Saarland University

Department of Computer Science

New Approaches to Privacy Preserving Signatures

Dissertation

zur Erlangung des Grades
des Doktors der Ingenieurwissenschaften (Dr.-Ing.)
der Fakultät für Mathematik und Informatik
der Universität des Saarlandes

vorgelegt von
Jonas Schneider-Bensch

Saarbrücken, 2021

Details des Kolloquiums

Tag des Kolloquiums:	26.09.2022
Dekan der Fakultät:	Univ.-Prof. Dr. Jürgen Steimle
Prüfungsausschuss:	
Vorsitz:	Prof. Dr. Raimund Seidel
Berichterstatter:	Prof. Dr. Michael Backes
	Prof. Dr. Aniket Kate
Akademischer Mitarbeiter:	Dr. Dipayan Das

Abstract

In this thesis we advance the theory and practice of privacy preserving digital signatures. Privacy preserving signatures such as group and ring signatures enable signers to hide in groups of potential signers. We design a cryptographic primitive called signatures with flexible public keys, which allows for modular construction of privacy preserving signatures. Its core is an equivalence relation between verification keys, such that key representatives can be transformed in their class to obscures their origin. The resulting constructions are more efficient than the state of the art, under the same or weaker assumptions. We show an extension of the security model of fully dynamic group signatures, which are those where members may join and leave the group over time. Our contribution here, which is facilitated by the new primitive, is the treatment of membership status as potentially sensitive information. In the theory of ring signatures, we show a construction of ring signatures which is the first in the literature with logarithmic signature size in the size of the ring without any trusted setup or reliance on non-standard assumptions. We show how to extend our techniques to the derived setting of linkable ring signatures, where different signatures of the same origin may be publicly linked. Here, we further revisit the notion of linkable anonymity, offering a significant strengthening compared to previous definitions.

Zusammenfassung

Diese Arbeit treibt die Theorie und Praxis der privatsphärewahrenden digitalen Signaturen voran. Privatsphärewahrende Signaturen, wie Gruppen- oder Ringsignaturen erlauben es Zeichnern sich in einer Gruppe potenzieller Zeichner zu verstecken. Wir entwerfen mit Signatures with Flexible Public Keys einen kryptografischen Baustein zur modularen Konstruktion von privatsphärewahrenden Signaturen. Dessen Kern ist eine Äquivalenzrelation zwischen den Schlüsseln, sodass ein Schlüsselvertreter in seiner Klasse bewegt werden kann, um seinen Ursprung zu verschleiern. Darauf aufbauende Konstruktionen sind effizienter als der Stand der Technik, unter gleichen oder schwächeren Annahmen. Wir erweitern das Sicherheitsmodell vollständig dynamischer Gruppensignaturen, die es Mitgliedern erlauben der Gruppe beizutreten oder sie zu verlassen: Durch das neue Primitiv, wird die Behandlung der Mitgliedschaft als potenziell sensibel ermöglicht. In der Theorie der Ringsignaturen geben wir die erste Konstruktion, welche über eine logarithmische Signaturgröße verfügt, ohne auf eine Vorkonfiguration oder unübliche Annahmen vertrauen zu müssen. Wir übertragen unsere Ergebnisse auf das Feld der verknüpfbaren Ringsignaturen, die eine öffentliche Verknüpfung von zeichnergleichen Signaturen ermöglichen. Unsere Neubetrachtung des Begriffs der verknüpfbaren Anonymität führt zu einer signifikanten Stärkung im Vergleich zu früheren Definitionen.

Acknowledgments

First, I want to thank my advisor, Michael Backes, for his invaluable guidance over the last years. You were very patient with me, and you saw potential where my outlook was grim.

I also want to thank Aniket Kate for agreeing to review this thesis. Long before I started my PhD, Aniket gave me the chance to attend my first academic conference, which I will never forget, thus introducing me to the world of research.

In my view, research can only be fruitful in the interaction and exchange of knowledge and perspectives between people. So, naturally, I want to express heart-felt gratitude to all of my co-authors over the years, for providing many unique insights, into our projects and into research itself, and for helping me grow into a better researcher than before I had met them. I especially want to thank Lucjan Hanzlik for motivating me to work on privacy preserving signatures with him and for offering additional guidance at many junctions. I want to thank you for providing a pragmatic counterweight when my self-doubt weighed heavy.

I want to thank my colleagues at CISPA, where I have met so many wonderful and kind people from whom I have learned so much. Especially, I would like to thank all of my office mates, past and present. Most recently I was fortunate enough to share an office with Ahmed, Ivan, and Min, who managed to keep the friendly spirit of our little office alive, even virtually, in this time of the global pandemic. I wish all of you the very best!

Thank you to Tim Ruffing and Dominik Schillo for agreeing to the arduous task of proofreading parts of this thesis. I am lucky to count you among my friends, all the more afterwards! 😉

Last, but not least, I want to thank all my friends and family for supporting me in my studies by not letting me lose my head in numerous cases. And for lovingly re-attaching it in countless other cases. My deepest gratitude goes to my wife, Nina. Thank you for being there for me, no matter what!

Contents

1 Introduction

In this chapter:

The field of information security recognizes the triad of confidentiality, integrity and availability as the summation of our goals when it comes to securing communications. This means, roughly, that messages should arrive only at those people for whom they were meant, and nowhere else, that they should not be illegitimately tampered with in transit, and that the systems which facilitate our communication themselves should be safe from malicious disruption.

In our modern information society we have developed design principles for information systems that harden them against denial of service and other types of disruptive attacks. We have notions of public and private key encryption, which give us fine-grained control over the confidentiality of our communications. Addressing issues of communication integrity, the solutions also come from cryptography: Digital signature schemes (and message authentication codes, their counterparts in the realm of symmetric cryptography) are the fundamental building blocks that ensure that the words others hear us say are really the words we spoke and that it was really us that spoke them.

Informally, a digital signature on some message represents a publicly verifiable claim that the owner of the signing key has at some point seen and signed this message. Hidden in this description are the two crucial aspects of the kind of statements digital signatures allow us to make:

- First, the digital signature is tied to the owner of a secret signing key that has an associated public identity represented by their public verification key. If we know of this association, the signature serves as a non-repudiable proof that exactly this person has seen the message. A different person without access to the signing key can never show us someone else's valid signature on a message that the signer has not actually created.[1]

- Second, the signature is valid only for the exact message the signer has created it for. This prevents anyone from being able to make the signer attest messages they did not explicitly accept.

Thus, digital signatures are the basis of the public key infrastructure that allows us to communicate with people we have never met, while retaining some modicum of assurance that they really are who they claim to be. Any form of online commerce like we know it today would be impossible without digital signatures. They are also at the heart of modern software updates, where software vendors provide signatures on the patches they hand out, allowing users to verify that any update came from the, presumably trustworthy, vendor and was not modified by a third party while in transit.

1.1 Privacy Preserving Signatures

Privacy preserving signature schemes are digital signature schemes where a signature can be made on behalf of a group of more than one signer, and where the signature does not reveal which member of the group in particular was the creator of any one signature. At the same time, it should remain infeasible for non-members to forge signatures on behalf of the group. This is a relaxation of the non-repudiation property of digital signature schemes described above, but the circle of potential signers is still limited and can be made up of meaningful group of people.

This has proved to be very useful in the construction of privacy preserving information systems of many types, where we know that a certain group of people share some property and one of them should give a signature, but we need not or must not reveal who exactly the actual signer is.

This thesis contains new results for two types of privacy preserving signatures, namely group signatures and ring signatures.

1.1.1 Group Signatures

The concept of group signatures was introduced by Chaum and van Heyst in [Cv91]. It allows a group manager to delegate signing rights to multiple signers. The group members may create publicly verifiable signatures on behalf of the entire group, such that the

[1]Unless they break the cryptographic hardness assumptions that are at the heart of the security proofs of any such scheme.

signature does not reveal the identity of the actual signer beyond their membership in the group. A designated opening authority has the ability to verifiably reveal the actual signer of a particular signature, in case of abuse. Ideal applications of group signatures are, for instance, business processes, where responsibility for certain actions should be shared among the members of one level of management by creating a signature, but accountability is preserved via the possibility of after-the-fact opening of a signature through a supervisory board.

1.1.2 Ring Signatures

Ring signatures, introduced by Rivest, Shamir and Tauman-Kalai [RST01] allow a signer to hide in a *ring* of potential signers that the signer has chosen themselves. More specifically, the signing algorithm of a ring signature scheme takes as additional input a list of verification keys R and outputs a signature. Such a signature can be verified given the ring R. The feature of interest of ring signatures is that given such a signature, no one, not even an insider in possession of all the secret keys corresponding to the verification keys in the ring, can tell which key was used to compute this signature. The original motivation for ring signatures was whistle-blowing, where the leaking party can hide their identity and at the same time convince outsiders that the leaked information is genuine (by using a ring composed only of people with access to this information). In terms of security two properties are required of ring signatures: unforgeability and anonymity. The first property requires that an efficient adversary should not be able to forge a signature on behalf of an honest ring of signers. Anonymity requires that signatures do not give away by which member they were created. This can be cast as an experiment in which the adversary has to guess which one out of two ring members created a signature.

1.2 An Overview of Our Results

Both primitives present challenges in terms of their security modeling and the unique requirements of their applications. For instance, group signatures can be made the basis of more complex schemes such as *direct anonymous attestation (DAA)* and the balance between privacy of group members and powers entrusted to privileged authorities must be carefully considered. One application of ring signatures is in the implementation of *confidential transaction protocols* in the context of cryptocurrencies. In this context, practical efficiency is paramount for the efficiency of signed transactions, which are the basic items that can be considered with respect to a cryptocurrency. At the same time, since the unforgeability of a ring signature in this context directly protects the integrity of digital financial assets in a decentralized system, there is no room in the security model for trusted parties or exotic cryptographic assumptions.

We now give a brief overview of our results in this area, beginning with a new cryptographic primitive that facilitates the construction of privacy preserving signature schemes.

1.2.1 Signatures with Flexible Public Keys

Signatures with flexible public keys (SFPK) are a type of digital signatures where public verification keys live in a system of equivalence classes induced by a relation $\mathcal{R}$. When a signature has been created under a pair of signing and verification keys it may later be transformed such that verification under a different verification key that is equivalent with respect to $\mathcal{R}$ becomes possible. Informally, an SFPK scheme offers two security properties:

Existential Unforgeability. Analogously to common digital signature schemes, we expect it to be infeasible for any computationally efficient adversary to forge valid signatures, even when given access to a signing oracle. In SFPK, this should hold even if the adversary may produce forgeries under any representative of the verification key under challenge.

Class Hiding. The transformation of a verification key to a different representative in its class should make the result of the transformation computationally indistinguishable from a key in a different equivalence class.

We first introduced this primitive in [Bac+18], where we gave a full formalization of the security properties of the scheme in the form of cryptographic games and offer an efficient instantiation based on bilinear groups. The functionality of SFPK can be seen as complementary to that of signatures on equivalence classes (SPS-EQ), first introduced by Fuchsbauer, Derler, and Slamanig [HS14]. In SPS-EQ, it is the messages which live in a system of equivalence classes, which already enables numerous applications to efficient anonymous credential constructions.

1.2.2 Applications of SFPK to Privacy-Preserving Signatures

One way to conceptualize both types of privacy-preserving signature schemes described above is a modular structure, which includes functionally distinct components:

1. An underlying signature on the message that is to be signed.
2. A membership proof that certifies the signer as a member of the claimed group or ring.

On a high level, to achieve unforgeability and signer anonymity it has to hold simultaneously that: (1) the underlying signature and membership proof belong to the same identity, which is part of the group and also (2) that no part of the final signature reveals this identity, at least without knowledge of a potentially available opening secret.

This forms the basis for the generic construction of such primitives in the *sign-encrypt-prove* approach. It works by encrypting both a regular signature on the message and a form of membership certificate, depending on the concrete scheme, and finally giving a non-interactive zero-knowledge (NIZK) proof about the encryption. The proof simultaneously ensures validity of the ciphertext, the contained signature, and membership proof and provides the cryptographic "glue" to bind everything to the same identity, all without revealing that identity.

This construction's generic nature comes at the price of less than optimal practical efficiency if instantiated with ill-fitting component schemes, in particular the NIZK proof becomes prohibitively expensive in terms of signer run time and signature size. Additionally, as described above, particularly in the ring signature case the use of a NIZK proof system is questionable due to its inherent requirement of either a common reference string shared by all parties, i.e. a trusted setup, or reliance on heuristic security arguments outside the cryptographic standard model, i.e. in the random oracle model.

In this thesis we show that SFPK offers a solution to these issues in privacy-preserving signatures.

SFPK and Group Signatures

Roughly following the generic approach outlined above, an SFPK signature scheme can provide the underlying signature component of a group signature. An SPS-EQ signature by the group manager where the signing member's SFPK verification key is the message serves as the certificate of membership in group. The crucial insight of our construction is that, if these two schemes are chosen such that both systems of equivalence classes match, then anonymity of the group signature can be achieved by simultaneous application of the transferability of public keys, respectively messages, to different representatives and the class hiding properties of both schemes. Unforgeability also follows from this joint change in representatives as well as the unforgeability of the underlying schemes with respect to equivalence classes of public keys, respectively messages. Observe that, in this construction, the *compatible choice of* SFPK *and* SPS-EQ *schemes* obviates the need for any general purpose proof system and thus eliminates the major source of practical inefficiency pointed out above.

Thus, in chapter 3 we present a group signature scheme in the standard model, with strongest-possible security properties in the static model due to Bellare, Micciancio and Warinschi [BMW03] that is in concrete terms more efficient than the state-of-the-art at the time of publication, and remains competitive even taking into account constructions with weaker security guarantees.

SFPK based group signatures also allow us to conceptually extend these frameworks. In the most expressive framework for group signature security modeling, the fully dynamic model due to Bootle et al. [Boo+16], group membership is not fixed forever a priori as in the static model, but an identity may or may not be a member in the group for any of a running number of epochs in a scheme's lifetime, as decided by the group manager. In chapter 4 we extend this model by the notion of *membership privacy*, that had not previously been considered for fully dynamic group signatures and which guards the group membership information from adversaries trying to make inferences about the groups members. The efficiency overhead of such anSFPK based membership private scheme is marginal compared to one that does not enjoy this feature.

