position $(t^\star, i^\star)$ and the KEM ciphertexts included in c equal the ones in the original ciphertext at position $(t^\star, i^\star)$

   then the challenger simply returns $(t^\star, i^\star, \perp)$

The gap between **Game** C2.7 and **Game** C2.8 can be reduced to the $\mathsf{IND\text{-}1CCA}$ security of SKE. The reduction simulates **Game** C2.7 honestly except for the $\mathcal{O}_{\mathsf{Transmit\text{-}B}}(\mathsf{ind}, m, r)$ and $\mathcal{O}_{\mathsf{Inject\text{-}A}}(\mathsf{ind}, c)$ that is described above.

For the $\mathcal{O}_{\mathsf{Transmit\text{-}B}}(\mathsf{ind}, m, r)$ query, the reduction forwards m to its encryption oracle for a ciphertext c'. The rest of this oracle is honestly simulated.

For the $\mathcal{O}_{\mathsf{Inject\text{-}A}}(\mathsf{ind}, c)$ query, the reduction forwards symmetric key ciphertext in the c to its decryption oracle for a reply m'. Then, the reduction returns $(t^\star, i^\star, m')$ to $\mathcal{A}$.

If the challenge bit is 0, then the reduction simulates **Game** C2.7 honestly, otherwise, it simulates **Game** C2.8. Thus, if $\mathcal{A}$ can distinguish **Game** C2.7 and **Game** C2.8, then the reduction can easily distinguish the challenge bit. Thus, we have that

$$\mathsf{Adv}_7^{C2} \leq \mathsf{Adv}_8^{C2} + \epsilon_{\mathsf{SKE}}^{\mathsf{ind\text{-}1cca}}$$

Final Analysis for Case C2:

Note that no matter what kind of $\mathcal{O}_{\mathsf{Inject\text{-}A}}(\mathsf{ind}, c)$ query $\mathcal{A}$ asks, where c corresponds to the position $(t^\star, i^\star)$, the challenger always returns $(t^\star, i^\star, \perp)$ immediately, according to **Game** C2.7 and **Game** C2.8. Thus, $\mathcal{A}$ can never win and we have that

$$\mathsf{Adv}_8^{C2} = 0$$

To sum up, we have that

$$\begin{aligned}
\mathsf{Adv}_1^{C2} \leq\ & \epsilon_{\mathsf{DS}}^{\mathsf{suf\text{-}cma}} + q_{\mathsf{ep}}\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}} + q\epsilon_{\mathsf{KDF}_4}^{\mathsf{prg}} + q(\epsilon_{\mathsf{KDF}_5}^{\mathsf{dual}} + 2\epsilon_{\mathsf{SKE}}^{\mathsf{ind\text{-}1cca}}) \\
& + q_{\mathsf{ep}} \max\Big(\epsilon_{\mathsf{KDF}_3}^{\mathsf{prf}}, q_{\mathsf{ep}}\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}} + \epsilon_{\mathsf{KEM}}^{\mathsf{ind\text{-}cca}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{dual}}, \\
& \quad q_{\mathsf{M}}(\epsilon_{\mathsf{KEM}}^{\mathsf{ind\text{-}cca}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{dual}})\Big) \\
\leq\ & \epsilon_{\mathsf{DS}}^{\mathsf{suf\text{-}cma}} + q(\epsilon_{\mathsf{KDF}_4}^{\mathsf{prg}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{dual}} + 2\epsilon_{\mathsf{SKE}}^{\mathsf{ind\text{-}1cca}}) \\
& + q_{\mathsf{ep}}\Big(\epsilon_{\mathsf{KDF}_3}^{\mathsf{prf}} + (q_{\mathsf{ep}}+1)\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}} + q_{\mathsf{M}}(\epsilon_{\mathsf{KEM}}^{\mathsf{ind\text{-}cca}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{dual}})\Big)
\end{aligned}$$

The following equation concludes the proof.

$$\begin{aligned}
\epsilon_{\mathsf{eSM}}^{\mathsf{AUTH}} &\leq \max(\mathsf{Adv}_1^{C1}, \mathsf{Adv}_1^{C2}) \\
&\leq \max\Bigg(\epsilon_{\mathsf{DS}}^{\mathsf{suf\text{-}cma}} + q_{\mathsf{ep}}\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}}, \epsilon_{\mathsf{DS}}^{\mathsf{suf\text{-}cma}} + q(\epsilon_{\mathsf{KDF}_4}^{\mathsf{prg}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{dual}} + 2\epsilon_{\mathsf{SKE}}^{\mathsf{ind\text{-}1cca}}) \\
&\quad + q_{\mathsf{ep}}\Big(\epsilon_{\mathsf{KDF}_3}^{\mathsf{prf}} + (q_{\mathsf{ep}}+1)\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}} + q_{\mathsf{M}}(\epsilon_{\mathsf{KEM}}^{\mathsf{ind\text{-}cca}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{dual}})\Big)\Bigg) \\
&\leq \epsilon_{\mathsf{DS}}^{\mathsf{suf\text{-}cma}} + q(\epsilon_{\mathsf{KDF}_4}^{\mathsf{prg}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{dual}} + 2\epsilon_{\mathsf{SKE}}^{\mathsf{ind\text{-}1cca}}) \\
&\quad + q_{\mathsf{ep}}\Big(\epsilon_{\mathsf{KDF}_3}^{\mathsf{prf}} + (q_{\mathsf{ep}}+1)\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}} + q_{\mathsf{M}}(\epsilon_{\mathsf{KEM}}^{\mathsf{ind\text{-}cca}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{dual}})\Big) \\
&\leq \epsilon_{\mathsf{DS}}^{\mathsf{suf\text{-}cma}} + q_{\mathsf{ep}}q_{\mathsf{M}}\epsilon_{\mathsf{KEM}}^{\mathsf{ind\text{-}cca}} + 2q\epsilon_{\mathsf{SKE}}^{\mathsf{ind\text{-}1cca}} + q_{\mathsf{ep}}q_{\mathsf{M}}\epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} \\
&\quad + q_{\mathsf{ep}}(q_{\mathsf{ep}}+1)\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}} + q_{\mathsf{ep}}\epsilon_{\mathsf{KDF}_3}^{\mathsf{prf}} + q\epsilon_{\mathsf{KDF}_4}^{\mathsf{prg}} + (q_{\mathsf{ep}}q_{\mathsf{M}} + q)\epsilon_{\mathsf{KDF}_5}^{\mathsf{dual}}
\end{aligned}$$

□

7.11.7 Proof of Theorem 29

Proof. The proof is given by reduction. Namely, if there exists an attacker $\mathcal{A}$ that breaks the offline deniability for the composition of a DAKE scheme Σ and our eSM construction Π in Section 7.5.2, then we can always construct an attacker $\mathcal{B}$ that breaks the offline deniability of Σ in terms of Definition 61, also see [63, Definition 11].

We first define the function $\mathsf{Fake}_{\Pi}^{\mathsf{eInit}}$ and the function $\mathsf{Fake}_{\Pi}^{\mathsf{eSend}}$ for our eSM construction Π.

- $\mathsf{Fake}_{\Pi}^{\mathsf{eInit}}(K, ipk_{\mathsf{did}}, ik_{\mathsf{aid}}, \mathcal{L}_{\mathsf{aid}}^{prek}, \mathsf{sid}, \mathsf{rid}, \mathsf{aid}, \mathsf{did})$: this algorithm inputs a key $K \in iss$, identity public keys $ipk_{\mathtt{A}}$ and $ipk_{\mathtt{B}}$, a list of private pre-keys $\mathcal{L}_{\mathsf{rid}}^{prek}$, the sender identity sid, the receiver identity rid, the accuser identity aid, and the defendant identity did, followed by executing the following steps:

```
1. $\mathsf{st}_{\mathtt{A}} \xleftarrow{\$} \Pi.\mathsf{eInit\text{-}A}(K)$
2. $\mathsf{st}_{\mathtt{B}} \xleftarrow{\$} \Pi.\mathsf{eInit\text{-}B}(K)$
3. $\mathsf{st}_{\mathsf{Fake}} \leftarrow \Big((\mathsf{st}_{\mathtt{A}}, \mathsf{rid}), (\mathsf{st}_{\mathtt{B}}, \mathsf{sid})\Big)$
4. **return** $\mathsf{st}_{\mathsf{Fake}}$
```

- $\mathsf{Fake}_{\Pi}^{\mathsf{eSend}}(\mathsf{st}_{\mathsf{Fake}}, ipk, prepk, m, \mathsf{sid}, \mathsf{rid}, \mathsf{ind})$: this algorithm inputs a fake state $\mathsf{st}_{\mathsf{Fake}}$, an public identity key ipk, a public pre-key $prepk$, a message m, a sender identity sid, a receiver identity rid, and a pre-key index ind, followed by executing the following steps:

```
1. Parse $\Big((\mathsf{st}_{\mathtt{A}}, \mathsf{id}_{\mathtt{A}}), (\mathsf{st}_{\mathtt{B}}, \mathsf{id}_{\mathtt{B}})\Big) \leftarrow \mathsf{st}_{\mathsf{Fake}}$
2. **if** $\mathsf{id}_{\mathtt{A}} = \mathsf{sid}$, **then**
   (a) $(\mathsf{st}_{\mathtt{A}}, c) \xleftarrow{\$} \Pi.\mathsf{eSend}(\mathsf{st}_{\mathtt{A}}, ipk, prepk, m)$
   (b) copy all symmetric values in session state $\mathsf{st}_{\mathtt{A}}$ to session state $\mathsf{st}_{\mathtt{B}}$
   (c) If $\mathsf{st}_{\mathtt{A}}.t$ is incremented in the above $\Pi.\mathsf{eSend}$ invocation, then extract the new
       verification key $vk$ and new encryption key $ek$ from $c$, followed by set $vk$ and
       $ek$ into $\mathsf{st}_{\mathtt{B}}$
   (d) $\mathsf{st}_{\mathsf{Fake}} \leftarrow ((\mathsf{st}_{\mathtt{A}}, \mathsf{id}_{\mathtt{A}}), (\mathsf{st}_{\mathtt{B}}, \mathsf{id}_{\mathtt{B}}))$
3. **else**
   (a) $(\mathsf{st}_{\mathtt{B}}, c) \xleftarrow{\$} \Pi.\mathsf{eSend}(\mathsf{st}_{\mathtt{B}}, ipk, prepk, m)$
   (b) copy all symmetric values in session state $\mathsf{st}_{\mathtt{B}}$ to session state $\mathsf{st}_{\mathtt{A}}$
   (c) If $\mathsf{st}_{\mathtt{B}}.t$ is incremented in the above $\Pi.\mathsf{eSend}$ invocation, then extract the new
       verification key $vk$ and new encryption key $ek$ from $c$, followed by set $vk$ and
       $ek$ into $\mathsf{st}_{\mathtt{A}}$
   (d) $\mathsf{st}_{\mathsf{Fake}} \leftarrow ((\mathsf{st}_{\mathtt{A}}, \mathsf{id}_{\mathtt{A}}), (\mathsf{st}_{\mathtt{B}}, \mathsf{id}_{\mathtt{B}}))$
```

At the beginning of the experiment, the attacker $\mathcal{B}$ inputs a list $\mathcal{L}_{\mathsf{all}}$ that includes all public-private key pairs of Σ from its challenger. Next, $\mathcal{B}$ honestly samples the random identity key and pre-key pairs of Π and sets them into the respective lists as in the $\mathrm{Expr}^{\mathsf{deni}}_{\Sigma,\Pi,q_{\mathsf{P}},q_{\mathsf{M}},q_{\mathsf{S}}}$. In particular, all public-private key pairs are added into the list $\mathcal{L}_{\mathsf{all}}$. $\mathcal{B}$ also initializes a empty dictionary $\mathcal{D}_{\mathsf{session}}$ and a counter n to 0. Then, $\mathcal{B}$ sends the list $\mathcal{L}_{\mathsf{all}}$ to $\mathcal{A}$.

When $\mathcal{A}$ queries $\mathsf{Session\text{-}Start}(\mathsf{sid}, \mathsf{rid}, \mathsf{aid}, \mathsf{did}, \mathsf{ind})$, $\mathcal{B}$ first checks whether $\{\mathsf{sid}, \mathsf{rid}\} = \{\mathsf{aid}, \mathsf{did}\}$ and $\mathsf{sid} \neq \mathsf{rid}$ holds. It either condition does not hold, $\mathcal{B}$ simply aborts the oracle. Next, $\mathcal{B}$ increments the counter n, followed by adding $\{\mathsf{sid}, \mathsf{rid}\}$ into the dictionary $\mathcal{D}_{\mathsf{session}}[n]$. Then, $\mathcal{B}$ checks whether $\mathsf{aid} = \mathsf{sid}$. If the conditions holds, then $\mathcal{B}$ simply honestly runs Σ on the corresponding input and finally derives a key $K \in \mathit{iss}$ and a transcript T. Otherwise, $\mathcal{B}$ queries its challenge oracle with the input $(\mathsf{sid}, \mathsf{rid}, \mathsf{ind})$ for the key K and the transcript T. After that, $\mathcal{B}$ runs the above defined function $\mathsf{Fake}^{\mathsf{eInit}}_{\Pi}(K, \mathit{ipk}_{\mathsf{did}}, \mathit{ik}_{\mathsf{aid}}, \mathcal{L}^{\mathit{prek}}_{\mathsf{aid}}, \mathsf{sid}, \mathsf{rid}, \mathsf{aid}, \mathsf{did})$ for a fake state $\mathsf{st}^{n}_{\mathsf{Fake}}$. Finally, $\mathcal{B}$ returns the transcript to $\mathcal{A}$.

When $\mathcal{A}$ queries $\mathsf{Session\text{-}Execute}(\mathsf{sid}, \mathsf{rid}, i, \mathsf{ind}, m)$, $\mathcal{B}$ simply simulates $\mathsf{Session\text{-}Execute}$ as if the bit $\mathsf{b} = 1$.

At the end of the experiment, when $\mathcal{A}$ outputs a bit b', $\mathcal{B}$ then forwards it to its challenger.

Note that our $\mathsf{Fake}^{\mathsf{eInit}}_{\Pi}$ algorithm perfectly simulates the process of running $\Pi.\mathsf{eInit\text{-}A}$ and $\Pi.\mathsf{eInit\text{-}B}$. Moreover, we consider two cases for the queries to the $\mathsf{Session\text{-}Execute}$ oracle:

1. If the sender identity sid in the $\mathsf{Session\text{-}Execute}$ oracle query is id_{A}. Note that when a party receives a message from the partner in our eSM construction Π, it only passively updates the symmetric state, and optionally update the verification key and encryption key from the partner. In this case, our $\mathsf{Fake}^{\mathsf{eSend}}_{\Pi}$ algorithm perfectly simulates the case that id_{A} sends messages to id_{B}.
2. If the sender identity sid in the $\mathsf{Session\text{-}Execute}$ oracle query is id_{B}. In this case, similar to the analysis above, our $\mathsf{Fake}^{\mathsf{eSend}}_{\Pi}$ algorithm also perfectly simulates the case that id_{B} sends messages to id_{A}.

To sum up, in both cases $\mathcal{B}$ perfectly simulates $\mathrm{Expr}^{\mathsf{deni}}_{\Sigma,\Pi,q_{\mathsf{P}},q_{\mathsf{M}},q_{\mathsf{S}}}$ to $\mathcal{A}$. Thus, $\mathcal{B}$ wins if and only if $\mathcal{A}$ wins. Obviously, the number of sessions at least as many as the number of challenge oracles that $\mathcal{B}$ queries. And $\mathcal{A}$ and $\mathcal{B}$ runs in the approximately same time, which concludes the proof. $\square$

Chapter 8

Conclusions

This thesis sets out to provide the missing provable security analysis of five real-world protocols, discloses the potential vulnerabilities, and propose their feasible mitigation. These protocols are:

1. one of the most efficient digital signature implementations Ed25519,
2. the generic authenticated encryption with associated data (AEAD) schemes,
3. the latest version of passwordless authentication standard FIDO2,
4. the popular video conferencing library Zoom, and
5. our own proposal of extended secure messaging protocol eSM.