SFPK and Ring Signatures

In chapter 3 we propose the *first sub-linear size ring signature scheme* in the literature that is rooted in the cryptographic standard model and does not require a trusted setup. Here again, anonymity is provided by SFPKs ability to transfer public keys from representative to representative, obscuring the original equivalence class. As the ring is chosen in an ad hoc fashion for each signature by the signer themselves, we cannot rely on a centrally trusted party to provide succinct membership certificates as in the case of group signatures. Instead, we observe that a full-blown zero-knowledge proof is not necessary to achieve the strongest security guarantees in ring signatures and instead the weaker property of witness indistinguishability is sufficient. In contrast to non-interactive zero-knowledge proof systems, non-interactive witness indistinguishable (NIWI) proofs can be built in the cryptographic standard model without trusted setup. We show how SFPK instantiations based on bilinear pairing groups integrate efficiently with NIWI proofs in the same setting and result in a scheme that has sub-linear signature size $\mathcal{O}(\sqrt{\ell})$ in the ring size ℓ and where signatures are even in concrete terms smaller in size than for comparable schemes with high security guarantees.

1.2.3 Logarithmic Size (Linkable) Ring Signatures

The notion of linkable ring signatures [LWW04] is an extension of the concept of ring signatures such that there is a public way of determining whether two signatures have been produced by the same signer. Linkable ring signatures yield a very elegant approach to e-voting [TW04]: Every voter is registered with their verification key. To cast a vote, all a voter has to do is to sign their vote on behalf of the ring of all registered voters. Linkability prevents voters from casting multiple votes. This can even be turned into an augmentation of the voting functionality by allowing voters to re-vote, where only the most recently cast votes of a set of votes that link counts.

Recently, linkable ring signature have also drawn attention in the domain of decentralized currencies, where they can be used to implement a mechanism for anonymized transactions. Linkable ring signatures are, for instance, used in a cryptocurrency called *Monero* [Noe15], where they allow payers to hide their identity in an anonymity set composed of identities from previous transactions. Currently, Monero uses a setup-free Schnorr based ring signature scheme [Sch90] where the size of signatures scales linearly in the size of the ring. To decrease the size of the transaction by default Monero uses small rings, which provide only a limited amount of anonymity. The anonymity definition for linkable ring signatures needs to be different from the definition for standard ring signatures. We will elaborate further on this topic below. In both of the above applications two aspects are of the essence:

- The ring signature scheme should not rely on a trusted setup. Especially in the e-voting application it is of paramount importance for the acceptance of such a system that there *cannot exist* a trapdoor that enables de-anonymization of voters.
- For practical purposes, e.g. for elections with millions of voters, the size of individual signatures should be essentially independent of the size of the ring of signers.

A ring signature scheme without trusted setup where the signature size is sub-linear in the size of the ring has long eluded the research community. The $\mathcal{O}(\sqrt{\ell})$ size scheme of chapter 3 was a promising first step and the construction based on SFPK is truly practical, but it is not the best we can hope for in terms of asymptotic signature size. The structure of typical ring signature constructions suggests it should be possible to achieve a size of $\mathcal{O}(\log(\ell))$, since we can think of a ring signature as a regular signature together with a kind of blinded pointer into the ring that allows to verify that the signature is valid under one of the keys in the ring. A straightforward approach to similar problems of proving membership in a list via a logarithmic size argument is through the use of Merkle trees. However, this approach fails in the case of ring signatures since the Merkle tree is only computationally binding, i.e. we are not perfectly guaranteed that a valid corresponding key is really in the ring. While Merkle trees give sufficient guarantees for many use cases in practice, this lack of perfect binding is a barrier in the proof of unforgeability for a ring signature scheme constructed in this way.

We address this issue by recognizing that Merkle trees in a way would give us too much of a property that is not enough, namely the computational binding actually holds for the whole ring, while we need perfect binding at just one position in the ring, namely where the signer verification key resides. At other places computational binding may be sufficient. Having recognized this, we make use of a primitive called somewhere perfectly binding hashing which guarantees exactly the right kind of binding property we need and allows us for the first time to construct logarithmic size ring signatures without any trusted setup.

In addition, we reconsider the security model for linkable ring signatures, significantly the notion linkable anonymity. Whereas earlier definitions of linkable anonymity fail to give any guarantees once the adversary has seen more than a single signature of the user in question, presumably since then the adversary can use the linking property to unmask the real signer, our new definition takes into account that a link between two signatures does not uniquely identify the signer unless the intersection of the two rings contains exactly one verification key, which must then belong to the signer. Indeed, an adversary may see many linking signatures without being able to further narrow down the identity of the real signer beyond the members of the ring that was signed under.

We then show that our logarithmic size ring signature can also be made linkable, retaining all the desirable characteristics of logarithmic size, no trusted setup and fulfilling our stronger notions of linkable security.

1.3 Structure of This Thesis

This thesis is organized in the following way.

In chapter 2, we introduce notation that is used throughout the thesis, and we recall fundamental definitions that allow us to keep further, chapter-specific preliminaries to a minimum in subsequent chapters. We also offer a brief recapitulation of basic notions relating to ring and group signatures.

We begin the detailed description of the contributions of this thesis in chapter 3 with a discussion of signatures with flexible public keys, their instantiations and a look at their basic application in the construction of ring signature and static group signature schemes.

Chapter 4 shows that the properties of SFPK enable new theoretical insights as well, by describing in detail an extension of the fully dynamic model of group signature security, which becomes feasible through the use of SFPK.

In chapter 5 we continue the exploration on the level of security models and generic constructions for the case of ring signatures, by giving our construction of ring signatures of logarithmic size without trusted setup and its linkable variant. This chapter also contains a redevelopment of the notion of linkable anonymity for linkable ring signatures, strengthening it substantially.

Finally, in chapter 6 we briefly summarize our results and provide possible avenues for future work based on this thesis.

1.3.1 Publication History

Most of the content of this thesis has previously been published in the following works:

- Michael Backes, Lucjan Hanzlik, Kamil Kluczniak, and Jonas Schneider. “Signatures with Flexible Public Key: Introducing Equivalence Classes for Public Keys.” In: *Advances in Cryptology – ASIACRYPT 2018, Part II.* ed. by Thomas Peyrin and Steven Galbraith. Vol. 11273. Lecture Notes in Computer Science. Brisbane, Queensland, Australia: Springer, Heidelberg, Germany, Dec. 2018, pp. 405–434. DOI: 10.1007/978-3-030-03329-3_14

- Michael Backes, Lucjan Hanzlik, and Jonas Schneider-Bensch. "Membership Privacy for Fully Dynamic Group Signatures." In: *ACM CCS 2019: 26th Conference on Computer and Communications Security*. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. ACM Press, Nov. 2019, pp. 2181–2198. DOI: 10.1145/3319535.3354257

- Michael Backes, Nico Döttling, Lucjan Hanzlik, Kamil Kluczniak, and Jonas Schneider. "Ring Signatures: Logarithmic-Size, No Setup - from Standard Assumptions." In: *Advances in Cryptology – EUROCRYPT 2019, Part III*. ed. by Yuval Ishai and Vincent Rijmen. Vol. 11478. Lecture Notes in Computer Science. Darmstadt, Germany: Springer, Heidelberg, Germany, May 2019, pp. 281–311. DOI: 10.1007/978-3-030-17659-4_10

Each of the main chapters chapters 3 to 5 includes a note on the publication history of its contents and any potential differences to the previously published materials.

2 Background and Building Blocks

In this chapter:

This chapter introduces our notation and a number of fundamental notions that will be relevant in all chapters that follow. Subsequent chapters will include chapter-specific preliminaries that is not presented here.

2.1 General Preliminaries

2.1.1 Notation and Other Conventions

Unless explicitly stated otherwise, we all presented algorithms are assumed to run in probabilistic polynomial time. This notion is the standard notion of asymptotic efficiency in cryptography. We often write that an algorithm *is* PPT to signify that it is probabilistic and has polynomially bounded runtime. We also extend this notion to algorithms that accept other inputs than bit strings, e.g. group elements. In that case, we assume there is a binary representation of the inputs and the run time is bounded by a polynomial in the length of that representation.

Throughout this thesis, we make use of a security parameter $\lambda \in \mathbb{N}$ when we want to make claims and give proofs about asymptotic security. In order to be able to argue about the run times of adversaries and other algorithms in terms of polynomials in the length of their inputs, we typically give algorithms an auxiliary input that encodes the security parameter in unary form 1^λ. An exception are algorithms that are given inputs that depend on the security parameter, such as public key material or scheme parameters, since these inputs are assumed to implicitly encode the security parameter.

We say a function $f : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is *negligible*, if for any positive polynomial p there exists a threshold $N \in \mathbb{N}$, such that for all $n > N$ it holds $f(n) < \frac{1}{p(n)}$. To ease notation, we write $f(\lambda) \leq \mathsf{negl}(\lambda)$ to mean that f is negligible in the security parameter λ.

2.1.2 The Game-Based Approach to Provable Security

Our approach to provable security follows the common game-based paradigm, where cryptographic primitives are defined as sets of abstract interfaces which implement a cryptographic functionality. The functionality is expressed through correctness requirements on the interactions between the different algorithms in the interface.

To formalize security notions, this approach makes use of probabilistic experiments, also known as *cryptographic games*, in which an adversary plays against a challenger. The challenger mediates access of the adversary to the interface offered by a cryptographic primitive. The challenger often restricts the capabilities of the adversary or manages some state that is relevant to determining whether the adversary has successfully subverted the security notion expressed by the experiment. In this context it is especially important which computational resources are afforded to the adversary.

We give an explanation of our presentation of experiments with oracles in example 2.1.

Pseudocode. We use an informal pseudocode to describe algorithms. We do not give a formal semantics for this pseudocode as the operations follow the conventions of the field and there is not much insight to gain from doing so in the context of this thesis.

The pseudocode allows a number of control flow constructs such as loops and branching as well as assignment statements and procedure calls. We assume the challenger can use simple data structures like sets and indexed arrays and that these can be efficiently implemented.

We denote by $y \leftarrow \mathcal{A}(x; r)$ the execution of algorithm $\mathcal{A}$ on input x with appropriately drawn randomness r, outputting y. If the specific randomness used is not relevant, we write just $y \leftarrow \mathcal{A}(x)$ instead. We denote by $[\mathcal{A}]$ the support of a probabilistic algorithm $\mathcal{A}$, i.e. those outputs which have a non-zero probability. For a finite set S we write $r \leftarrow S$ to mean that r is chosen uniformly at random from S. We will use $[n]$ to denote the set $\{1, \ldots, n\}$, and $[n]_0$ if we mean to include 0. We write $\vec{u}$ to denote a vector u and $\left(x_0 \ldots x_{|x|}\right)_{\mathsf{bin}}$ to denote the binary representation of some object x.

We will use the symbol $\emptyset$ to denote an undefined value.

Example 2.1: Example Experiment

Consider the following experiment, or *game*, between a challenger and an algorithm $\mathcal{A}$. The challenger, running the code of the experiment, sets up an environment containing the two variables target and counter. It initializes them to 5 and 0 respectively. The challenger then runs the adversary $\mathcal{A}$, giving it access to the example oracle OExample. The code of the oracle will be executed by the challenger if it is invoked by the adversary with the expected argument increment. To avoid clutter, especially in the presence of many oracles, we specify the oracles an adversary has access to below the code of the experiment.

```
Example − Experiment
target := $5$
counter := $0$
'done' ← $\mathcal{A}$(counter)
**if** counter = target
 **then** **return** true
**else** **return** false
- - - - - - - - - - - - - - -
$\mathcal{A}$ may invoke OExample.
```

```
OExample(increment)
counter := counter + increment
**return** counter
```

The experiment terminates, whenever the challenger reaches a **return** or **abort** statement. We then write Example − Experiment $\Rightarrow$ true for the event that the experiment terminates with outcome true and Example − Experiment $\Rightarrow$ false for the event that the experiment terminates with outcome false. If the termination is the result of an abort, the probability is split evenly between the two outcomes.

2.2 Basic Primitives

In this section we recall standard definitions of basic cryptographic building blocks, or primitives.

2.2.1 Digital Signature Schemes

Digital signature schemes allow a person in possession of a signing key to create digital signatures on documents, which may later be verified by anyone holding the person's verification key. This provides a non-repudiable piece of evidence regarding the originator of the signature, since only the person in possession of the signing key can create signatures that verify under the corresponding signing key and the signature will only be valid for the exact message that was originally signed.

Definition 2.1: Digital Signature

$\mathsf{KeyGen}(1^\lambda) \rightarrow (\mathsf{sk}, \mathsf{vk})$
: Takes as input a security parameter. Outputs a pair of signing key sk and verification key vk.

$\mathsf{Sign}(\mathsf{sk}, m) \rightarrow \sigma$
: Takes as input a signing key sk and a message $m \in \mathcal{M}$. Outputs a signature σ.

$\mathsf{Verify}(\mathsf{vk}, m, \sigma) \rightarrow r \in \{\mathsf{true}, \mathsf{false}\}$
: Takes as input a verification key vk, a message m and a signature σ. Outputs either true or false.

A signature scheme is *correct* if the following holds for all $\lambda \in \mathbb{N}$ and messages $m \in \mathcal{M}$: Given $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$, the output of $\mathsf{Verify}(\mathsf{vk}, m, \sigma)$ is true.

The security properties informally described above are captured in the well-known notion of existential unforgeability under chosen message attacks, which says that a malicious party cannot produce signatures under a signing key that is not known to the malicious party, unless it has previously seen such signatures, created by the legitimate owner of that signing key.[1]

Definition 2.2: Unforgeability Under Chosen-Message Attacks

For any signature scheme DS, consider the unforgeability under chosen message attacks experiment EUF-CMA between a challenger and an adversary $\mathcal{A}$:

[1]There is a subtle detail here in the informal phrasing: Namely, a signature scheme could well be unforgeable in the above sense and still allow an adversary to produce previously not seen signatures for messages that had been signed once by the legitimate owner, e.g. by modifying the existing signature in some way that does not affect its validity. If that is undesirable, one must ask for a signature scheme which is *strongly* existentially unforgeable. In that case, an adversary cannot even produce different signatures for previously signed messages.

```
$\mathsf{EUF\text{-}CMA}^{\mathcal{A}}_{\mathsf{DS}}(1^\lambda)$
$\mathbf{Q} := \emptyset$
$(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{DS.KeyGen}(1^\lambda)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}(\mathsf{vk})$
if $m^* \notin \mathbf{Q}$ and
   $\mathsf{DS.Verify}(\mathsf{vk}, m^*, \sigma^*) = \mathsf{true}$
 then return $\mathsf{true}$
else return $\mathsf{false}$
- - - - - - - - - - - - - - - - - - -
$\mathcal{A}$ may query OSign at any point during its runtime.
```

```
$\mathsf{OSign}(m)$
$\sigma \leftarrow \mathsf{DS.Sign}(\mathsf{sk}, m)$
$\mathbf{Q} := \mathbf{Q} \cup \{m\}$
return $\sigma$
```

We define the advantage of $\mathcal{A}$ in this experiment as

$$\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A},\mathsf{DS}}(\lambda) := \Pr\left[\mathsf{EUF\text{-}CMA}^{\mathcal{A}}_{\mathsf{DS}}(1^\lambda) \Rightarrow \mathsf{true}\right].$$

A signature scheme DS is *unforgeable under chosen-message attacks* if for any PPT adversary $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A},\mathsf{DS}}(\lambda) \leq \mathsf{negl}(\lambda).$$

There are many more security properties that can be considered for digital signatures and choosing the correct one for a given application is critical to prevent subtle subversion of the application security [Jac+19]. In our case the above standard notion is sufficient.