Each of these protocols has various characteristics and can be categorized into several representative research domains, based on different classification criteria. I summarize some characteristics of these five protocols in Table 8.1. Moreover, our security analysis covers not only the heart of information security "CIA" (**C**onfidentiality, **I**ntegrity, and **A**vailability) but also several state-of-the-art security properties that surface and have become relevant only recently in our modern life. I summarize the security properties that this thesis investigated for these five protocols in Table 8.2. I expect this thesis to be able to motivate and impact further research in a wider scope of cryptography.

In this chapter, I conclude this thesis by a brief summary of contributions in Section 8.1 and some open challenges in Section 8.2.

8.1 Contributions

A brief summary of our contributions follows:

- In Chapter 3:
    1. We provide the first detailed proof that Ed25519-Original [43] is indeed EUF-CMA secure.

	Interactivity	Synchronicity	Symmetry	Use Case
Chapter 3: Ed25519	non-interactive	-	asymmetric	-
Chapter 4: AEAD	non-interactive	-	symmetric	-
Chapter 5: FIDO2	interactive	synchronous	-	two-party
Chapter 6: Zoom	interactive	synchronous	-	group
Chapter 7: eSM	interactive	asynchronous	-	two-party

Table 8.1: Summary of characteristics of five protocols in this thesis. We use "-" to denote that a characteristic is irrelevant for a protocol.

	Security Properties
Chapter 3: Ed25519	integrity, resilience against key substitution attacks
Chapter 4: AEAD	privacy, integrity, collision resistance, input-bound ciphertext, full robustness, key committing, multi-key collision resistance, receiver binding
Chapter 5: FIDO2	user authentication, resilience against downgrade attacks, post-quantum security
Chapter 6: Zoom	(implicit) group key authentication, group key secrecy, perfect forward secrecy, (implicit) group member authentication, (full) end-to-end security
Chapter 7: eSM	resilience against fine-grained state compromise, immediate decryption, temporal privacy, correctness, forward secrecy, post-compromise security, strong authenticity, strong privacy, randomness leakage/failures, state compromise/failures, periodic privacy recovery, offline deniability, post-quantum security

Table 8.2: Summary of security properties of five protocols in this thesis.

2. We provide the first proof that Ed25519-IETF [121] is actually SUF-CMA secure.
3. We prove that all Ed25519 schemes are resilient against key substitution attacks, and that if small subgroup keys are rejected as in LibSodium, a signature uniquely identifies a message, even for malicious keys.
4. In a wider sense, our results retroactively support the standardisation of Ed25519-IETF, and support the ongoing standardisation by NIST.

- In Chapter 4:

1. We formally prove some well-known but merely conjectured relations for AEAD between the fundamental privacy and integrity.
2. We formally prove the missing or conjectured relations between existing AEAD security notions w.r.t. collision resistance, completing the picture in the domain.

- In Chapter 5:

1. We prove that FIDO2 with WebAuthn 2 and CTAP 2.1 is provably secure against classical attackers in a fine-grained security and protocol model. Our security models

are more fine-grained or cover other aspects than previous versions such as [21, 103]. For example, we add important aspects such as algorithm negotiation, required user actions, and token binding. For CTAP 2.1, our security proofs confirm the stronger containment properties (reduced "blast radius") offered by the protocol compared to CTAP 2.0. Our analysis of WebAuthn 2 also has new implications for WebAuthn 1: we provide the first guarantees of the most widely used `None` attestation mode, user verification, user presence, and token binding.

2. We prove that if FIDO2 with WebAuthn 2 and CTAP 2.1 is instantiated with post-quantum (PQ) secure KEMs and signatures, then it is secure against quantum attackers in the same model. We give concrete suggestions for PQ secure algorithm and negotiation design choices, including classical-PQ hybrids as suggested by standardization agencies, such as NIST (National Institute for Standards and Technology) [69].
3. We propose a simple improvement to WebAuthn 2 that improves its resilience to certain types of downgrade attack. While these can only occur for strong threat models, these improvements yield stronger classical security against broken cryptographic primitives, and are even more relevant for their PQ instantiations.

- In Chapter 6:

1. We develop a solution to improve the security of Zoom-like apps against malicious servers, without introducing new security elements. The core observation is that Zoom already uses group-specific passwords, but they are by design known to the server. By leveraging techniques from password-authenticated key exchange, we can get rid of the reliance on the server for trusted channels.
2. To formally prove the security of our solution, we need to develop substantial machinery. We propose a formal model and syntax of multi-stage group key distribution protocols, called mGKD, of which Zoom can be seen as an instance. For such protocols, we develop a basic security notion $\mathsf{Sec\text{-}mGKD\text{-}pki}$, which assumes the server did not interfere with the public keys of a group's participants, and prove that Zoom meets this notion. We show how real-world attacks manifest in this basic notion and notably how malicious zoom servers can manipulate groups.
3. We formally prove that our transformation turns a protocol that is $\mathsf{Sec\text{-}mGKD\text{-}pki}$ secure into one that is also secure in a model that makes no assumptions on the server but only on the password, which we call $\mathsf{Sec\text{-}mGKD\text{-}pw}$.
4. We show how to efficiently apply our transformation to the Zoom version 4.0 protocol to obtain the ZoomPAKE protocol, in which the server no longer knows the password, and groups are protected against malicious servers.

- In Chapter 7:

1. Our main contribution is the first provably secure messaging protocol with immediate decryption and constant-size overhead, temporal privacy, and resilience against fine-grained state compromise. To this end, we introduce a related new strong security notion called Extended-Secure-Messaging (eSM). We show that the eSM notion covers above strong properties and prove that our protocol meets it, in particular, in the PQ setting.
2. Furthermore, to show that our protocol is a suitable PQ-secure candidate for the DR in Signal, which is provably offline deniable, we extend the offline deniability definition for SPQR [63] (currently the only provably secure PQ-asynchronous key establishment) to the multi-stage setting. We prove that the combination of our eSM-secure protocol and SPQR is offline deniable, making it the first full messaging protocol that is provably offline deniable in the PQ setting.

8.2 Open Challenges

Below, I will introduce the open challenges that I am interested in and the ultimate implications of their resolutions.

Open Questions. During my doctoral research, I observed that the conventional research approach to proposing a new cryptographically secure protocol design is often given in the following two steps: (1) first formally model the desired security requirements, and (2) then prove that there are met in some threat model either by an existing protocol or a novel proposal. However, such a conventional approach has many shortcomings:

- First, we define concrete threat models for a class of protocols that following the same syntax. This threat model is expected to capture some target attackers' capabilities and desired security guarantees.
- Second, we introduce concrete protocols, which either exist in the literature, underlie some real-world applications, or are designed by ourselves.
- Third, we claim that these protocols are secure in the defined threat models, followed by formally proving our claims.

1. **The existing protocol designs often lack generality**: their constructions and instantiations are concrete and fixed. In real life, any currently state-of-the-art protocol might become out-of-date, since some attackers with stronger capability might appear in the future. For example, CTAP 2.0 underlying FIDO2 concretely deploys Diffie-Hellman key exchange, the security of which is broken against quantum attackers. To improve the security against quantum attackers in the future, the FIDO alliance has to propose a new FIDO2 with CTAP 2.1 and we propose its PQ-secure generic hybrid instantiation.

2. **The existing threat models often lack generality**: some of them claim to capture the same security goals but indeed interpret them independently to different extents. This complicates the comparison between different threat models and prevents developers from quickly figuring out the most suitable candidates for the desired security requirements. For example, there are a number of secure messaging designs in the literature, all aiming at strong privacy and authenticity, e.g., "optimal", "almost-optimal", "sub-optimal", and "ID-optimal" security, the differences between which are however very subtle and vague. This potentially leads their comparisons with our eSM proposal, which aims at the balance between strong security and efficiency, to be even more complicated.
3. **The existing security analyses often lack generality**: they are merely subject to a given protocol and threat model and do not allow for even slight modifications in the design. This causes an unnecessarily high workload for the analysis of different but similar threat models, in particular for large-scale communication protocols, which always require extremely high effort on the proof due to the cumbersome methodology in complicated threat models. On the one hand, this makes the proofs too obscure to be verified even by experts. On the other hand, this triggers the urgent deployment of some modern protocol designs without comprehensive and timely analysis. For example, although a lot of adjacent related work to real-time group protocols exists in the literature, their analyses cannot directly apply to Zoom. We are able to formally analyze the security of Zoom, disclose the loss of full E2EE security, and propose mitigation only in the post-Covid-19 era.

The above observations motivate my invention of a novel and more efficient methodology for security analyses and protocol designs. My future plan will focus on the following open questions:

1. [**Generic Protocol Designs**] How to generically recombine different cryptographic building blocks into full-fledged proposals in a (possibly) intertwined manner?
2. [**Generic Security Analyses**] How to inherit the security of the composed proposal from the security of the underlying building blocks?
3. [**Generic Threat Models**] How to easily compare the security of different re-combinations?

Implications. The resolution of the open questions has the following implications:

1. **Simplified designs and reduced workload**. Developers can freely choose and combine the desired building blocks according to their needs (e.g., bandwidth or storage limits), as well as understand the final guarantees. Meanwhile, only the security analysis of each (generic) building block, rather than the full protocols, is necessary.

2. **Interchangeability**. It suffices to swap one concrete instantiation (of a generic building block) with another to obtain security against attackers with stronger capability, e.g., post-quantum security.
3. **Generality**. The security of all generic building blocks and their instantiations can be repeatedly used in the security analysis of different messaging protocols.

Bibliography

[1] Michel Abdalla, Jee Hea An, Mihir Bellare, and Chanathip Namprempre. "From Identification to Signatures via the Fiat-Shamir Transform: Minimizing Assumptions for Security and Forward-Security". In: *Advances in Cryptology — EUROCRYPT 2002*. Springer, 2002.

[2] Michel Abdalla and Manuel Barbosa. *Perfect Forward Security of SPAKE2*. Cryptology ePrint Archive, Paper 2019/1194. `https://eprint.iacr.org/2019/1194`. 2019.

[3] Michel Abdalla, Manuel Barbosa, Tatiana Bradley, Stanislaw Jarecki, Jonathan Katz, and Jiayu Xu. *Universally Composable Relaxed Password Authenticated Key Exchange*. Cryptology ePrint Archive, Paper 2020/320. `https://eprint.iacr.org/2020/320`. 2020.

[4] Michel Abdalla, Mihir Bellare, and Gregory Neven. *Robust Encryption*. Cryptology ePrint Archive, Report 2008/440. `https://ia.cr/2008/440`. 2008.

[5] Michel Abdalla, Jens-Matthias Bohli, Maria Isabel Gonzalez Vasco, and Rainer Steinwandt. "(Password) authenticated key establishment: From 2-party to group". In: *Theory of Cryptography: 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007. Proceedings 4*. Springer. 2007.

[6] Michel Abdalla, Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. "Password-based group key exchange in a constant number of rounds". In: *Public Key Cryptography-PKC 2006: 9th International Conference on Theory and Practice in Public-Key Cryptography, New York, NY, USA, April 24-26, 2006. Proceedings 9*. Springer. 2006.

[7] Michel Abdalla, Björn Haase, and Julia Hesse. "Security analysis of CPace". In: *Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part IV*. Springer. 2021.

[8] Michel Abdalla and David Pointcheval. "A scalable password-based group key exchange protocol in the standard model". In: *ASIACRYPT*. Vol. 4284. Springer. 2006.

[9] Michel Abdalla and David Pointcheval. "Simple password-based encrypted key exchange protocols". In: *Topics in Cryptology–CT-RSA 2005: The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005. Proceedings*. Springer. 2005.

[10] Ange Albertini, Thai Duong, Shay Gueron, Stefan Kölbl, Atul Luykx, and Sophie Schmieg. "How to Abuse and Fix Authenticated Encryption Without Key Commitment". In: *31st USENIX Security Symposium*. 2020.

[11] Martin R Albrecht, Jean Paul Degabriele, Torben Brandt Hansen, and Kenneth G Paterson. "A surfeit of SSH cipher suites". In: *ACM Conference on Computer and Communications Security (CCS)*. 2016.

[12] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. "The Double Ratchet: Security Notions, Proofs, and Modularization for the Signal Protocol". In: *Advances in Cryptology – EUROCRYPT 2019*. Springer, 2019.

[13] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. "Security Analysis and Improvements for the IETF MLS Standard for Group Messaging". In: *Advances in Cryptology – CRYPTO 2020*. Springer, 2020.

[14] Diego F Aranha, Felipe Rodrigues Novaes, Akira Takahashi, Mehdi Tibouchi, and Yuval Yarom. "LadderLeak: Breaking ECDSA With Less Than One Bit Of Nonce Leakage". In: *ACM SIGSAC Conference on Computer and Communications Security*. ACM Association for Computing Machinery. 2020.

[15] Scott Arciszewski. *XChaCha: eXtended-nonce ChaCha and AEAD_XChaCha20_Poly1305*. Internet-Draft draft-irtf-cfrg-xchacha-03. Work in Progress: https://datatracker.ietf.org/doc/draft-irtf-cfrg-xchacha/03/. Internet Engineering Task Force, Jan. 2020. 18 pp.

[16] Matilda Backendal, Mihir Bellare, Jessica Sorrell, and Jiahao Sun. "The Fiat-Shamir Zoo: Relating the Security of Different Signature Variants". In: *Secure IT Systems - 23rd Nordic Conference, NordSec 2018,Proceedings*. Vol. 11252. LNCS. Springer, 2018.

[17] Shi Bai, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. *CRYSTALS-Dilithium Algorithm Specifications and Supporting Documentation (Version 3.1)*. https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf.

[18] Dirk Balfanz, Alexei Czeskis, Jeff Hodges, J.C. Jones, Michael B. Jones, Akshay Kumar, Angelo Liao, Rolf Lindemann, Emil Lundberg, Vijay Bharadwaj, Arnar Birgisson, Hubert Le Van Gong, Christiaan Brand, Langley Adam, Giridhar Mandyam, Mike West, and Jeffrey Yasskin. *Web authentication: An API for accessing public key credentials level 1 – W3C recommendation*. https://www.w3.org/TR/2019/REC-webauthn-1-20190304/. March 2019.

[19] Fatih Balli, Paul Rösler, and Serge Vaudenay. "Determining the core primitive for optimally secure ratcheting". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2020.

[20] Manuel Barbosa, Alexandra Boldyreva, Shan Chen, and Bogdan Warinschi. *Provable Security Analysis of FIDO2*. Cryptology ePrint Archive, Paper 2020/756. https://eprint.iacr.org/2020/756, accessed on 18.08.2022. 2020.

[21] Manuel Barbosa, Alexandra Boldyreva, Shan Chen, and Bogdan Warinschi. "Provable security analysis of FIDO2". In: *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part III 41*. Springer. 2021.

[22] R. Barnes, B. Beurdouche, R. Robert, J. Millican, E. Omara, and K. Cohn-Gordon. *The Messaging Layer Security (MLS) Protocol*. https://datatracker.ietf.org/doc/html/draft-ietf-mls-protocol-20.

[23] Guy Barwell, Daniel Page, and Martijn Stam. "Rogue Decryption Failures: Reconciling AE Robustness Notions". In: *Proceedings of the 15th IMA International Conference on Cryptography and Coding*. 2015.