2.2.2 Public Key Encryption

Public key encryption allows a person to freely distribute encryption keys, which may be used to encrypt messages to that person. Ciphertext may only be decrypted using a secret decryption key.

Definition 2.3: Public Key Encryption

$\mathsf{KeyGen}(1^\lambda) \to (\mathsf{ek}, \mathsf{dk})$
: Takes as input a security parameter. Outputs a pair of encryption key ek and decryption key dk.

$\mathsf{Enc}(\mathsf{ek}, m) \to \mathsf{ctx}$
: Takes as input an encryption key ek and a message $m \in \mathcal{M}$. Outputs a

ciphertext ctx.

$\mathsf{Dec}(\mathsf{dk}, \mathsf{ctx}) \to r \in \mathcal{M} \cup \{\perp\}$
: Takes as input a decryption key dk and a ciphertext ctx. Outputs either a decryption result in $\mathcal{M}$ or the error symbol $\perp$ in case of decryption error

A public key encryption scheme is *perfectly correct* if the following holds for all $\lambda \in \mathbb{N}$ and messages $m \in \mathcal{M}$: Given $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and $\mathsf{ctx} \leftarrow \mathsf{Enc}(\mathsf{ek}, m)$, the output of $\mathsf{Dec}(\mathsf{dk}, \mathsf{ctx})$ is m.[a]

[a]For some encryption schemes, there may be a negligible possibility of decryption error, even for well-formed ciphertexts. In this case we speak merely of *correctness* instead of perfect correctness.

The basic security notion of public key encryption is indistinguishability of ciphertexts under chosen plaintext attacks. It says that an attacker may not learn a single bit about messages encrypted in ciphertexts it did not create itself, even if it knows the message that is encrypted must be one it has seen before.

Definition 2.4: Indistinguishability Under Chosen Plaintext Attacks

For any public key encryption scheme PKE, consider the ciphertext indistinguishability under chosen plaintext attacks experiment IND-CPA between a challenger and a two-stage adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$:

```
$\mathsf{IND\text{-}CPA}^{\mathcal{A}}_{\mathsf{PKE}}(1^\lambda)$

$(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$
$(\mathsf{state}, m_0, m_1) \leftarrow \mathcal{A}_0(\mathsf{ek})$
$b \leftarrow \{0,1\}$
$\mathsf{ctx} \leftarrow \mathsf{PKE.Enc}(\mathsf{ek}, m_b)$
$b' \leftarrow \mathcal{A}_1(\mathsf{state}, \mathsf{ctx})$
**if** $b' = b$ **then return** true
**else return** false
```

We define the advantage of $\mathcal{A}$ in this experiment as

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{PKE}}(\lambda) := \left| \Pr\left[\mathsf{IND\text{-}CPA}^{\mathcal{A}}_{\mathsf{PKE}}(1^\lambda) \Rightarrow \mathsf{true}\right] - \frac{1}{2} \right|.$$

A public key encryption scheme PKE has *indistinguishable ciphertexts under* *chosen plaintext attacks* if for any PPT adversary $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{PKE}}(\lambda) \leq \mathsf{negl}(\lambda)\,.$$

In our use of public key encryption in later chapters it will often occur that, for instance, the anonymity of a group signature depends on the inability of an adversary to determine the intended recipient of a given ciphertext, i.e. a ciphertext itself should not reveal which public key was used to create it. We formalize this in the vein of [Bel+01].

Definition 2.5: Key Privacy Under Chosen Plaintext Attacks

For any public key encryption scheme PKE, consider the key privacy under chosen plaintext attacks experiment IK-CPA between a challenger and a two-stage adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$:

```
$\mathsf{IK\text{-}CPA}^{\mathcal{A}}_{\mathsf{PKE}}(1^\lambda)$

$(\mathsf{ek}_0, \mathsf{dk}_0) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$
$(\mathsf{ek}_1, \mathsf{dk}_1) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$
$(\mathsf{state}, m) \leftarrow \mathcal{A}_0(\mathsf{ek}_0, \mathsf{ek}_1)$
$b \leftarrow \{0,1\}$
$\mathsf{ctx} \leftarrow \mathsf{PKE.Enc}(\mathsf{ek}_b, m)$
$b' \leftarrow \mathcal{A}_1(\mathsf{state}, \mathsf{ctx})$
**if** $b' = b$ **then return** true
**else return** false
```

We define the advantage of $\mathcal{A}$ in this experiment as

$$\mathsf{Adv}^{\mathsf{IK\text{-}CPA}}_{\mathcal{A},\mathsf{PKE}}(\lambda) := \left|\Pr\left[\mathsf{IK\text{-}CPA}^{\mathcal{A}}_{\mathsf{PKE}}(1^\lambda) \Rightarrow \mathsf{true}\right] - \frac{1}{2}\right|.$$

A public key encryption scheme PKE has *key privacy under chosen plaintext attacks* if for any PPT adversary $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathsf{IK\text{-}CPA}}_{\mathcal{A},\mathsf{PKE}}(\lambda) \leq \mathsf{negl}(\lambda)\,.$$

A priori it is not clear that public key encryption schemes should satisfy this property, since it does not seem relevant to the other security properties expected of public key encryption. After all, the public key is by definition not hidden itself. Luckily for us, many well-known public key encryption schemes fulfill this property, e.g. the ElGamal encryption scheme. For more details regarding key privacy and more examples of schemes that fulfill the property, please consider [Bel+01].

2.2.3 Non-interactive Proof Systems

Interactive proof systems a prover and a verifier to establish certain facts between each other, often overcoming some differential in computational power between the parties.

For instance, using a type of proof system called zero-knowledge proof of knowledge a prover can convince a verifier that the prover has knowledge of an object, called *a witness*, that satisfies a given statement without revealing information about anything more than this fact, in particular without revealing information about the witness itself.

In many cryptographic protocols such proofs allow us to argue for stronger security notions, since they enable parties to keep each other in check without revealing their cryptographic secrets to each other.

Astonishingly, cryptographers have discovered ways to avoid the interactivity of these proofs, allowing provers to convince verifiers of some statement by giving a single message.

Let $\mathcal{R}$ be an efficiently computable binary relation, where for $\mathcal{R}(\mathsf{x}, \mathsf{w}) \Rightarrow \mathsf{true}$ we call x a statement and w a witness. Moreover, we denote by $\mathcal{L}_\mathcal{R}$ the language consisting of statements in $\mathcal{R}$, i.e. $\mathcal{L}_\mathcal{R} = \{\mathsf{x} | \exists \mathsf{w} : (\mathsf{x}, \mathsf{w}) \in \mathcal{R}\}$.

Definition 2.6: Non-interactive Proof System

Let $\mathcal{R}$ be an efficiently computable witness relation and $\mathcal{L}_\mathcal{R}$ be the language accepted by $\mathcal{R}$.

$\mathsf{Prove}(1^\lambda, \mathsf{x}, \mathsf{w}) \rightarrow \pi \cup \{\perp\}$
: Takes as input a security parameter, a statement x and a witness w. Outputs either a proof π or an error $\perp$.

$\mathsf{Verify}(\mathsf{x}, \pi) \rightarrow r \in \{\mathsf{true}, \mathsf{false}\}$
: Takes as input a statement x and proof π Outputs either true or false.

A non-interactive proof system NIP for $\mathcal{L}_\mathcal{R}$ is called *perfectly complete*, if for all $\lambda \in \mathbb{N}$, all statements $\mathsf{x} \in \mathcal{L}_\mathcal{R}$ and all witnesses w, such that $\mathcal{R}(\mathsf{x}, \mathsf{w}) = \mathsf{true}$, that given any $\pi \leftarrow \mathsf{Prove}(\mathsf{x}, \mathsf{w})$ it holds $\mathsf{Verify}(\mathsf{x}, \pi) = \mathsf{true}$.

The price for this non-interactivity is paid in the weaker security guarantees we can give about these kinds of non-interactive proofs. The strongest forms of cryptographic security of the witness, namely zero-knowledge can only be achieved non-interactively by either assuming the existence of random oracles, a strong idealization of cryptographic hash functions, or by assuming that every party has access to a precomputed piece of information, a common reference string.

Definition 2.7: Non-interactive Proof System with Setup

A non-interactive proof system may NIP, in addition to Prove, Verify, include a setup procedure.

$\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$
: Takes as input the security parameter and outputs a common reference string crs.

In this case, both Prove and Verify take a common reference string crs as an additional argument and *completeness* should hold with respect to any $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$ for all $\lambda \in \mathbb{N}$.

Definition 2.8: Soundness

A non-interactive proof system NIP is *sound* if for all $\lambda \in \mathbb{N}$ and any PPT adversary $\mathcal{A}$ the following holds: Given $\mathsf{crs} \leftarrow \mathsf{NIP.Setup}(1^\lambda)$ and $\mathsf{x}, \pi \leftarrow \mathcal{A}(\mathsf{crs})$ we have

$$\Pr[\mathsf{NIP.Verify}(\mathsf{crs}, \mathsf{x}, \pi) = \mathsf{true} \text{ and } \mathsf{x} \notin \mathcal{L}_{\mathcal{R}}] \leq \mathsf{negl}(\lambda)\,. \tag{2.1}$$

For any adversary $\mathcal{A}$, we refer to the probability in eq. (2.1) as its soundness advantage $\mathsf{Adv}^{\mathsf{Soundness}}_{\mathcal{A},\mathsf{NIP}}(\lambda)$. If the advantage vanishes completely, we say NIP has *perfect soundness.*

It has been shown that security proofs in the random oracle can only offer heuristic guarantees, since there are non-trivial cases where it is impossible to instantiate random oracles securely[CGH98]. Furthermore, in scenarios where we have to assume none of the parties trust each other it is unclear how to implement a precomputation step that could not be tampered with to the advantage of one party. Because of this, these so-called *trusted setup* approaches are a serious concern in real-world applications such as cryptocurrencies.

Remark

In the following we will give all definitions as though every proof system had a setup, as described in definition 2.7, unless explicitly stated otherwise. Note that any non-interactive proof system without setup can be made to include a trivial setup algorithm which simply returns its input, the security parameter as crs.

Definition 2.9: Non-interactive Proof System with Simulator

A non-interactive proof system with setup NIP has a simulator if, in addition to the interface defined in definition 2.7, the following two PPT algorithms exist.

$\mathsf{SimSetup}(1^\lambda) \to (\mathsf{crs}, \rho)$
: Takes as input a security parameter. Outputs a common reference string crs and a simulation trapdoor ρ.

$\mathsf{Sim}(\mathsf{crs}, \rho, \mathsf{x}) \to \pi$
: Takes as input a common reference string crs, a simulation trapdoor ρ and a statement x. Outputs a proof π.

Definition 2.10: Zero-Knowledge

For any non-interactive proof system with simulator NIP, consider the zero-knowledge experiment ZK between a challenger and an adversary $\mathcal{A}$:

```
$\mathsf{ZK}^{\mathcal{A}}_{\mathsf{NIP}}(1^\lambda)$
$b \leftarrow \{0,1\}$
if $b = 1$ then
   $\mathsf{crs} \leftarrow \mathsf{NIP.Setup}(1^\lambda)$
else
   $(\mathsf{crs}, \rho) \leftarrow \mathsf{NIP.SimSetup}(1^\lambda)$
$b' \leftarrow \mathcal{A}(\mathsf{crs})$
if $b' = b$ then return true
else return false
```

$\mathcal{A}$ may query OProve at any point during its runtime.

```
$\mathsf{OProve}(\mathsf{x}, \mathsf{w})$
if $\mathcal{R}(\mathsf{x}, \mathsf{w}) = \mathsf{false}$
 then return $\perp$
elseif $b = 1$
 then return $\mathsf{NIP.Prove}(\mathsf{crs}, \mathsf{x}, \mathsf{w})$
else return $\mathsf{NIP.Sim}(\mathsf{crs}, \rho, \mathsf{x})$
```

We define the advantage of $\mathcal{A}$ in this experiment as

$$\mathsf{Adv}^{\mathsf{ZK}}_{\mathcal{A},\mathsf{NIP}}(\lambda) := \left| \Pr\left[\mathsf{ZK}^{\mathcal{A}}_{\mathsf{NIP}}(1^\lambda) \Rightarrow \mathsf{true}\right] - \frac{1}{2} \right| .$$

A non-interactive proof system NIP is *(computationally) zero-knowledge* if for any PPT adversary $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathsf{ZK}}_{\mathcal{A},\mathsf{NIP}}(\lambda) \leq \mathsf{negl}(\lambda) .$$

On the other hand, a weaker property called witness indistinguishability is possible non-interactively without assuming any kind of trusted precomputation or shared access to a random oracle.

Definition 2.11: Witness Indistinguishability

For any non-interactive proof system NIP, consider the witness indistinguishability experiment WI between a challenger and a two-stage adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$:

```
$\mathsf{WI}^{\mathcal{A}}_{\mathsf{NIP}}(1^\lambda)$

$\mathsf{crs} \leftarrow \mathsf{NIP.Setup}(1^\lambda)$
$(\mathsf{state}, \mathsf{x}, \mathsf{w}_0, \mathsf{w}_1) \leftarrow \mathcal{A}_0(\mathsf{crs})$
$b \leftarrow \{0,1\}$
$\pi \leftarrow \mathsf{NIP.Prove}(\mathsf{crs}, \mathsf{x}, \mathsf{w}_b)$
$b' \leftarrow \mathcal{A}_1(\mathsf{state}, \pi)$
**if** $b' = b$ **then return** true
**else return** false
```

We define the advantage of $\mathcal{A}$ in this experiment as

$$\mathsf{Adv}^{\mathsf{WI}}_{\mathcal{A},\mathsf{NIP}}(\lambda) := \left| \Pr\left[\mathsf{WI}^{\mathcal{A}}_{\mathsf{NIP}}(1^\lambda) \Rightarrow \mathsf{true}\right] - \frac{1}{2} \right| .$$

A non-interactive proof system NIP is *witness-indistinguishable* if for any PPT adversary $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathsf{WI}}_{\mathcal{A},\mathsf{NIP}}(\lambda) \leq \mathsf{negl}(\lambda) .$$

If the advantage vanishes completely, we say the proof system is *perfectly witness-indistinguishable.*

Non-interactive witness-indistinguishable proofs can be constructed from NIZK proofs and derandomization assumptions [DN00b; BOV03], from bilinear pairings [GOS06] and indistinguishability obfuscation [BP15].