[24] Mihir Bellare and Wei Dai. *The Multi-Base Discrete Logarithm Problem: Non-Rewinding Proofs and Improved Reduction Tightness for Identification and Signatures*. Cryptology ePrint Archive, Report 2020/416. https://eprint.iacr.org/2020/416. 2020.

[25] Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. "A concrete security treatment of symmetric encryption". In: *Proceedings 38th Annual Symposium on Foundations of Computer Science*. IEEE. 1997.

[26] Mihir Bellare, Marc Fischlin, Adam O'Neill, and Thomas Ristenpart. "Deterministic encryption: Definitional equivalences and constructions without random oracles". In: *Annual International Cryptology Conference*. Springer. 2008.

[27] Mihir Bellare and Viet Tung Hoang. *Efficient Schemes for Committing Authenticated Encryption*. Cryptology ePrint Archive, Report 2022/268. https://ia.cr/2022/268. 2022.

[28] Mihir Bellare and Viet Tung Hoang. "Efficient schemes for committing authenticated encryption". In: *Advances in Cryptology–EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30–June 3, 2022, Proceedings, Part II*. Springer. 2022.

[29] Mihir Bellare and Anna Lysyanskaya. *Symmetric and Dual PRFs from Standard Assumptions: A Generic Validation of an HMAC Assumption*. Cryptology ePrint Archive, Report 2015/1198. https://ia.cr/2015/1198. 2015.

[30] Mihir Bellare and Chanathip Namprempre. "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2000.

[31] Mihir Bellare, Ruth Ng, and Björn Tackmann. *Nonces are Noticed: AEAD Revisited.* Cryptology ePrint Archive, Report 2019/624. https://ia.cr/2019/624. 2019.

[32] Mihir Bellare and Adriana Palacio. "GQ and Schnorr Identification Schemes: Proofs of Security against Impersonation under Active and Concurrent Attacks". In: *Advances in Cryptology — CRYPTO 2002.* Springer, 2002.

[33] Mihir Bellare, Bertram Poettering, and Douglas Stebila. "From Identification to Signatures, Tightly: A Framework and Generic Transforms". In: *Advances in Cryptology – ASIACRYPT 2016.* Springer, 2016.

[34] Mihir Bellare and Phillip Rogaway. "Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols". In: *Proceedings of the 1st ACM Conference on Computer and Communications Security.* CCS '93. Fairfax, Virginia, USA: ACM, 1993.

[35] Mihir Bellare, Phillip Rogaway, and David Wagner. "The EAX mode of operation". In: *International Workshop on Fast Software Encryption.* 2004.

[36] Mihir Bellare, Asha Camper Singh, Joseph Jaeger, Maya Nyayapati, and Igors Stepanovs. "Ratcheted encryption and key exchange: The security of messaging". In: *Annual International Cryptology Conference.* Springer. 2017.

[37] Florian Bergsma, Benjamin Dowling, Florian Kohlar, Jörg Schwenk, and Douglas Stebila. "Multi-ciphersuite security of the Secure Shell (SSH) protocol". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security.* 2014.

[38] Florian Bergsma, Benjamin Dowling, Florian Kohlar, Jörg Schwenk, and Douglas Stebila. *Multi-ciphersuite security of the Secure Shell (SSH) protocol.* Cryptology ePrint Archive, Report 2013/813. https://eprint.iacr.org/2013/813. 2013.

[39] Daniel J. Bernstein. "Curve25519: New Diffie-Hellman Speed Records". In: *Public Key Cryptography - PKC 2006.* Springer, 2006.

[40] Daniel J. Bernstein. "Multi-user Schnorr security, revisited". In: *IACR Cryptology ePrint Archive* 2015 (2015).

[41] Daniel J. Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. "Twisted Edwards Curves". In: *Progress in Cryptology – AFRICACRYPT 2008.* Springer, 2008.

[42] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. *Ed25519: high-speed high-security signatures.* https://ed25519.cr.yp.to/, (Accessed March 09, 2020).

[43] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. "High-Speed High-Security Signatures". In: *CHES.* Vol. 6917. Lecture Notes in Computer Science. Springer, 2011.

[44] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. "High-speed high-security signatures". In: *J. Cryptographic Engineering* 2.2 (2012).

[45] Daniel J Bernstein, Simon Josefsson, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. "EdDSA for more curves". In: *Cryptology ePrint Archive* (2015).

[46] Daniel J. Bernstein and Edoardo Persichetti. *Towards KEM Unification.* Cryptology ePrint Archive, Paper 2018/526. https://eprint.iacr.org/2018/526.

[47] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. "Verified models and reference implementations for the TLS 1.3 standard candidate". In: *2017 IEEE Symposium on Security and Privacy (SP).* IEEE. 2017.

[48] Karthikeyan Bhargavan, Christina Brzuska, Cédric Fournet, Matthew Green, Markulf Kohlweiss, and Santiago Zanella-Béguelin. "Downgrade resilience in key-exchange protocols". In: *2016 IEEE Symposium on Security and Privacy (SP).* IEEE. 2016.

[49] Ritam Bhaumik and Mridul Nandi. "Improved security for OCB3". In: *International Conference on the Theory and Application of Cryptology and Information Security.* 2017.

[50] Alexander Bienstock, Jaiden Fairoze, Sanjam Garg, Pratyay Mukherjee, and Srinivasan Raghuraman. *A More Complete Analysis of the Signal Double Ratchet Algorithm.* Cryptology ePrint Archive, Paper 2022/355. https://eprint.iacr.org/2022/355.

[51] Nina Bindel, Udyani Herath, Matthew McKague, and Douglas Stebila. “Transitioning to a quantum-resistant public key infrastructure”. In: *Post-Quantum Cryptography: 8th International Workshop, PQCrypto 2017, Utrecht, The Netherlands, June 26-28, 2017, Proceedings 8*. Springer. 2017.

[52] Simon Blake-Wilson and Alfred Menezes. “Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol”. In: *Public Key Cryptography*. Springer, 1999.

[53] Josh Blum, Simon Booth, Brian Chen, Oded Gal, Maxwell Krohn, Julia Len, Karan Lyons, Antonio Marcedone, Mike Maxim, Merry Ember Mou, Armin Namavari, Jack O’Connor, Surya Rien, Miles Steele, Matthew Green, Lea Kissner, and Alex Stamos. *Zoom end-to-end encryption whitepaper*. https://github.com/zoom/zoom-e2e-whitepaper Version 4.0 (Released on 18.11.2022).

[54] Jens-Matthias Bohli, Stefan Röhrich, and Rainer Steinwandt. “Key substitution attacks revisited: Taking into account malicious signers”. In: *International Journal of Information Security* 5.1 (2006).

[55] Alexandra Boldyreva, Jean Paul Degabriele, Kenneth G Paterson, and Martijn Stam. “Security of symmetric encryption in the presence of ciphertext fragmentation”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 2012.

[56] Dan Boneh and Xavier Boyen. “Short Signatures Without Random Oracles”. In: *Advances in Cryptology - EUROCRYPT 2004*. Springer, 2004.

[57] Joppe W. Bos, Craig Costello, Patrick Longa, and Michael Naehrig. “Selecting Elliptic Curves for Cryptography: An Efficiency and Security Analysis”. In: *Journal of Cryptographic Engineering* 6.4 (Nov. 2016).

[58] Colin Boyd, Anish Mathuria, and Douglas Stebila. *Protocols for Authentication and Key Establishment, Second Edition*. Information Security and Cryptography. Springer, 2020.

[59] John Bradley, Jeff Hodges, Michael B. Jones, Akshay Kumar, Rolf Lindemann, Johan Verrept, Chad Armstrong, Konstantinos Georgantas, Fabian Kaczmarczyck, Nina Satragno, and Nuno Sung. *Client to Authenticator Protocol (CTAP) – Proposed Standard, June 15, 2021*. https://fidoalliance.org/specs/fido-v2.1-ps-20210615/fido-client-to-authenticator-protocol-v2.1-ps-20210615.html. 2021.

[60] Christiaan Brand, Alexei Czeskis, Ehrensvärd Jakob, Michael B. Jones, Akshay Kumar, Rolf Lindemann, Adam Powers, and Johan Verrept. *Client to Authenticator Protocol (CTAP) – Proposed Standard, January 30, 2019*. https://fidoalliance.org/specs/fido-v2.0-ps-20190130/fido-client-to-authenticator-protocol-v2.0-ps-20190130.html. 2019.

[61] Jacqueline Brendel, Cas Cremers, Dennis Jackson, and Mang Zhao. “The provable security of Ed25519: theory and practice”. In: *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2021.

[62] Jacqueline Brendel, Cas Cremers, Dennis Jackson, and Mang Zhao. “The provable security of Ed25519: theory and practice”. In: *2021 IEEE Symposium on Security and Privacy (S&P)*. IEEE. 2021.

[63] Jacqueline Brendel, Rune Fiedler, Felix Günther, Christian Janson, and Douglas Stebila. *Post-quantum Asynchronous Deniable Key Exchange and the Signal Handshake*. Cryptology ePrint Archive, Report 2021/769. https://ia.cr/2021/769. 2021.

[64] Jacqueline Brendel, Marc Fischlin, Felix Günther, and Christian Janson. “PRF-ODH: Relations, instantiations, and impossibility results”. In: *Advances in Cryptology–CRYPTO 2017: 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part III 37*. Springer. 2017.

[65] Jacqueline Brendel, Marc Fischlin, Felix Günther, Christian Janson, and Douglas Stebila. “Towards post-quantum security for Signal’s X3DH handshake”. In: *International Conference on Selected Areas in Cryptography*. Springer. 2020.

[66] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. “Group Diffie-Hellman key exchange secure against dictionary attacks”. In: *Advances in Cryptology—ASIACRYPT 2002: 8th International Conference on the Theory and Application of Cryptology and Information Security Queenstown, New Zealand, December 1–5, 2002 Proceedings 8*. Springer. 2002.

[67] John Chan and Phillip Rogaway. *Anonymous AE.* Cryptology ePrint Archive, Report 2019/1033. https://ia.cr/2019/1033. 2019.

[68] Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hülsing, Joost Rijneveld, John M Schanck, Peter Schwabe, William Whyte, and Zhenfei Zhang. "Algorithm specifications and supporting documentation". In: *Brown University and Onboard security company, Wilmington USA* (2019).

[69] Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. *NISTIR 8105 Report on Post-Quantum Cryptography.* Tech. rep. National Institute for Standards and Technology (NIST), 2016.

[70] Lily Chen, Dustin Moody, Andrew Regenscheid, and Karen Randall. *Recommendations for discrete logarithm-based cryptography: Elliptic curve domain parameters.* Tech. rep. Federal Information Processing Standard (FIPS) 186-5 (Draft). National Institute of Standards and Technology, 2019.

[71] *Handbook of Elliptic and Hyperelliptic Curve Cryptography.* Chapman and Hall/CRC, 2005.

[72] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. "A Formal Security Analysis of the Signal Messaging Protocol". In: *Journal of Cryptology* 33.4 (2020).

[73] Katriel Cohn-Gordon, Cas Cremers, and Luke Garratt. *Post-Compromise Security.* Cryptology ePrint Archive, Paper 2016/221. https://eprint.iacr.org/2016/221. 2016.

[74] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. "On Ends-to-Ends Encryption: Asynchronous group messaging with strong security guarantees". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security.* 2018.

[75] Katriel Cohn-Gordon, Cas J. F. Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. "A Formal Security Analysis of the Signal Messaging Protocol". In: *EuroS&P.* IEEE, 2017.

[76] Ronald Cramer and Victor Shoup. "Design and Analysis of Practical Public-Key Encryption Schemes Secure against Adaptive Chosen Ciphertext Attack". In: *SIAM Journal on Computing* 33.1 (2003). eprint: https://doi.org/10.1137/S0097539702403773.

[77] Cas Cremers and Dennis Jackson. "Prime, order please! Revisiting small subgroup and invalid curve attacks on protocols using Diffie-Hellman". In: *2019 IEEE 32nd Computer Security Foundations Symposium (CSF).* IEEE. 2019.

[78] Quynh Dang. *Recommendation for Applications Using Approved Hash Algorithms.* en. https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=911479 (Accessed Feb 2023). Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, 2012.

[79] Frank Denis. *The Sodium cryptography library.* https://doc.libsodium.org/, (Accessed Jan 2023).

[80] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk. "Deniable Authentication and Key Exchange". In: *ACM CCS.* ACM, 2006.

[81] Samuel Dobson and Steven D. Galbraith. *Post-Quantum Signal Key Agreement with SIDH.* Cryptology ePrint Archive. https://ia.cr/2021/1187. 2021.

[82] Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, and Joanne Woodage. *Fast Message Franking: From Invisible Salamanders to Encryptment.* Cryptology ePrint Archive, Report 2019/016. https://ia.cr/2019/016. 2019.

[83] Yevgeniy Dodis, Daniel Jost, Balachandar Kesavan, and Antonio Marcedone. "End-to-End Encrypted Zoom Meetings: Proving Security and Strengthening Liveness". In: *Advances in Cryptology–EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, Proceedings.* Springer-Verlag, 2023.

[84] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. "A cryptographic analysis of the TLS 1.3 handshake protocol candidates". In: *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security.* 2015.

[85] F. Betül Durak and Serge Vaudenay. "Bidirectional Asynchronous Ratcheted Key Agreement with Linear Complexity". In: *Advances in Information and Computer Security.* Springer, 2019.

[86] Peter Eckersley and Jesse Burns. “The (Decentralized) SSL Observatory”. In: *20th USENIX Security Symposium (USENIX Security 11)*. https://www.usenix.org/conference/usenix-security-11/decentralized-ssl-observatory. San Francisco, CA: USENIX Association, Aug. 2011.

[87] William F Ehrsam, Carl HW Meyer, John L Smith, and Walter L Tuchman. *Message verification and transmission error detection by block chaining*. US Patent 4,074,066. 1978.

[88] Computer Aided Cryptography Engineering. *NaCl: Networking and Cryptography library*. https://nacl.cr.yp.to/sign.html (Accessed May 29th, 2020). 2019.

[89] Pooya Farshim, Claudio Orlandi, and Răzvan Roşie. *Security of Symmetric Primitives under Incorrect Usage of Keys*. Cryptology ePrint Archive, Report 2017/288. https://ia.cr/2017/288. 2017.

[90] Amos Fiat and Adi Shamir. “How To Prove Yourself: Practical Solutions to Identification and Signature Problems”. In: *Advances in Cryptology — CRYPTO' 86*. Springer, 1987.

[91] Pierre-Alain Fouque, Gwenaëlle Martinet, Frédéric Valette, and Sébastien Zimmer. “On the Security of the CCM Encryption Mode and of a Slight Variant”. In: *International Conference on Applied Cryptography and Network Security*. 2008.

[92] S. Galbraith, J. Malone-Lee, and N.P. Smart. “Public key signatures in the multi-user setting”. In: *Information Processing Letters* 83.5 (2002).

[93] Rosario Gennaro, Shai Halevi, and Tal Rabin. “Secure Hash-and-Sign Signatures Without the Random Oracle”. In: *Advances in Cryptology — EUROCRYPT '99*. Springer, 1999.

[94] Shafi Goldwasser, Silvio Micali, and Ronald L Rivest. “A digital signature scheme secure against adaptive chosen-message attacks”. In: *SIAM Journal on Computing* 17.2 (1988).

[95] Justin Goshi and Richard E Ladner. “Algorithms for dynamic multicast key distribution trees”. In: *Proceedings of the twenty-second annual symposium on Principles of distributed computing*. 2003.