It may be advantageous for a soundness adversary to see simulated proofs. A strong requirement that makes this impossible is the notion of simulation sound extractability that requires that whenever an adversary produces a valid proof, we can extract a witness from this proof, even if the adversary has seen many simulated proofs before.

Definition 2.12: Non-interactive Proof System with Extractor

A non-interactive proof system with setup NIP has an extractor if, in addition to the interface defined in definition 2.7, the following three PPT algorithms exist.

$\mathsf{ExtSetup}(1^\lambda) \rightarrow (\mathsf{crs}, \rho, \xi)$
: Takes as input a security parameter. Outputs a common reference string crs, a simulation trapdoor ρ and an extraction trapdoor.

$\mathsf{Sim}(\mathsf{crs}, \rho, \mathsf{x}) \rightarrow \pi$
: Takes as input a common reference string crs, a simulation trapdoor ρ and a statement x. Outputs a proof π.

$\mathsf{Ext}(\mathsf{crs}, \xi, \mathsf{x}, \pi) \rightarrow \mathsf{w}$
: Takes as input a common reference string crs, an extraction trapdoor ξ and a statement x as well as proof π. Outputs a witness w.

Definition 2.13: Simulation Sound Extractability

For any non-interactive proof system with extraction NIP, consider the extractability experiment SSE between a challenger and an adversary $\mathcal{A}$:

$\mathsf{SSE}^{\mathcal{A}}_{\mathsf{NIP}}(1^\lambda)$

```
$(\mathsf{crs}, \rho, \xi) \leftarrow \mathsf{NIP.ExtSetup}(1^\lambda)$
$(\mathsf{x}, \pi) \leftarrow \mathcal{A}(\mathsf{crs}, \xi)$
$\mathsf{w} \leftarrow \mathsf{NIP.Ext}(\mathsf{crs}, \xi, \mathsf{x}, \pi)$
if $\mathsf{NIP.Verify}(\mathsf{crs}, \mathsf{x}, \pi)$ and
   $\mathcal{R}(\mathsf{x}, \mathsf{w}) = \mathsf{false}$ and
   $(\mathsf{x}, \pi) \notin \mathbf{Q}$
 then return true
else return false
```

$\mathcal{A}$ may query OProve at any point during its runtime.

$\mathsf{OProve}(\mathsf{x}, \mathsf{w})$

```
if $\mathcal{R}(\mathsf{x}, \mathsf{w}) = \mathsf{false}$
 then return $\bot$
$\pi \leftarrow \mathsf{NIP.Sim}(\mathsf{crs}, \rho, \mathsf{x})$
$\mathbf{Q} := \mathbf{Q} \cup \{(\mathsf{x}, \pi)\}$
return $\pi$
```

We define the advantage of $\mathcal{A}$ in this experiment as

$$\mathsf{Adv}^{\mathsf{SSE}}_{\mathcal{A},\mathsf{NIP}}(\lambda) := \Pr\left[\mathsf{SSE}^{\mathcal{A}}_{\mathsf{NIP}}(1^\lambda) \Rightarrow \mathsf{true}\right].$$

A non-interactive proof system NIP with extraction has *simulation sound extractability* if for any PPT adversary $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathsf{SSE}}_{\mathcal{A},\mathsf{NIP}}(\lambda) \leq \mathsf{negl}(\lambda)\,.$$

2.3 Background on Group Signatures

Here, we recall the formalization of static group signatures due to Bellare, Micciancio and Warinschi [BMW03], which will be the model for our construction in chapter 3 and is the basis for the extended model due to Bootle et al. [Boo+16] that is reconsidered in chapter 4.

Definition 2.14: Static Group Signature Scheme

$\mathsf{KeyGen}(1^\lambda, n) \to (\mathsf{gpk}, \mathsf{gmsk}, \vec{\mathsf{gsk}})$
Takes as input a security parameter 1^λ and the group size $n \in \mathbb{N}$. Outputs the group verification key gpk, the group manager secret key gmsk and $\vec{\mathsf{gsk}}$, a vector of size n, where $\vec{\mathsf{gsk}}[i]$ is the signing key of the i-th group member.

$\mathsf{Sign}(\vec{\mathsf{gsk}}[i], m) \to \sigma$
Takes as input the signing key of the i-th group member $\vec{\mathsf{gsk}}[i]$ and a message $m \in \mathcal{M}$. Outputs a signature σ.

$\mathsf{Verify}(\mathsf{gpk}, m, \sigma) \to r \in \{\mathsf{true}, \mathsf{false}\}$
Takes as input the group verification key gpk, a message m and a signature σ. Outputs either true or false.

$\mathsf{Open}(\mathsf{gmsk}, m, \sigma) \to i \in [n] \cup \{\perp\}$
Takes as input the group manager secret key gmsk, message m and a signature σ. Outputs an identity i or the symbol $\perp$ in case of failure.

For simplicity group members are assigned consecutive integer identities from the set $[n]$.
We say that a static group signature scheme is *correct* if the following holds for all $\lambda, n \in \mathbb{N}$ and $m \in \mathcal{M}$: Given $(\mathsf{gpk}, \mathsf{gmsk}, \vec{\mathsf{gsk}}) \leftarrow \mathsf{KeyGen}(1^\lambda, n)$, for all $i \in [n]$, whenever $\sigma \leftarrow \mathsf{Sign}(\vec{\mathsf{gsk}}[i], m)$ we have

$$\mathsf{Verify}(\mathsf{gpk}, m, \sigma) = \mathsf{true} \qquad \text{and} \qquad \mathsf{Open}(\mathsf{gmsk}, m, \sigma) = i.$$

2.3.1 Security of Static Group Signatures

Prior to the work of [BMW03], many notions of security for group signatures were considered, including notions of unforgeability, exculpability, non-frameability, and unlinkability. The BMW model shows that these can be subsumed under two strong notions of anonymity and traceability.

Full-Anonymity. Informally, anonymity means that it should be hard for an adversary to recover the identity of the signer from a signature without knowledge of the group manager's secret key. To properly model collusion among group members the adversary is given the secret keys of all group members. Moreover, the adversary can use an opening oracle OOpen$(\cdot,\cdot)$, which models the possibility of the adversary seeing previous openings.

Definition 2.15: Full Anonymity

For static group signature scheme GS, consider the anonymity experiment sG-Anonymity for group size n between a challenger and a two-stage adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$.

sG-Anonymity$^{\mathcal{A}}_{\mathsf{GS}}(\lambda, n)$

```
$b \leftarrow \{0,1\}$
$\mathbf{Q} := \emptyset$
$(\mathsf{gpk}, \mathsf{gmsk}, \vec{\mathsf{gsk}}) \leftarrow \mathsf{GS.KeyGen}(1^\lambda, n)$
$(\mathsf{state}, i_0, i_1, m^*) \leftarrow \mathcal{A}_0(\mathsf{gpk}, \vec{\mathsf{gsk}})$
$\sigma^* \leftarrow \mathsf{GS.Sign}(\vec{\mathsf{gsk}}[i_b], m^*)$
$b' \leftarrow \mathcal{A}_1(\mathsf{state}, \sigma^*)$
**if** $(m^*, \sigma^*) \in \mathbf{Q}$
 **then** **return** false
**elseif** $b' = b$ **then** **return** true
**else return** false
```

$\mathcal{A}_0$ and $\mathcal{A}_1$ may query OOpen at any point during their run time.

OOpen(m, σ)

```
$\mathbf{Q} := \mathbf{Q} \cup \{(m, \sigma)\}$
**return** $\mathsf{GS.Open}(\mathsf{gmsk}, m, \sigma)$
```

We define the advantage of $\mathcal{A}$ in the above experiment as

$$\mathsf{Adv}^{\mathsf{sG\text{-}Anonymity}_n}_{\mathcal{A},\mathsf{GS}}(\lambda) = \left| \Pr\left[\mathsf{sG\text{-}Anonymity}^{\mathsf{RS}}_{\mathcal{A}}(1^\lambda, n) \Rightarrow \mathsf{true}\right] - \frac{1}{2} \right|.$$

A group signature scheme GS is *fully anonymous* if for all PPT adversaries $\mathcal{A}$ and group sizes $n = \mathsf{poly}(\lambda)$, we have

$$\mathsf{Adv}_{\mathcal{A},\mathsf{GS}}^{\mathsf{sG\text{-}Anonymity}_n}(\lambda) \leq \mathsf{negl}(\lambda)\,.$$

This property subsumes the notion of unlinkability that was considered before [BMW03].

Full-Traceability. The next required property is called traceability. In case of misuse, we would like the group manager to always be able to identify the signer. In particular, this means that it should not be possible to create a signature that cannot be opened. Moreover, a colluding set S of group members should not be able to frame an honest member, i.e. create a signature that opens to a member that is not in S.

Definition 2.16: Full Traceability

For static group signature scheme GS, consider the traceability experiment sG-Traceability for group size n between a challenger and an adversary $\mathcal{A}$.

```
sG-Traceability$(1^\lambda, n)$

$\mathbf{Q} := \emptyset; \mathbf{C} := \emptyset$
$(\mathsf{gpk}, \mathsf{gmsk}, \vec{\mathsf{gsk}}) \leftarrow \mathsf{GS.KeyGen}(1^\lambda, n)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}(\mathsf{gmsk}, \mathsf{gpk})$
if $\mathsf{GS.Verify}(\mathsf{gpk}, m^*, \sigma^*) = \mathsf{false}$ then return false
if $\mathsf{GS.Open}(\mathsf{gmsk}, m^*, \sigma^*) = \perp$ then return true
elseif $\mathsf{GS.Open}(\mathsf{gmsk}, m^*, \sigma^*) = i$ and $i \notin \mathbf{C}$ and $(i, m^*) \notin \mathbf{Q}$
 then return true
else return false

$\mathcal{A}$ may query OSign and OCorrupt at any point during its run time.
```

```
OSign$(i, m)$

$\mathbf{Q} := \mathbf{Q} \cup \{(i, m)\}$
return $\mathsf{GS.Sign}(\vec{\mathsf{gsk}}[i], m)$
```

```
OCorrupt$(i)$

$\mathbf{C} := \mathbf{C} \cup \{i\}$
return $\vec{\mathsf{gsk}}[i]$
```

We define the advantage of $\mathcal{A}$ in this experiment as

$$\mathsf{Adv}^{\text{sG-Traceability}}_{\mathcal{A},\mathsf{GS}}(\lambda) := \Pr\left[\text{sG-Traceability}^{\mathcal{A}}_{\mathsf{GS}}(1^\lambda, n) \Rightarrow \mathsf{true}\right].$$

A static group signature scheme GS is *fully traceable* if for any PPT adversary $\mathcal{A}$, and all group sizes $n = \mathsf{poly}(\lambda)$ we have

$$\mathsf{Adv}^{\text{sG-Traceability}}_{\mathcal{A},\mathsf{GS}}(\lambda) \leq \mathsf{negl}(\lambda).$$

This property implies the notions of unforgeability, exculpability, non-frameability and coalition-resistance that were considered before [BMW03].

2.4 Background on Ring Signatures

Here, we state the formal model of ring signatures as defined by [BKM06]. They give a plethora of possible security definitions, of varying strength, but we only state the strongest notions of security, since these are the ones relevant to our constructions.

Definition 2.17: Ring Signature

$\mathsf{KeyGen}(1^\lambda) \to (\mathsf{rsk}, \mathsf{rvk})$
: Takes as input a security parameter 1^λ. Outputs a pair $(\mathsf{rsk}, \mathsf{rvk})$ of signing and verification keys.

$\mathsf{Sign}(m, \mathsf{rsk}^{(s)}, \mathsf{R}) \to \varsigma$
: Takes as input a message $m \in \mathcal{M}$, a signing key $\mathsf{rsk}^{(s)}$ and an ordered set (a ring) of verification keys $\mathsf{R} = \left(\mathsf{rvk}^{(1)}, \ldots, \mathsf{rvk}^{(n)}\right)$ with $\mathsf{rvk}^{(s)} \in \mathsf{R}$. Outputs a signature ς.

$\mathsf{Verify}(m, \varsigma, \mathsf{R}) \to r \in \{\mathsf{true}, \mathsf{false}\}$
: Takes as input a message m, signature ς, and a ring of verification keys R. Outputs either true or false.

A ring signature scheme is *correct* if for all $\lambda \in \mathbb{N}$, all ring sizes $n = \mathsf{poly}(\lambda)$, any $\left\{\left(\mathsf{rsk}^{(i)}, \mathsf{rvk}^{(i)}\right)\right\}_{i=1}^{n}$ generated with $\mathsf{KeyGen}(1^\lambda)$, any $s \in [n]$ and any message $m \in \mathcal{M}$, we have $\mathsf{Verify}(m, \mathsf{Sign}(m, \mathsf{rsk}^{(s)}, \mathsf{R}), \mathsf{R}) = \mathsf{true}$, where $\mathsf{R} = \left(\mathsf{rvk}^{(1)}, \ldots, \mathsf{rvk}^{(n)}\right)$.

2.4.1 Security of Ring Signatures

Ring signatures should be unforgeable with respect to the specific message that was signed and the ring of public keys that it was signed to, i.e. besides being unable to forge signatures on new messages, an adversary should also be unable to create a new signature for a previously signed message but with a modified ring.