[96] Jack Grigg, George Tankersley, Henry de Valence, Isis Lovecruft, and Filippo Valsorda. *The Ristretto255 Group*. https://tools.ietf.org/html/draft-irtf-cfrg-ristretto255-00 (Accessed May 29th, 2020).

[97] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. *Message Franking via Committing Authenticated Encryption*. Cryptology ePrint Archive, Report 2017/664. https://ia.cr/2017/664. 2017.

[98] Shay Gueron and Yehuda Lindell. “GCM-SIV: full nonce misuse-resistant authenticated encryption at under one cycle per byte”. In: *ACM Conference on Computer and Communications Security (CCS)*. 2015.

[99] Louis C. Guillou and Jean-Jacques Quisquater. “A Practical Zero-Knowledge Protocol Fitted to Security Microprocessor Minimizing Both Transmission and Memory”. In: *Advances in Cryptology — EUROCRYPT '88*. Springer, 1988.

[100] Felix Günther and Bertram Poettering. “Linkable Message Tagging: Solving the Key Distribution Problem of Signature Schemes”. In: *Information Security and Privacy*. Springer, 2015.

[101] Björn Haase and Benoît Labrique. “AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2019.2 (2019). https://tches.iacr.org/index.php/TCHES/article/view/7384.

[102] Darrel Hankerson and Alfred Menezes. “Elliptic Curve Cryptography”. In: *Encyclopedia of Cryptography and Security*. Springer, 2011.

[103] Lucjan Hanzlik, Julian Loss, and Benedikt Wagner. *Token meets Wallet: Formalizing Privacy and Revocation for FIDO2*. Cryptology ePrint Archive, Report 2022/084. https://ia.cr/2022/084. 2022.

[104] H. Harney and C. Muckenhirn. *Group Key Management Protocol (GKMP) Architecture*. RFC 2094 (Experimental). Internet Engineering Task Force, July 1997.

[105] H. Harney and C. Muckenhirn. *Group Key Management Protocol (GKMP) Specification*. RFC 2093 (Experimental). Internet Engineering Task Force, July 1997.

[106] Keitaro Hashimoto, Shuichi Katsumata, Kris Kwiatkowski, and Thomas Prest. “An Efficient and Generic Construction for Signal’s Handshake (X3DH): Post-Quantum, State Leakage Secure, and Deniable”. In: *Public-Key Cryptography - PKC 2021*. Springer, 2021.

[107] Jeff Hodges, J.C. Jones, Michael B. Jones, Akshay Kumar, Emil Lundberg, John Bradley, Christiaan Brand, Langley Adam, Giridhar Mandyam, Nina Satragno, Nick Steele, Jiewen Tan, Shane Weeden, Mike West, and Jeffrey Yasskin. *Web authentication: An API for accessing public key credentials level 2 – W3C recommendation.* `https://www.w3.org/TR/2021/REC-webauthn-2-20210408/`. April 2021.

[108] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. *ZCash Protocol Specification Version 2020.1.14.* `https://github.com/zcash/zips/blob/master/protocol/protocol.pdf` (Accessed September 15, 2020). 2020.

[109] *How many people use Zoom?* `https://www.zippia.com/advice/zoom-meeting-statistics/`, (Accessed Feb 2023). 2022.

[110] IETF Mail Archive. *[Cfrg] Does the Curve25519 Montgomery Ladder Always Work?* `https://mailarchive.ietf.org/arch/msg/cfrg/pt2bt3fGQbNF8qdEcorp-rJSJrc/`. (Accessed: June 04,2020).

[111] IETF Mail Archive. *Re: [Cfrg] EC signature: next steps.* `https://mailarchive.ietf.org/arch/msg/cfrg/TOWH1DSzB-PfDGK8qEXtF3iC6Vc/`. (Accessed: June 04, 2020).

[112] IETF Mail Archive. *Re: [Cfrg] EC signature: next steps.* `https://mailarchive.ietf.org/arch/msg/cfrg/af170b6OrLyNZUHBMOPWxcDrVRI/`. (Accessed: June 04, 2020).

[113] Takanori Isobe and Ryoma Ito. *Security Analysis of End-to-End Encryption for Zoom Meetings.* Cryptology ePrint Archive, Paper 2021/486. `https://eprint.iacr.org/2021/486`. 2021.

[114] Takanori Isobe and Ryoma Ito. “Security analysis of end-to-end encryption for Zoom meetings”. In: *IEEE Access* 9 (2021).

[115] Tetsu Iwata, Keisuke Ohashi, and Kazuhiko Minematsu. “Breaking and repairing GCM security proofs”. In: *Annual Cryptology Conference*. 2012.

[116] Tetsu Iwata and Yannick Seurin. “Reconsidering the Security Bound of AES-GCM-SIV”. In: *IACR Transactions on Symmetric Cryptology* (2017).

[117] Dennis Jackson, Cas Cremers, Katriel Cohn-Gordon, and Ralf Sasse. “Seems Legit: Automated Analysis of Subtle Attacks on Protocols that Use Signatures”. In: *ACM Conference on Computer and Communications Security*. ACM, 2019.

[118] Joseph Jaeger and Igors Stepanovs. “Optimal Channel Security Against Fine-Grained State Compromise: The Safety of Messaging”. In: *Advances in Cryptology – CRYPTO 2018*. Springer, 2018.

[119] *Jedisct1/Libsodium.* `https://github.com/jedisct1/libsodium` (Accessed May 31th, 2020).

[120] Jakob Jonsson. “On the security of CTR+ CBC-MAC”. In: *International Workshop on Selected Areas in Cryptography*. 2002.

[121] S. Josefsson and I. Liusvaara. *RFC 8032: Edwards-Curve Digital Signature Algorithm (EdDSA).* `https://tools.ietf.org/html/rfc8032` (Accessed March 9th, 2020). 2017.

[122] Daniel Jost, Ueli Maurer, and Marta Mularczyk. “Efficient Ratcheting: Almost-Optimal Guarantees for Secure Messaging”. In: *Advances in Cryptology – EUROCRYPT 2019*. Springer, 2019.

[123] B. Kaliski, J. Jonsson, and A. Rusch. *PKCS #1: RSA Cryptography Specifications Version 2.2.* RFC 8017 (Informational). Internet Engineering Task Force, Nov. 2016.

[124] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography (Second Edition).* Chapman & Hall CRC, 2014.

[125] Eike Kiltz, Daniel Masny, and Jiaxin Pan. “Optimal Security Proofs for Signatures from Identification Schemes”. In: *Advances in Cryptology – CRYPTO 2016*. Springer, 2016.

[126] Eike Kiltz, Daniel Masny, and Jiaxin Pan. *Optimal Security Proofs for Signatures from Identification Schemes.* Cryptology ePrint Archive, Report 2016/191. `https://eprint.iacr.org/2016/191`. 2016.

[127] Yongdae Kim, Adrian Perrig, and Gene Tsudik. "Simple and fault-tolerant key agreement for dynamic collaborative groups". In: *Proceedings of the 7th ACM Conference on Computer and Communications Security*. 2000.

[128] Karen Klein, Guillermo Pascual-Perez, Michael Walter, Chethan Kamath, Margarita Capretto, Miguel Cueto, Ilia Markov, Michelle Yeo, Joël Alwen, and Krzysztof Pietrzak. "Keep the dirt: tainted treekem, adaptively and actively secure continuous group key agreement". In: *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2021.

[129] Evan Klitzke. *Bitcoin Transaction Malleability*. https://eklitzke.org/bitcoin-transaction-malleability, (Accessed June 1, 2020). 2017.

[130] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. "Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach". In: *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2017.

[131] Markulf Kohlweiss, Ueli Maurer, Cristina Onete, Björn Tackmann, and Daniele Venturi. "(De-)constructing TLS 1.3". In: *International Conference on Cryptology in India*. Springer. 2015.

[132] Ted Krovetz and Phillip Rogaway. "The software performance of authenticated-encryption modes". In: *International Workshop on Fast Software Encryption*. 2011.

[133] Peter S Kruus. *A survey of multicast security issues and architectures*. Tech. rep. NAVAL RESEARCH LAB WASHINGTON DC, 1998.