Definition 2.18: Unforgeability w.r.t. Insider Corruption

For ring signature scheme RS, and key universe size q consider the unforgeability with respect to insider corruption experiment RUF-IC between a challenger and an adversary $\mathcal{A}$.

```
$\mathsf{RUF\text{-}IC}_{\mathcal{A}}^{\mathsf{RS}}(1^\lambda, q)$

$\mathbf{Q} := \emptyset, \mathbf{C} := \emptyset$
for $i = 1 \dots q$ do
    $(\mathsf{rsk}^i, \mathsf{rvk}^i) \leftarrow \mathsf{RS.KeyGen}(1^\lambda)$
$(m^*, \varsigma^*, \mathsf{R}^*) \leftarrow \mathcal{A}\left(S := \{\mathsf{rvk}^i\}_{i=1}^q\right)$
if $\mathsf{RS.Verify}(m^*, \varsigma^*, \mathsf{R}^*) = \mathsf{true}$
    and $(m^*, \mathsf{R}^*) \notin \mathbf{Q}$
    and $\mathsf{R}^* \subseteq S \setminus \mathbf{C}$
 then return true
else return false
```

$\mathcal{A}$ may query OSign and OCorrupt at any point during its run time.

```
$\mathsf{OSign}(m, s, \mathsf{R})$

$\mathbf{Q} := \mathbf{Q} \cup \{(m, \mathsf{R})\}$
if $\mathsf{rvk}^s \in \mathsf{R}$ then
    $\varsigma \leftarrow \mathsf{RS.Sign}(\mathsf{rsk}^s, m, \mathsf{R})$
    return $\varsigma$
else return $\perp$
```

```
$\mathsf{OCorrupt}(i)$

$\mathbf{C} := \mathbf{C} \cup \{\mathsf{rvk}^i\}$
return $\mathsf{rsk}^i$
```

We define the advantage of $\mathcal{A}$ in the above experiment as

$$\mathsf{Adv}_{\mathcal{A},\mathsf{RS},q}^{\mathsf{RUF\text{-}IC}}(\lambda) = \Pr\left[\mathsf{RUF\text{-}IC}_{\mathcal{A}}^{\mathsf{RS}}(1^\lambda, q) \Rightarrow \mathsf{true}\right].$$

A signature scheme RS is *unforgeable with respect to insider corruption* if for all PPT adversaries $\mathcal{A}$, and for all $q = \mathsf{poly}(\lambda)$ we have

$$\mathsf{Adv}^{\mathsf{RUF\text{-}IC}}_{\mathcal{A},\mathsf{RS},q}(\lambda) \leq \mathsf{negl}(\lambda)\,.$$

A ring signature scheme should also be anonymous, i.e. it should be infeasible for an attacker, given a signature, to establish which ring member actually created this signature. In its strongest form, this property should hold true, even if the adversary has access to all key material, including the signing keys, of the members of the ring.

Definition 2.19: Anonymity Against Full Key Exposure

For ring signature scheme RS and key universe size q consider the anonymity against full key exposure experiment RAnon-FKE between a challenger and a two-stage adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$.

$\mathsf{RAnon\text{-}FKE}^{\mathsf{RS}}_{\mathcal{A}}(1^\lambda, q)$

```
for $i = 1 \dots q$ do
    $(\mathsf{rsk}^i, \mathsf{rvk}^i) \leftarrow \mathsf{RS.KeyGen}(1^\lambda; r_i)$
$(\mathsf{state}, m, i_0, i_1, \mathsf{R}) \leftarrow \mathcal{A}_0(\{r_i\}_{i=1}^q)$
    $b \leftarrow \{0,1\}$
if $\mathsf{rvk}^{i_0} \notin \mathsf{R}$ or $\mathsf{rvk}^{i_1} \notin \mathsf{R}$ then
    $\varsigma := \perp$
else
    $\varsigma \leftarrow \mathsf{RS.Sign}(\mathsf{rsk}^{i_b}, m, \mathsf{R})$
$b' \leftarrow \mathcal{A}_1(\mathsf{state}, \varsigma)$
if $b = b'$ then return true
else return false
```

adv_0 and $\mathcal{A}_1$ may query OSign at any point during their run time.

$\mathsf{OSign}(m, s, \mathsf{R})$

```
$\mathbf{Q} := \mathbf{Q} \cup \{(m, \mathsf{R})\}$
if $\mathsf{rvk}_s \in \mathsf{R}$ then
    $\varsigma \leftarrow \mathsf{Sign}(m, \mathsf{rsk}^{(s)}, \mathsf{R})$
    return $\varsigma$
else return $\perp$
```

We define the advantage of $\mathcal{A}$ in the above experiment as

$$\mathsf{Adv}^{\mathsf{RUF\text{-}IC}}_{\mathcal{A},\mathsf{RS},q}(\lambda) = \Pr\left[\mathsf{RUF\text{-}IC}^{\mathsf{RS}}_{\mathcal{A}}(1^\lambda, q) \Rightarrow \mathsf{true}\right].$$

A signature scheme RS is *unforgeable with respect to insider corruption* if for all PPT adversaries $\mathcal{A}$, and for all $q = \mathsf{poly}(\lambda)$ we have

$$\mathsf{Adv}^{\mathsf{RUF\text{-}IC}}_{\mathcal{A},\mathsf{RS},q}(\lambda) \leq \mathsf{negl}(\lambda)\,.$$

3 Signatures with Flexible Public Keys

In this chapter:

3.1 Introduction

Digital signature schemes are commonly used to ensure two security properties: integrity of the signed message and authenticity of the signer's identity. In some applications, these requirements, as implemented in traditional signature schemes are *not quite* right for the job. They might be too rigid and prohibit efficient implementation of some useful functionality. As one example, consider sanitizable signatures [Ate+05b], which allow designated parties to redact certain parts of a signed message, while otherwise preserving its integrity and the signer's intent. This could be implemented, in principle, *Publication History*

Most of the material presented in this chapter is based on, and was first published in

> Michael Backes, Lucjan Hanzlik, Kamil Kluczniak, and Jonas Schneider. "Signatures with Flexible Public Key: Introducing Equivalence Classes for Public Keys." In: *Advances in Cryptology – ASIACRYPT 2018, Part II.* ed. by Thomas Peyrin and Steven Galbraith. Vol. 11273. Lecture Notes in Computer Science. Brisbane, Queensland, Australia: Springer, Heidelberg, Germany, Dec. 2018, pp. 405–434. DOI: 10.1007/978-3-030-03329-3_14.

The instantiation of signatures with flexible public keys presented in construction 3.4 was first given in [BHS19].

using regular digital signatures, e.g. by having the signer simply re-sign the redacted messages. It turns out that more efficient schemes can be designed, where the original signer does not have to be involved in the creation of a redacted signature. Since this presents a deviation from the strong integrity guarantees of traditional signatures, care has to be taken to formulate new security guarantees, which capture exactly the kind of desired deviation, and nothing more. If successful, the new primitive can possibly be instantiated differently from traditional signatures and its thus extended functionality can find further application in different contexts beyond the application originally envisioned.

In this chapter, we introduce such a primitive, called signatures with flexible public keys (SFPK). In contrast to the sanitizable signatures described above our primitive allows for a relaxation of signer authenticity instead. If this property was completely dropped, any impostor could sign messages on behalf of a legitimate signer. Rather than this, with our primitive authenticity holds with respect to a *specific but hidden* previously established legitimate signer.

Signatures with flexible public key formalize a signature scheme, where verification and signing keys live in a system of equivalence classes induced by a relation $\mathcal{R}$. Given a signing or verification key it is possible to transform the key into a different representative of the same equivalence class, i.e., the pair of old key and new key are related via $\mathcal{R}$. Signatures under a transformed signing key can be verified using an equally transformed verification key. Thus, one possibility of attack that was not possible previously is that an adversary might be able to create signatures which verify under a key in the class of the challenge signing key. We must therefore extend the requirement of unforgeability of signatures to the whole equivalence class of the given key under attack.

Additionally, it should be infeasible, without a trapdoor, to check whether two keys are in the same class. This property, which we call *class hiding*, ensures that given an old verification key, a signature under a fresh representative is indistinguishable from a signature under a different newly generated key, which lives in a different class altogether with overwhelming probability. Intuitively this means that signers can produce signatures for their whole class of keys, but they cannot sign for a different class (because of unforgeability) and they are able to hide the class to which the signature belongs, i.e., to hide their own identity in the signature (because of class hiding). This primitive is motivated by (structure-preserving) signatures on equivalence classes [HS14] (SPS-EQ), where relations are defined for the message space, instead of the key space. Both notions are complementary, in the sense that we can use SPS-EQ to *certify* the verification key of an SFPK scheme if the respective equivalence relations are compatible.

Signatures with flexible public key are especially useful in applications where there is a (possibly pre-defined) set of known verification keys and a verifier only needs to know that the creator of a given signature was part of that set. Indeed, upon reading the first description of the scheme's properties, what should come to mind immediately is the setting of group signatures [Cv91] and to some extent ring signatures [RST01] where the group is chosen at signing time and considered a part of the signature. In both settings, however, we need a way to tie a random representative of a key to its corresponding key in the ring or group to ensure unforgeability of the resulting group or ring signature.

The basic idea how to build a group signature scheme from signatures with flexible public key is to combine them with an equally re-randomizable certificate on the signing key. Such a certificate is created through structure-preserving signatures on equivalence classes by the group manager on the members' verification key. A group signature is then produced by signing the message under a fresh representative of the flexible public key and tying that signature to the group by also providing a blinded certificate corresponding to the fresh flexible key. This fresh certificate can be generated from the one provided by the group manager. Opening of group signatures is done using the trapdoor that can be used to distinguish if public keys belong to the same equivalence class.

In the case of ring signatures, the certification of keys becomes slightly more complex, since we cannot make any assumption on the presence of a trusted group manager. Therefore, the membership certificate is realized through a perfectly sound non-interactive proof of membership.

The basic principle, however, remains the same, pointing to an elegant, unified approach to both group and ring signatures.

3.1.1 Contributions in this Chapter

This chapter develops a new cryptographic building block, presenting security definitions, concrete instantiations and applications. In short, the contributions are as follows:

Signatures with Flexible Public Key
We present the new primitive of signatures with flexible public keys, which can be seen as a natural counterpart of structure-preserving signatures on equivalence classes, but for the public key space.

Two Instantiations of SFPK
We present two possible instantiations of SFPK, with different characteristics in terms of underlying cryptographic assumptions and trade-offs.

Applications to Ring and Group Signatures
We demonstrate how SFPK can be used to build group and ring signatures. The resulting group and ring signature schemes have smaller (asymptotic and concrete) signature sizes than the previous state-of-the-art schemes also secure in the strongest attacker model, including schemes with non-standard assumptions.

At time of publication the ring signature construction presented in this chapter was the first to achieve sub-linear signature size, concretely size in $\mathcal{O}(\sqrt{\ell})$ for a ring size of ℓ, without trusted setup and with security under standard assumptions in the strongest security model by Bender, Katz and Morselli [BKM06]. We also show how to efficiently instantiate the scheme using Groth-Sahai proofs and thereby we solve an open problem stated at ASIACRYPT'2017 by Malavolta and Schröder [MS17], namely: *Are there efficient ring signature schemes without trusted setup provably secure under falsifiable assumptions?*

3.2 Chapter Preliminaries

In this section, we introduce preliminaries relevant in the constructions and arguments of this chapter. In particular, we recall the well-known setting of cryptographic groups equipped with a bilinear pairing, we recall the notion of programmable group hash functions, we sketch a non-interactive proof system for pairing product equations, and we recall the concept of structure preserving signatures on equivalence classes.

3.2.1 The Bilinear Group Setting

In cryptography, bilinear pairings had their start as a tool for elliptic curve cryptanalysis [MVO91], only later to become a powerful tool for building sophisticated crypto-systems that rely on the algebraic flexibility afforded by the pairing, beginning with the seminal work by Boneh and Franklin [BF01] on identity-based encryption.

Definition 3.1: Bilinear Map

Let us consider cyclic groups $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ of prime order p. We call $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ a *bilinear map* or simply *pairing* if it is efficiently computable and the following conditions hold:

Bilinearity: For all $(s, t) \in \mathbb{G}_1 \times \mathbb{G}_2$, and for all $a, b \in \mathbb{Z}_p$, we have

$$e(s^a, t^b) = e(s, t)^{a \cdot b}.$$

Non-degeneracy: Let g_1, g_2 be generators of respectively $\mathbb{G}_1$ and $\mathbb{G}_2$ and let $\hat{g} = e(g_1, g_2)$. Then $\hat{g} \neq 1_{\mathbb{G}_T}$ and $\hat{g}$ is a generator of group $\mathbb{G}_T$.

Depending on the choice of groups we say that map e is

- of type I if $\mathbb{G}_1 = \mathbb{G}_2$,
- of type II if $\mathbb{G}_1 \neq \mathbb{G}_2$ and there is an efficiently computable isomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$,
- of type III if no such isomorphism ψ is known.

Unless explicitly stated otherwise we consider only type III pairing groups in this work.

In order to argue about the asymptotic hardness of computational problems in the bilinear group setting we assume the existence of an algorithm that generates groups at a suitable security level.

Definition 3.2: Bilinear Group Generator

A bilinear-group generator is a deterministic polynomial-time algorithm BGGen that on input a security parameter 1^λ returns a bilinear group $\mathsf{BG} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ such that $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$ and $\mathbb{G}_T$ are groups of order p and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map.

We will assume that all parties have access to the bilinear group family that is used to instantiate a certain construction. Bilinear groups with an efficient bilinear-group generator are known to be instantiable e.g. with ordinary elliptic curves such as those introduced by Barreto and Naehrig [BN06].

Invertible Sampling. In our proofs, we will make use of a sampling technique due to Damgård and Nielsen [DN00a]: A *standard sampler* returns a group element X on input coins ω. An *inverted sampler* returns coins ω' on input a group element X. Invertible sampling requires that (X, ω) and (X, ω') are indistinguishably distributed.

Cryptographic Assumptions in the Bilinear Group Setting

We recall assumptions relevant to the instantiations of our constructions presented in this chapter. They are stated relative to bilinear group parameters

$$\mathsf{BG} := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow \mathsf{BGGen}(1^\lambda).$$

We begin with the well-known Diffie-Hellman assumption, given for groups with a type III pairing.

Definition 3.3: Decisional Diffie-Hellman Assumption in $\mathbb{G}_i$

Consider the decisional Diffie-Hellman experiment in group $\mathbb{G}_i$ between a challenger and an adversary $\mathcal{A}$:

```
$\mathsf{DDH}^{\mathcal{A}}_{\mathbb{G}_i}(1^\lambda)$

$x, y \leftarrow \mathbb{Z}_p^*$
$b \leftarrow \{0,1\}$
if $b = 1$ then
    $z \leftarrow \mathbb{Z}_p^*$
else
    $z = x \cdot y$
$b' \leftarrow \mathcal{A}(g_i^x, g_i^y, g_i^z)$
if $b' = b$ then return true
else return false
```

We define the advantage of $\mathcal{A}$ in the above experiment as

$$\mathsf{Adv}^{\mathsf{DDH}-i}_{\mathcal{A}}(\lambda) := \left| \Pr\left[\mathsf{DDH}^{\mathcal{A}}_{\mathbb{G}_i}(1^\lambda) \Rightarrow \mathsf{true}\right] - \frac{1}{2} \right| .$$

The *decisional Diffie-Hellman assumption in* $\mathbb{G}_i$ states that for all PPT adversaries $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathsf{DDH}-i}_{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda) .$$

If the instance were given in both groups, i.e. $(g_1^x, g_1^y, g_1^z, g_2^x, g_2^y, g_2^z)$ then the pairing would allow a simple test $e(g_1^x, g_2^y) = e(g_1^z, g_2)$. This also immediately renders the assumption false in groups with a type I pairing. Similarly, the existence of an isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$ in groups with a type II pairing invalidates the assumption for $\mathbb{G}_2$, but as far as we know not for $\mathbb{G}_1$. To the best of our knowledge the existence of a type III pairing does not invalidate the assumption in either of the two base groups $\mathbb{G}_1$, $\mathbb{G}_2$.

Boneh and Franklin [BF01] introduce an extension of the decisional Diffie-Hellman problem, initially called the Weil decisional Diffie-Hellman problem, that is assumed to remain infeasible in the bilinear setting, even for type I pairings between the groups. In their later work [BF03] it was given its commonly used name today as the bilinear decisional Diffie-Hellman assumption. We restate it for type III pairings as follows:

Definition 3.4: Bilinear Decisional Diffie-Hellman Assumption

Consider the bilinear decisional Diffie-Hellman experiment between a challenger and an adversary $\mathcal{A}$:

```
$\mathsf{BDDH}_{\mathsf{BG}}^{\mathcal{A}}(1^\lambda)$
$u, v, w \leftarrow \mathbb{Z}_p^*$
$b \leftarrow \{0,1\}$
if $b = 1$ then
    $z \leftarrow \mathbb{Z}_p^*$
else
    $z = u \cdot v \cdot w$
$b' \leftarrow \mathcal{A}(g_1^u, g_1^v, g_1^w, g_1^z, g_2^u, g_2^v, g_2^w, g_2^z)$
if $b' = b$ then return true
else return false
```

We define the advantage of $\mathcal{A}$ in the above experiment as

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{BDDH}}(\lambda) := \left| \Pr\left[\mathsf{BDDH}_{\mathsf{BG}}^{\mathcal{A}}(1^\lambda) \Rightarrow \mathsf{true}\right] - \frac{1}{2} \right|.$$

The *bilinear decisional Diffie-Hellman assumption* states that for all PPT adversaries $\mathcal{A}$, we have

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{BDDH}}(\lambda) \leq \mathsf{negl}(\lambda).$$

An extension of the decisional Diffie-Hellman assumption that is assumed to hold even in groups where the decisional Diffie-Hellman assumption does not hold is the decisional linear assumption. We now state the symmetric variant of the decisional linear assumption, where the problem instance is given in both $\mathbb{G}_1$ and $\mathbb{G}_2$. This definition was also used by Ghadafi, Smart and Warinschi [GSW10].

Definition 3.5: Symmetric Decisional Linear Assumption

Consider the symmetric decisional linear experiment between a challenger and an adversary $\mathcal{A}$:

```
$\mathsf{SDLIN}^{\mathcal{A}}_{\mathsf{BG}}(1^\lambda)$
$\varphi, \theta, \alpha, \beta \leftarrow \mathbb{Z}_p^*$
$f_1 := g_1^{\varphi}; h_1 := g_1^{\theta}$
$f_2 := g_2^{\varphi}; h_2 := g_2^{\theta}$
$b \leftarrow \{0,1\}$
**if** $b = 1$ **then**
    $\gamma \leftarrow \mathbb{Z}_p^*$
**else**
    $\gamma = \alpha \cdot \beta$
$b' \leftarrow \mathcal{A}(f_1, h_1, f_1^{\alpha}, h_1^{\beta}, f_2, h_2, f_2^{\alpha}, h_2^{\beta}, g_1^{\gamma}, g_2^{\gamma})$
**if** $b' = b$ **then** **return** true
**else return** false
```

We define the advantage of $\mathcal{A}$ in the above experiment as

$$\mathsf{Adv}^{\mathsf{SDLIN}}_{\mathcal{A}}(\lambda) := \left| \Pr\left[\mathsf{SDLIN}^{\mathcal{A}}_{\mathsf{BG}}(1^\lambda) \Rightarrow \mathsf{true}\right] - \frac{1}{2} \right| .$$

The *symmetric decisional linear assumption* states that for all PPT adversaries $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathsf{SDLIN}}_{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda) .$$

3.2.2 Programmable Hash Functions

Programmable hash functions, introduced by Hofheinz and Kiltz [HK08] present a way to hash into groups with limited programmability. To formally define such functions we first define so-called *group hash functions* into a group $\mathbb{G}$.

Definition 3.6: Group Hash Function into $\mathbb{G}$

Let $\ell \in \mathbb{N}$ be the input length of the hash function.

$\mathsf{Gen}(1^\lambda) \to K$
: Takes as input a security parameter 1^λ. Outputs a hash key K.

$\mathsf{Eval}(K, x) \to r \in \mathbb{G}$
: Takes as input a hash key K and a bit string $x \in \{0,1\}^\ell$. Outputs a group element in $\mathbb{G}$.

Definition 3.7: Programmable Hash Function

A group hash function into $\mathbb{G}$ is called (m, n, γ, δ)-*programmable* if there are polynomial time algorithms PHF.tdGen and PHF.tdEval such that:

- For any $g, h \in \mathbb{G}$ the trapdoor algorithm

$$(K', t) \leftarrow \mathsf{PHF.tdGen}(1^\lambda, g, h)$$

  outputs a key K' and trapdoor t. Moreover, for every $x \in \{0,1\}^\ell$ we have

$$(a_x, b_x) \leftarrow \mathsf{PHF.tdEval}(t, x),$$

  such that $\mathsf{PHF.Eval}(K', x) = g^{a_x} h^{b_x}$.

- For all $g, h \in \mathbb{G}$ and for

$$\begin{aligned}(K', t) &\leftarrow \mathsf{PHF.tdGen}(1^\lambda, g, h) \text{ and}\\ K &\leftarrow \mathsf{PHF.Gen}(1^\lambda),\end{aligned}$$

  the keys K and K' are statistically γ-close.

- For all $g, h \in \mathbb{G}$ and all possible keys K' from the range of $\mathsf{PHF.tdGen}(1^\lambda, g, h)$, for all $x_1, \ldots, x_m, z_1, \ldots, z_n \in \{0,1\}^\ell$ such that $x_i \neq z_j$ for any i, j and for the corresponding

$$\begin{aligned}(a_{x_i}, b_{x_i}) &\leftarrow \mathsf{PHF.tdEval}(t, x_i) \text{ and}\\ (a_{z_i}, b_{z_i}) &\leftarrow \mathsf{PHF.tdEval}(t, z_i),\end{aligned}$$

  we have

$$\Pr[a_{x_1} = \cdots = a_{x_m} = 0 \;\wedge\; a_{z_1} = \cdots = a_{z_n} \neq 0] \geq \delta,$$

  where the probability is over trapdoor t that was generated with key K'.

An example of a programmable hash function is the following construction based on the one due to Waters [Wat05] and modified for type II and type III pairings by Chatterjee and Menezes [CM11].

Construction 3.1: Waters Group Hash Function

```
PHF.Gen($1^\lambda$)
  for $i \in [\ell]_0$ do
    $h_i \leftarrow \mathbb{G}$
  $K := (h_0, \ldots, h_\ell)$
  return $K$
```

```
PHF.Eval($K, x$)
  parse $K = (h_0, \ldots, h_\ell) \in \mathbb{G}^{\ell+1}$
  $r := h_0 \cdot \prod_{i=1}^{\ell} h_i^{x_i}$
  return $r$
```

We recall the following result about the construction above, due to Hofheinz and Kiltz[HK08, Theorem 4].

Lemma 3.1: Construction 3.1 is $(1, q, 0, 1/8 \cdot (\ell + 1) \cdot q)$-programmable

For any fixed $q = \mathsf{poly}(\lambda)$ construction 3.1 is a $(1, q, 0, 1/8 \cdot (\ell + 1) \cdot q)$-programmable group hash function.

Unless mentioned otherwise, we will always instantiate the programmable hash function using the Waters function and use input length $\ell = \lambda$.

3.2.3 Non-interactive Proof Systems for Pairing Product Equations

We have introduced the general formalism of non-interactive proof systems in section 2.2.3. In this chapter, the constructions we design make use of a specific instantiation of a proof system for statements in the language of *pairing product equations* in a bilinear group setting. Namely, we recall the framework of pairing product equations that is used for the languages of the Groth-Sahai proof system [GS08]. For constants $A_i \in \mathbb{G}_1$, $B_i \in \mathbb{G}_2$, $t_T \in \mathbb{G}_T$, $\gamma_{ij} \in \mathbb{Z}_p$ which are either publicly known or part of the statement, and witnesses $X_i \in \mathbb{G}_1$, $Y_i \in \mathbb{G}_2$ given as commitments, we can prove statements of the form:

$$\prod_{i=1}^{n} e(A_i, Y_i) \cdot \prod_{i=1}^{m} e(X_i, B_i) \cdot \prod_{j=1}^{m} \prod_{i=1}^{n} e(X_i, Y_i)^{\gamma_{ij}} = t_T.$$

For our constructions of privacy-preserving signatures we require a proof system for pairing product equations which remains perfectly sound even if there is no trusted third party that could perform an initial setup phase to generate a common reference string. We will construct such a proof system using two underlying proof systems:

- Let $\mathsf{NIP}_{\mathsf{PPE}}$ be the proof system for pairing product equations given by Ghadafi, Smart and Warinschi [GSW10]. This proof system is perfectly sound in the common reference string model under the symmetric decisional linear assumption.
- Let $\mathsf{NIP}_{\mathsf{DLIN}}$ be the proof system due to Groth, Ostrovsky and Sahai [GOS06] which is perfectly sound and perfectly witness-indistinguishable without any trusted setup. Using this proof system one can show that given tuples T_1, T_2 as a statement, at least one of T_1 and T_2 is a valid DLIN tuple. [1]

The soundness of $\mathsf{NIP}_{\mathsf{PPE}}$ hinges on the correctness of the common reference string, which must be a valid DLIN tuple for the proof to be sound. The crucial idea is now that we can use $\mathsf{NIP}_{\mathsf{DLIN}}$ to enforce a correct common reference string generation even for a malicious party by letting the prover generate two common reference strings for $\mathsf{NIP}_{\mathsf{PPE}}$, at least one of which must be valid for the proof in $\mathsf{NIP}_{\mathsf{DLIN}}$ to be valid. Since at least one is a valid DLIN tuple and one of two proofs in $\mathsf{NIP}_{\mathsf{PPE}}$ uses the valid common reference string, our proof system is perfectly sound even if the setup phase is executed by an untrusted party.

We cannot hope for more than computational witness indistinguishability for the resulting proof system, since we are making the common reference strings of the underlying proof systems $\mathsf{NIP}_{\mathsf{PPE}}$ part of our own proof, but this is still sufficient for our applications of the proof system.

The full scheme is presented in construction 3.2.

Construction 3.2: Proof System for Pairing Product Equations

```
NIP.Prove(x, w)

$r, s \leftarrow \mathbb{Z}_p^*$
$\mathsf{crs}_1 := (f_1, f_2, h_1, h_2, \ldots) \leftarrow \mathsf{NIP}_{\mathsf{PPE}}.\mathsf{Setup}(1^\lambda)$
$\mathsf{crs}_2 := (f_1, f_2, h_1, h_2, f_1^r, f_2^r, h_1^s, h_2^s, g_1^{r+s}, g_2^{r+s})$
$\pi_{\mathsf{DLIN}} \leftarrow \mathsf{NIP}_{\mathsf{DLIN}}.\mathsf{Prove}((\mathsf{crs}_1, \mathsf{crs}_2), (r, s))$
$\pi_1 \leftarrow \mathsf{NIP}_{\mathsf{PPE}}.\mathsf{Prove}(\mathsf{crs}_1, \mathsf{x}, \mathsf{w})$
$\pi_2 \leftarrow \mathsf{NIP}_{\mathsf{PPE}}.\mathsf{Prove}(\mathsf{crs}_2, \mathsf{x}, \mathsf{w})$
return $\pi := (\mathsf{crs}_1, \mathsf{crs}_2, \pi_{\mathsf{DLIN}}, \pi_1, \pi_2)$
```

[1] The results were shown for type I pairings but the proof itself is only given as elements in $\mathbb{G}_2$. Moreover, our variant of the DLIN assumption gives the elements in both groups. Thus, we can apply the same steps as in [GOS06]. The size of such a proof is 6 elements in $\mathbb{G}_2$.

$\mathsf{NIP.Verify}(\mathsf{x}, \pi)$

```
parse $\pi = (\mathsf{crs}_1, \mathsf{crs}_2, \pi_{\mathsf{DLIN}}, \pi_1, \pi_2)$
if $(\mathsf{NIP}_{\mathsf{PPE}}.\mathsf{Verify}(\mathsf{crs}_1, \mathsf{x}, \pi_1) = \mathsf{true}$ or
   $\mathsf{NIP}_{\mathsf{PPE}}.\mathsf{Verify}(\mathsf{crs}_2, \mathsf{x}, \pi_2) = \mathsf{true})$ and
   $\mathsf{NIP}_{\mathsf{DLIN}}.\mathsf{Verify}((\mathsf{crs}_1, \mathsf{crs}_2), \pi_{\mathsf{DLIN}}) = \mathsf{true}$ then return $\mathsf{true}$
else return $\mathsf{false}$
```

Theorem 3.2: Perfect Soundness of Construction 3.2

Construction 3.2 is perfectly sound without trusted setup if $\mathsf{NIP}_{\mathsf{PPE}}$ is perfectly sound in the common reference string model and $\mathsf{NIP}_{\mathsf{DLIN}}$ is perfectly sound.

Proof (Sketch). Because $\mathsf{NIP}_{\mathsf{DLIN}}$ is perfectly sound $\mathsf{NIP}_{\mathsf{DLIN}}.\mathsf{Verify}((\mathsf{crs}_1, \mathsf{crs}_2), \pi_{\mathsf{DLIN}}) =$ true means that at least one of crs_1 and crs_2 is a valid DLIN tuple. It follows from the perfect soundness of $\mathsf{NIP}_{\mathsf{PPE}}$ that at least one of π_1 and π_2 is a perfectly sound proof for statement x. Thus, statement x must be true. □

Theorem 3.3: Computational Witness Indistinguishability of Construction 3.2

Construction 3.2 is computationally witness-indistinguishable if $\mathsf{NIP}_{\mathsf{PPE}}$ is perfectly witness-indistinguishable in the common reference string model.

Proof (Sketch). Because the proof system for pairing product equations is witness-indistinguishable, we change the witness we use in proof π_1. Note that this change may include the change of crs_1 to a non-DLIN tuple but the proof π_{DLIN} is still valid because crs_2 is a DLIN tuple. Next we replace crs_1 with crs_2 and use $\mathsf{Setup}_{\mathsf{PPE}}$ to compute crs_2. Finally, we change the witness used to compute π_2. □

3.2.4 Structure-Preserving Signatures on Equivalence Classes

The concept of structure-preserving signatures on equivalence classes (SPS-EQ) was first introduced by Hanser and Slamanig [HS14]. This work was further extended by Fuchsbauer, Hanser and Slamanig in [FHS14] and [FHS15].