[134] Brian LaMacchia, Kristin Lauter, and Anton Mityagin. "Stronger Security of Authenticated Key Exchange". In: *Provable Security*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.

[135] Julia Len, Paul Grubbs, and Thomas Ristenpart. "Partitioning Oracle Attacks". In: *30th USENIX Security Symposium*. 2021.

[136] Zenghui Liu, Yingxu Lai, Xubo Ren, and Shupo Bu. "An efficient LKH tree balancing algorithm for group key management". In: *2012 International Conference on Control Engineering and Communication Technology*. IEEE. 2012.

[137] Haibin Lu. "A novel high-order tree for secure multicast key management". In: *IEEE Transactions on Computers* 54.2 (2005).

[138] Moxie Marlinspike and Trevor Perrin. *The X3DH Key Agreement Protocol*. https://signal.org/docs/specifications/x3dh/. November 2016.

[139] David A McGrew and John Viega. "The security and performance of the Galois/Counter Mode (GCM) of operation". In: *International Conference on Cryptology in India*. 2004.

[140] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 2018.

[141] Kazuhiko Minematsu, Stefan Lucks, and Tetsu Iwata. "Improved authenticity bound of EAX, and refinements". In: *International Conference on Provable Security*. 2013.

[142] Modern Crypto Mail Archive. *[Curves] Ed25519 "Clamping" and Its Effect on Hierarchical Key Derivation*. https://moderncrypto.org/mail-archive/curves/2017/000874.html. (Accessed: June 04, 2020).

[143] S. Moonesamy. *RFC 7479: Using Ed25519 in SSHFP Resource Records*. https://tools.ietf.org/html/rfc7479 (Accessed May 29th, 2020). 2015.

[144] Matthew J Moyer, Josyula R Rao, and Pankaj Rohatgi. "A survey of security issues in multicast communications". In: *IEEE network* 13.6 (1999).

[145] National Institute of Standards and Technology. *Digital Signature Standard (DSS)*. https://csrc.nist.gov/publications/detail/fips/186/5/draft (Accessed May 31th, 2020). 2019.

[146] National Institute of Standards and Technology. "The Digital Signature Standard". In: *Communications of the ACM* 35.7 (1992).

[147] National Institute of Standards and Technology. *Request for Comments on FIPS 186-5 and SP 800-186*. https://www.federalregister.gov/documents/2019/10/31/2019-23742/request-for-comments-on-fips-186-5-and-sp-800-186, (Accessed June 1, 2020). 2019.

[148] Tatsuaki Okamoto. “Provably Secure and Practical Identification Schemes and Corresponding Signature Schemes”. In: *Advances in Cryptology — CRYPTO’ 92*. Springer, 1993.

[149] OpenBSD. *ed25519.c*. https://anongit.mindrot.org/openssh.git/tree/ed25519.c, (Accessed June 1, 2020). 2013.

[150] Adrian Perrig. “Efficient collaborative key management protocols for secure autonomous group communication”. In: *International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC’99)*. 1999.

[151] Trevor Perrin. *The XEdDSA and VXEdDSA Signature Schemes*. https://signal.org/docs/specifications/xeddsa/ (Accessed March 9th, 2020). 2016.

[152] Trevor Perrin and Moxie Marlinspike. *The Double Ratchet Algorithm*. https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf. November 2016.

[153] Jeroen Pijnenburg and Bertram Poettering. “On Secure Ratcheting with Immediate Decryption”. In: *Advances in Cryptology – ASIACRYPT 2022*. Cham: Springer Nature Switzerland, 2022.

[154] Bertram Poettering and Paul Rösler. “Towards Bidirectional Ratcheted Key Exchange”. In: *Advances in Cryptology – CRYPTO 2018*. Springer, 2018.

[155] Bertram Poettering, Paul Rösler, Jörg Schwenk, and Douglas Stebila. *SoK: Game-based Security Models for Group Key Exchange*. Cryptology ePrint Archive, Paper 2021/305. https://eprint.iacr.org/2021/305. 2021.

[156] David Pointcheval and Jacques Stern. “Security Arguments for Digital Signatures and Blind Signatures”. In: *Journal of Cryptology* 13.3 (June 2000).

[157] David Pointcheval and Jacques Stern. “Security Proofs for Signature Schemes”. In: *Advances in Cryptology — EUROCRYPT ’96*. Springer, 1996.

[158] Thomas Pornin and Julien P. Stern. “Digital Signatures Do Not Guarantee Exclusive Ownership”. In: *Applied Cryptography and Network Security*. Springer, 2005.

[159] Gordon Procter. “A Security Analysis of the Composition of ChaCha20 and Poly1305”. In: *Cryptology ePrint Archive, Paper 2014/613*. https://eprint.iacr.org/2014/613. 2014.

[160] Sandro Rafaeli and David Hutchison. “A Survey of Key Management for Secure Group Communication”. In: *ACM Comput. Surv.* 35.3 (2003). https://doi.org/10.1145/937503.937506.

[161] E. Rescorla. *RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3*. https://tools.ietf.org/html/rfc8446 (Accessed May 29th, 2020). 2018.

[162] *Ristretto - The Ristretto Group*. https://ristretto.group/ristretto.html (Accessed May 29th, 2020).

[163] Phillip Rogaway. “Authenticated-Encryption with Associated-Data”. In: *ACM Conference on Computer and Communications Security (CCS)*. CCS ’02. 2002.

[164] Phillip Rogaway and John Steinberger. “Security/Efficiency Tradeoffs for Permutation-Based Hashing”. In: *EUROCRYPT 2008*. 2008.

[165] Jim Schaad. *CBOR Object Signing and Encryption (COSE)*. RFC 8152. July 2017.

[166] C.-P. Schnorr. “Efficient Signature Generation by Smart Cards”. In: *J. Cryptol.* 4.3 (Jan. 1991).

[167] Claus-Peter Schnorr. “Efficient identification and signatures for smart cards”. In: *Conference on the Theory and Application of Cryptology*. Springer, 1989.

[168] Alon Shakevsky, Eyal Ronen, and Avishai Wool. “Trust Dies in Darkness: Shedding Light on Samsung’s TrustZone Keymaster Design”. In: *Cryptology ePrint Archive, Paper 2022/208*. https://eprint.iacr.org/2022/208. 2022.

[169] Alan T Sherman and David A McGrew. “Key establishment in large dynamic groups using one-way function trees”. In: *IEEE transactions on Software Engineering* 29.5 (2003).

[170] NP Smart and A Menezes. “Security of signature schemes in a multi-user setting”. In: *Designs, Codes and Cryptography* 33 (Aug. 2004).

[171] Jeff Stapleton. “PKI Under Attack”. In: *ISSA* 11.3 (2013). https://cdn.ymaws.com/www.members.issa.org/resource/resmgr/JournalPDFs/PKI_Under_Attack_ISSA0313.pdf.

[172] Nik Unger and Ian Goldberg. “Deniable Key Exchanges for Secure Messaging”. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. CCS ’15. ACM, 2015.

[173] Nik Unger and Ian Goldberg. “Improved strongly deniable authenticated key exchanges for secure messaging”. In: *Proceedings on Privacy Enhancing Technologies* 2018.1 (2018).

[174] John R Vacca. *Public key infrastructure: building trusted applications and Web services*. Auerbach Publications, 2004.

[175] Nihal Vatandas, Rosario Gennaro, Bertrand Ithurburn, and Hugo Krawczyk. “On the Cryptographic Deniability of the Signal Protocol”. In: *Applied Cryptography and Network Security*. Springer, 2020.

[176] Chung Kei Wong, Mohamed Gouda, and Simon S Lam. “Secure group communications using key graphs”. In: *IEEE/ACM transactions on networking* 8.1 (2000).

[177] Lihao Xu and Cheng Huang. “Computation-efficient multicast key distribution”. In: *IEEE Transactions on Parallel and Distributed Systems* 19.5 (2008).