The core idea of SPS-EQ is that messages live in a system of equivalence classes induced by equivalence relation $\mathcal{R}$. By signing a message, i.e. one representative of a class, the signer conceptually provides a signature for all the elements in that same equivalence class.

This is made clear by the existence of a procedure $\mathsf{SPS.Move}(\mathsf{vk}_{\mathsf{SPS}}, m, \sigma_{\mathsf{EQ}}, \delta)$ that can be used to change a given signature on representative m to a different representative without knowledge of the signing key, allowing one signature on any representative of the class to cover the whole class, in effect. We thus require that there exists a set of randomizers for the equivalence relation $\Delta_{\mathcal{R}}$, which can be used to move around between different representatives of the same class using the $\mathsf{SPS.Move}$ algorithm.

Structure-preserving signatures on equivalence classes are formally defined as follows:

Definition 3.8: Structure-preserving Signatures on Equivalence Classes

$\mathsf{Setup}(1^\lambda) \to \mathsf{pp}_{\mathsf{SPS}}$
: Takes as input a security parameter 1^λ. Outputs public parameters $\mathsf{pp}_{\mathsf{SPS}}$.

$\mathsf{KeyGen}(\mathsf{pp}_{\mathsf{SPS}}) \to (\mathsf{sk}_{\mathsf{SPS}}, \mathsf{vk}_{\mathsf{SPS}})$
: Takes as input the public parameters $\mathsf{pp}_{\mathsf{SPS}}$. Outputs a pair of signing and verification keys $(\mathsf{sk}_{\mathsf{SPS}}, \mathsf{vk}_{\mathsf{SPS}})$.

$\mathsf{Sign}(\mathsf{sk}_{\mathsf{SPS}}, m) \to \sigma_{\mathsf{EQ}}$
: Takes as input a message $m \in \mathcal{M}$ and signing key $\mathsf{sk}_{\mathsf{SPS}}$. Outputs a signature σ_{EQ}.

$\mathsf{Verify}(\mathsf{vk}_{\mathsf{SPS}}, m, \sigma_{\mathsf{EQ}}) \to r \in \{\mathsf{true}, \mathsf{false}\}$
: Takes as input a verification key $\mathsf{vk}_{\mathsf{SPS}}$, a message m, and signature σ_{EQ}. Outputs either true or false.

$\mathsf{Move}(\mathsf{vk}_{\mathsf{SPS}}, m, \sigma_{\mathsf{EQ}}, \delta) \to (m', \sigma'_{\mathsf{EQ}})$
: Takes as input a verification key $\mathsf{vk}_{\mathsf{SPS}}$, a message m, a signature σ_{EQ}, and randomizer $\delta \in \Delta_{\mathcal{R}}$. Outputs a message-signature pair $(m', \sigma'_{\mathsf{EQ}})$.

$\mathsf{ValKey}(\mathsf{sk}_{\mathsf{SPS}}, \mathsf{vk}_{\mathsf{SPS}}) \to r \in \{\mathsf{true}, \mathsf{false}\}$
: Takes as input a signing key $\mathsf{sk}_{\mathsf{SPS}}$ and verification key $\mathsf{vk}_{\mathsf{SPS}}$. Outputs either true or false.

An SPS-EQ scheme for equivalence relation $\mathcal{R}$ is *correct*, if for all $\lambda \in \mathbb{N}$, all $\mathsf{pp}_{\mathsf{SPS}} \leftarrow \mathsf{Setup}(1^\lambda)$ and $(\mathsf{sk}_{\mathsf{SPS}}, \mathsf{vk}_{\mathsf{SPS}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}_{\mathsf{SPS}})$, all messages m and all randomizers $\delta \in \Delta_{\mathcal{R}}$, we have:

- $\mathsf{ValKey}(\mathsf{sk}_{\mathsf{SPS}}, \mathsf{vk}_{\mathsf{SPS}}) = \mathsf{true}$,
- $\Pr[\mathsf{Verify}(\mathsf{vk}_{\mathsf{SPS}}, m, \mathsf{Sign}(\mathsf{sk}_{\mathsf{SPS}}, m)) = \mathsf{true}] = 1$,
- and $\Pr[\mathsf{Verify}(\mathsf{vk}_{\mathsf{SPS}}, \mathsf{Move}(\mathsf{vk}_{\mathsf{SPS}}, m, \mathsf{Sign}(\mathsf{sk}_{\mathsf{SPS}}, m), \delta)) = \mathsf{true}] = 1$.

The original work [HS14] defines two security notions for SPS-EQ namely unforgeability under chosen-message attacks and class hiding. We will only require the original notion of unforgeability and a stronger notion of class hiding, that we recall shortly.

Definition 3.9: EUF-CMA for SPS-EQ

For any structure preserving signature scheme on equivalence classes SPS consider the unforgeability under chosen message attacks experiment EQ-EUF-CMA between a challenger and an adversary $\mathcal{A}$:

```
EQ-EUF-CMA$^{\mathcal{A}}_{\mathsf{SPS}}(1^\lambda)$

pp$_{\mathsf{SPS}}$ ← SPS.Setup$(1^\lambda)$
(sk$_{\mathsf{SPS}}$, vk$_{\mathsf{SPS}}$) ← SPS.KeyGen(pp$_{\mathsf{SPS}}$)
$(m^*, \sigma^*) \leftarrow \mathcal{A}(\mathsf{vk}_{\mathsf{SPS}})$
if $\forall m \in \mathbf{Q} : (m^*, m) \notin \mathcal{R}$ and
   SPS.Verify(vk$_{\mathsf{SPS}}$, $m^*, \sigma^*$) = true
 then return true
else return false
- - - - - - - - - - - - - - - - - - - -
$\mathcal{A}$ may invoke OSign at any point
during its runtime.
```

```
OSign$(m)$

$\mathbf{Q} := \mathbf{Q} \cup \{m\}$
return SPS.Sign(sk$_{\mathsf{SPS}}$, $m$)
```

We define the advantage of $\mathcal{A}$ in this experiment as

$$\mathsf{Adv}^{\mathsf{EQ\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{SPS}}(\lambda) := \Pr\left[\mathsf{EQ\text{-}EUF\text{-}CMA}^{\mathcal{A}}_{\mathsf{SPS}}(1^\lambda) \Rightarrow \mathsf{true}\right].$$

A signature scheme on equivalence classes SPS is *unforgeable under chosen-message attacks* if for any PPT adversary $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathsf{EQ\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{SPS}}(\lambda) \leq \mathsf{negl}(\lambda).$$

Possible equivalence relations. To the best of our knowledge, all known instantiations of SPS-EQ allow signing messages from the space $(\mathbb{G}^*)^\ell$, for some $\ell > 1$, and cyclic group $\mathbb{G}$ and consider the following equivalence relation $\mathcal{R}_{\mathsf{exp}}$: given two messages

$$m = (m_1, \ldots, m_\ell) \text{ and}$$
$$m' = (m'_1, \ldots, m'_\ell),$$

we say that m and m' are equivalent (denoted by $m \approx_{\mathsf{exp}} m'$) if there exists a scalar $r \in \mathbb{Z}_p^*$, such that for all $i \in [\ell]$ we have

$$m_i{}^r = m_i'.$$

The set of randomizers in this relation is then also the set of non-zero scalars $\Delta_{\mathsf{exp}} = \mathbb{Z}_p^*$ and changing the representative amounts to component-wise exponentiation with the randomizer, which we will, in slight abuse of notation, denote as

$$m^\delta = \left(m_1{}^\delta, \ldots, m_\ell{}^\delta\right)$$

for a message $m \in (\mathbb{G}^*)^\ell$ and randomizer $\delta \in \mathbb{Z}_p^*$.

Once we restrict ourselves to constructions where the equivalence relation is $\mathcal{R}_{\mathsf{exp}}$ we can consider stronger notions of security that are specific to the group framework that comes with that relation.

A stronger notion of class hiding, called perfect adaptation of signatures, was proposed by Fuchsbauer et al. in [FHS15]. Informally, this definition states that signatures received by changing the representative of the class and new signatures for the representative are identically distributed. In our schemes we will only use this stronger notion.

Definition 3.10: Perfect Adaptation of Signatures

A structure preserving signature scheme on equivalence classes SPS for $\mathcal{R}_{\mathsf{exp}}$ on $(\mathbb{G}^*)^\ell$ has *perfect adaption of signatures* if for all $m \in (\mathbb{G}_1^*)^\ell$, all $(\mathsf{sk}_{\mathsf{SPS}}, \mathsf{vk}_{\mathsf{SPS}})$ such that $\mathsf{SPS.ValKey}(\mathsf{sk}_{\mathsf{SPS}}, \mathsf{vk}_{\mathsf{SPS}}) = \mathsf{true}$ and all $\delta \in \mathbb{Z}_p^*$ and σ_{EQ} such that $\mathsf{SPS.Verify}(\mathsf{vk}_{\mathsf{SPS}}, m, \sigma_{\mathsf{EQ}}) = \mathsf{true}$, the distributions of

$$(m^\delta, \mathsf{SPS.Sign}(\mathsf{sk}_{\mathsf{SPS}}, m^\delta)) \text{ and } \mathsf{SPS.Move}(\mathsf{vk}_{\mathsf{SPS}}, m, \sigma_{\mathsf{EQ}}, \delta)$$

are identical.

Fuchsbauer and Gay [FG18] recently introduced a weaker version of unforgeability called unforgeability under chosen-open-message attacks, which restricts the adversary's signing queries to messages where it knows all exponents.

Definition 3.11: EUF-CoMA for SPS-EQ

For any structure preserving signature scheme on equivalence classes SPS consider the unforgeability under chosen open message attacks experiment EQ-EUF-CoMA between a challenger and an adversary $\mathcal{A}$:

$\mathsf{EQ\text{-}EUF\text{-}CoMA}^{\mathcal{A}}_{\mathsf{SPS}}(1^\lambda)$

```
$\mathsf{pp}_{\mathsf{SPS}} \leftarrow \mathsf{SPS.Setup}(1^\lambda)$
$(\mathsf{sk}_{\mathsf{SPS}}, \mathsf{vk}_{\mathsf{SPS}}) \leftarrow \mathsf{SPS.KeyGen}(\mathsf{pp}_{\mathsf{SPS}})$
$(m^*, \sigma^*) \leftarrow \mathcal{A}(\mathsf{vk}_{\mathsf{SPS}})$
if $\forall m \in \mathbf{Q} : m^* \not\approx_{\mathsf{exp}} m$ and
  $\mathsf{SPS.Verify}(\mathsf{vk}_{\mathsf{SPS}}, m^*, \sigma^*) = \mathsf{true}$
 then return true
else return false
```

$\mathcal{A}$ may invoke $\mathsf{OSign}_{\mathsf{open}}$ at any point during its runtime.

$\mathsf{OSign}_{\mathsf{open}}(e_1, \ldots, e_\ell)$

```
$m := (g_1^{e_1}, \ldots, g_1^{e_\ell})$
$\mathbf{Q} := \mathbf{Q} \cup \{(m, (e_1, \ldots, e_\ell))\}$
return $\mathsf{Sign}(\mathsf{sk}_{\mathsf{SPS}}, m)$
```

We define the advantage of $\mathcal{A}$ in this experiment as

$$\mathsf{Adv}^{\mathsf{EQ\text{-}EUF\text{-}CoMA}}_{\mathcal{A},\mathsf{SPS}}(\lambda) := \Pr\left[\mathsf{EQ\text{-}EUF\text{-}CoMA}^{\mathcal{A}}_{\mathsf{SPS}}(1^\lambda) \Rightarrow \mathsf{true}\right].$$

A signature scheme on equivalence classes SPS is *unforgeable under chosen-message attacks* if for any PPT adversary $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathsf{EQ\text{-}EUF\text{-}CoMA}}_{\mathcal{A},\mathsf{SPS}}(\lambda) \leq \mathsf{negl}(\lambda).$$

3.3 Signatures with Flexible Public Keys

In this section we introduce the main contribution of this chapter, the concept of signatures with flexible public keys. We begin by motivating the idea behind our primitive.

In the notion of existential unforgeability of regular digital signatures, the adversary must return a signature valid under the verification key given to it by the challenger. Imagine now that we allow a more flexible forgery. The adversary could instead return a signature that is valid under a verification key that is in some relation $\mathcal{R}$ to the verification key chosen by the challenger. In our new primitive, this relation induces a system of randomizable equivalence classes on the set of possible public keys. This is analogous to the system of equivalence classes that forms the message space of SPS-EQ signatures. A given public key, along with the corresponding secret key can be transformed to a different representative in the same class.

A technical detail of the described notion is that there may be other ways of obtaining a new representative, hence the forgery on the challenge equivalence class is valid as long as the relation holds, even without knowledge of the explicit randomness that leads to the given transformation.

In addition to this relaxed unforgeability, we require a property named class hiding –reminiscent of the related property for SPS-EQ– that requires that it should not be feasible, in absence of the concrete transformation randomness, to determine whether a given verification key belongs to one class or another. This property should hold even for an adversary which has access to the randomness used to create the key pairs in question. Class Hiding is essential in the applications of signatures with flexible public keys to privacy-preserving signatures.

Observe that an apparent conflict arises between class hiding of a proposed scheme and our ability to prove unforgeability as described above, because the challenger needs a way to efficiently determine whether a forgery is valid, even if no transformation randomness is given. The conflict is resolved by requiring the existence of a trapdoor key generation algorithm tdGen which outputs a key pair (fsk, fvk) and a class trapdoor td for the class the key pair is in. The trapdoor allows the challenger to check whether a given key is in the same class as fvk, even if doing so efficiently is otherwise assumed difficult. Since we require that the keys generated using the trapdoor key generation and the regular key generation are distributed identically, unforgeability results with respect to the former also hold with respect to the latter.

Definition 3.12: Signatures with Flexible Public Key

$\mathsf{KeyGen}(1^\lambda) \to (\mathsf{fsk}, \mathsf{fvk})$

Takes as input a security parameter 1^λ. Outputs a pair $(\mathsf{fsk}, \mathsf{fvk})$ of signing and verification keys.

$\mathsf{tdGen}(1^\lambda) \to (\mathsf{fsk}, \mathsf{fvk}, \mathsf{td})$

Takes as input a security parameter 1^λ. Outputs a pair $(\mathsf{fsk}, \mathsf{fvk})$ of signing and verification keys, as well as a trapdoor td.

$\mathsf{Sign}(\mathsf{fsk}, m) \to \sigma$

Takes as input a message $m \in \mathcal{M}$ and a signing key fsk. Outputs a signature σ.

$\mathsf{Verify}(\mathsf{fvk}, m, \sigma) \to r \in \{\mathsf{true}, \mathsf{false}\}$

Takes as input a message m, a signature σ and verification key fvk. Outputs either true or false.

$\mathsf{Check}(\mathsf{td}, \mathsf{fvk}) \to r \in \{\mathsf{true}, \mathsf{false}\}$

Takes as input a trapdoor td and a verification key fvk. Outputs either true or false

$\mathsf{MoveVk}(\mathsf{fvk}, \delta) \rightarrow \mathsf{fvk}'$
: Takes as input a verification key fvk and randomizer $\delta \in \Delta_{\mathcal{R}}$. Outputs a verification key fvk'.

$\mathsf{MoveSk}(\mathsf{fsk}, \delta) \rightarrow \mathsf{fsk}'$
: Takes as input a signing key fsk and randomizer $\delta \in \Delta_{\mathcal{R}}$. Outputs a signing key fsk'.

A signature scheme with flexible public key is *correct* if for all $1^\lambda \in \mathbb{N}$ the following conditions hold:

- The distribution of key pairs produced by KeyGen and tdGen is identical.
- For all key pairs
$$(\mathsf{fsk}, \mathsf{fvk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$$
and all messages $m \in \mathcal{M}$ we have
$$\mathsf{Verify}(\mathsf{fvk}, m, \mathsf{Sign}(\mathsf{fsk}, m)) = \mathsf{true} \text{ and}$$
$$\mathsf{Verify}(\mathsf{fvk}', m, \mathsf{Sign}(\mathsf{fsk}', m)) = \mathsf{true},$$
where $\mathsf{MoveVk}(\mathsf{fvk}, \delta) = \mathsf{fvk}'$ and $\mathsf{MoveSk}(\mathsf{fsk}, \delta) = \mathsf{fsk}'$ for any randomizer $\delta \in \Delta_{\mathcal{R}}$.
- For all $(\mathsf{fsk}, \mathsf{fvk}, \mathsf{td}) \leftarrow \mathsf{tdGen}(1^\lambda)$ and all fvk' we have
$$\mathsf{Check}(\mathsf{td}, \mathsf{fvk}') = \mathsf{true}$$
if and only if $\mathsf{fvk}' \approx_{\mathcal{R}} \mathsf{fvk}$.

Remark

As a matter of convenient notation, it will be useful to consider the joint transformation of representatives of signing and verification keys of an SFPK scheme using the same randomizer δ. In this case we write
$$(\mathsf{fsk}', \mathsf{fvk}') \leftarrow \mathsf{SFPK.MoveKeys}(\mathsf{fsk}, \mathsf{fvk}, \delta)$$
instead of sequential calls to $\mathsf{SFPK.MoveSk}(\mathsf{fsk}, \delta)$ and $\mathsf{SFPK.MoveVk}(\mathsf{fvk}, \delta)$.

3.3.1 Security of Signatures with Flexible Public Keys

Class Hiding

We require, informally, that an adversary should not be able to link verification keys to their base representatives after they have been transformed with SFPK.MoveVk. This should hold even if the adversary has access to the possible previous base representatives and their corresponding signing keys, as well as the possibility to obtain signatures under the new signing key.

Definition 3.13: Class Hiding with Key Corruption

For any signature scheme with flexible public keys SFPK with relation $\mathcal{R}$, consider the class hiding experiment with key corruption Class-Hiding-Keys between a challenger and an adversary $\mathcal{A}$:

```
Class-Hiding-Keys$^{\mathcal{A}}_{\mathsf{SFPK},\mathcal{R}}(1^\lambda)$

$(\mathsf{fsk}_0, \mathsf{fvk}_0) \leftarrow \mathsf{SFPK.KeyGen}(1^\lambda)$
$(\mathsf{fsk}_1, \mathsf{fvk}_1) \leftarrow \mathsf{SFPK.KeyGen}(1^\lambda)$
$b \leftarrow \{0,1\}$
$\delta \leftarrow \Delta_{\mathcal{R}}$
$\mathsf{fsk}' \leftarrow \mathsf{SFPK.MoveSk}(\mathsf{fsk}_b, \delta)$
$\mathsf{fvk}' \leftarrow \mathsf{SFPK.MoveVk}(\mathsf{fvk}_b, \delta)$
$\hat{b} \leftarrow \mathcal{A}((\mathsf{fsk}_0, \mathsf{fvk}_0), (\mathsf{fsk}_1, \mathsf{fvk}_1), \mathsf{fvk}')$
if $b = \hat{b}$ then return true
else return false
--------------------------------
$\mathcal{A}$ may query OSign at any point
during its runtime.
```

```
OSign$(m)$

return $\mathsf{SFPK.Sign}(\mathsf{fsk}', m)$
```

We define the advantage of $\mathcal{A}$ in this experiment as

$$\mathsf{Adv}^{\mathsf{Class\text{-}Hiding\text{-}Keys}}_{\mathcal{A},\mathsf{SFPK}}(\lambda) := \left| \Pr\left[\mathsf{Class\text{-}Hiding\text{-}Keys}^{\mathcal{A}}_{\mathsf{SFPK}}(1^\lambda) \Rightarrow \mathsf{true}\right] - \frac{1}{2} \right| .$$

A SFPK is *class hiding with key corruption* if for any PPT adversary $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathsf{Class\text{-}Hiding\text{-}Keys}}_{\mathcal{A},\mathsf{SFPK}}(\lambda) \leq \mathsf{negl}(\lambda) .$$

We can consider an even stronger notion of class hiding, if we allow the adversary access to the randomness that was used in generation of the base keys.

Definition 3.14: Full Class Hiding

For any signature scheme with flexible public keys SFPK with relation $\mathcal{R}$, consider the class hiding experiment Class-Hiding between a challenger and an adversary $\mathcal{A}$:

```
$\mathsf{Class\text{-}Hiding}^{\mathcal{A}}_{\mathsf{SFPK},\mathcal{R}}(1^\lambda)$
$(\mathsf{fsk}_0, \mathsf{fvk}_0) \leftarrow \mathsf{SFPK.KeyGen}(1^\lambda; \omega_0)$
$(\mathsf{fsk}_1, \mathsf{fvk}_1) \leftarrow \mathsf{SFPK.KeyGen}(1^\lambda; \omega_1)$
$b \leftarrow \{0,1\}$
$\delta \leftarrow \Delta_{\mathcal{R}}$
$\mathsf{fsk}' \leftarrow \mathsf{SFPK.MoveSk}(\mathsf{fsk}_b, \delta)$
$\mathsf{fvk}' \leftarrow \mathsf{SFPK.MoveVk}(\mathsf{fvk}_b, \delta)$
$\hat{b} \leftarrow \mathcal{A}(\omega_0, \omega_1, \mathsf{fvk}')$
if $b = \hat{b}$ then return true
else return false
-------------------------------
$\mathcal{A}$ may query OSign at any point during its runtime.
```

```
$\mathsf{OSign}(m)$
return $\mathsf{SFPK.Sign}(\mathsf{fsk}', m)$
```

We define the advantage of $\mathcal{A}$ in this experiment as

$$\mathsf{Adv}^{\mathsf{Class\text{-}Hiding}}_{\mathcal{A},\mathsf{SFPK}}(\lambda) := \left| \Pr\left[\mathsf{Class\text{-}Hiding}^{\mathcal{A}}_{\mathsf{SFPK}}(1^\lambda) \Rightarrow \mathsf{true}\right] - \frac{1}{2} \right| .$$

A SFPK is *class hiding* if for any PPT adversary $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathsf{Class\text{-}Hiding}}_{\mathcal{A},\mathsf{SFPK}}(\lambda) \leq \mathsf{negl}(\lambda) .$$

Unforgeability under Flexible Public Keys

A signature scheme with flexible public keys should provide unforgeability for the entire class represented by a given verification key. To make this a falsifiable notion without harming the class hiding properties, we have to let the challenger use the SFPK.tdGen algorithm. In order to consider the strongest possible notion of unforgeability, we thus also give the adversary access to the class trapdoor and require that forgery should still be infeasible.

Definition 3.15: Existential Unforgeability under Flexible Public Key

For any signature scheme with flexible public keys SFPK with relation $\mathcal{R}$, consider the unforgeability under flexible public keys experiment Flex-Unforgeability between a challenger and an adversary $\mathcal{A}$:

$\mathsf{Flex\text{-}Unforgeability}^{\mathcal{A}}_{\mathsf{SFPK},\mathcal{R}}(1^\lambda)$

```
$\mathbf{Q} := \emptyset$
$(\mathsf{fsk}, \mathsf{fvk}, \mathsf{td}) \leftarrow \mathsf{SFPK.tdGen}(1^\lambda)$
$(\mathsf{fvk}', m^*, \sigma^*) \leftarrow \mathcal{A}(\mathsf{fvk}, \mathsf{td})$
if $(m^*, \cdot) \notin \mathbf{Q}$ and
   $\mathsf{SFPK.Check}(\mathsf{td}, \mathsf{fvk}') = \mathsf{true}$ and
   $\mathsf{SFPK.Verify}(\mathsf{fvk}', m^*, \sigma^*) = \mathsf{true}$
 then return true
else return false
```

$\mathcal{A}$ may query OSign^1 and OSign^2 at any point during its runtime.

$\mathsf{OSign}^1(m)$

```
$\sigma \leftarrow \mathsf{SFPK.Sign}(\mathsf{fsk}, m)$
$\mathbf{Q} := \mathbf{Q} \cup \{(m, \sigma)\}$
return $\sigma$
```

$\mathsf{OSign}^2(m, \delta)$

```
$\mathsf{fsk}' \leftarrow \mathsf{SFPK.MoveSk}(\mathsf{fsk}, \delta)$
$\sigma \leftarrow \mathsf{SFPK.Sign}(\mathsf{fsk}', m)$
$\mathbf{Q} := \mathbf{Q} \cup \{(m, \sigma)\}$
return $\sigma$
```

We define the advantage of $\mathcal{A}$ in this experiment as

$$\mathsf{Adv}^{\mathsf{Flex\text{-}Unforgeability}}_{\mathcal{A},\mathsf{SFPK}}(\lambda) := \Pr\left[\mathsf{Flex\text{-}Unforgeability}^{\mathcal{A}}_{\mathsf{SFPK}}(1^\lambda) \Rightarrow \mathsf{true}\right].$$

A scheme SFPK is *existentially unforgeable with flexible public key under chosen message attack* if for any PPT adversary $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathsf{Flex\text{-}Unforgeability}}_{\mathcal{A},\mathsf{SFPK}}(\lambda) \le \mathsf{negl}(\lambda).$$

Definition 3.16: Strong Existential Unforgeability under Flexible Public Key

A signature scheme with flexible public keys SFPK is *strongly existentially unforgeable with flexible public key under chosen message attack* if for all PPT adversaries $\mathcal{A}$ the advantage $\mathsf{Adv}^{\mathsf{Flex\text{-}Unforgeability}}_{\mathcal{A},\mathsf{SFPK}}(\lambda)$ in the above experiment is negligible in λ, where we replace the condition $(m^*, \cdot) \notin \mathbf{Q}$ with $(m^*, \sigma^*) \notin \mathbf{Q}$.

Compatibility with SPS-EQ. Our primitive can be seen as complementary in functionality to signatures on equivalence classes and indeed the applications we showcase below benefit greatly from this complementarity. However, for there to be a useful interaction between two such schemes, we require that they speak the same language, i.e. that they consider the same system of equivalence classes. We thus define the following useful property between signatures on equivalence classes and signatures with flexible public keys.

Definition 3.17: SPS-EQ/SFPK **Compatibility**

A structure preserving signature scheme on equivalence classes SPS-EQ and a signature scheme with flexible public keys SFPK are called *compatible* if the message space of the former is the same as the key space of the latter and they share the same equivalence relation $\mathcal{R}$.

Canonical Representatives. In some applications it might be required that every equivalence class has a unique representative that can act as a description of the class. We will call such objects the *canonical representatives* of the given classes and further assume that if a scheme has canonical representatives, there is an efficient predicate SFPK.Canonical? to determine whether a verification key is canonical or not.

Definition 3.18: Canonical Representatives

A signature scheme with flexible public keys has *canonical representatives* if, in addition to the interface specified in definition 3.12, there exists an algorithm Canonical? as follows:

$\mathsf{Canonical?}(\mathsf{fvk}) \to r \in \{\mathsf{true}, \mathsf{false}\}$
Takes as input a verification key fvk. Outputs either true or false.

Key Recovery. In the applications we will see later, the verification and signing keys are jointly randomized by the signer using the same randomizer in SFPK.MoveVk and SFPK.MoveSk. However, the SFPK.MoveVk algorithm alone can be executed by a third party given only the verification key and a randomizer δ. Revealing δ to the holder of the signing key allows them to compute the corresponding randomized signing key. A potentially useful property in this case would be a way to avoid interaction during this recovery of the signing key. Allowing the signer to extract the new signing key using only their old signing key would break class hiding, since the attacker in this case has access to the base signing keys. Fortunately, we can instead use the additional trapdoor returned by the SFPK.tdGen algorithm. More formally, we define this optional property as follows.

Definition 3.19: Key Recovery

A signature scheme with flexible public keys SFPK has *recoverable signing keys* if there exists an efficient algorithm $\mathsf{SFPK.Recover}$ such that for all security parameters $\lambda \in \mathbb{N}$, all randomizers $\delta \in \Delta_{\mathcal{R}}$ and all

$$(\mathsf{fsk}, \mathsf{fvk}, \mathsf{td}) \leftarrow \mathsf{SFPK.tdGen}(1^\lambda) \text{ and}$$
$$\mathsf{fvk}' \leftarrow \mathsf{SFPK.MoveVk}(\mathsf{fvk}, \delta)$$

we have

$$\mathsf{SFPK.MoveSk}(\mathsf{fsk}, \delta) = \mathsf{SFPK.Recover}(\mathsf{fsk}, \mathsf{td}, \mathsf{fvk}').$$

3.3.2 SFPK with Setup

In this subsection, we address applications where part of each user's public key is shared with all the other public keys and is precomputed by a trusted third party in a setup phase, e.g. the key used in a programmable hash function. We therefore define an additional algorithm $\mathsf{SFPK.Setup}$ that, given a security parameter, outputs a set of public parameters $\mathsf{pp}_{\mathsf{SFPK}}$. We assume that these parameters are an implicit input to all algorithms of such a scheme. If the $\mathsf{SFPK.KeyGen}$ is independent of $\mathsf{pp}_{\mathsf{SFPK}}$, we say that such a scheme supports *key generation without setup*.

Definition 3.20: SFPK with Public Parameters

A signature scheme with flexible public keys is *with public parameters* if, in addition to the interface specified in definition 3.12, there exists an algorithm Setup as follows:

$\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{pp}_{\mathsf{SFPK}}$
Takes as input a security parameter 1^λ. Outputs public parameters $\mathsf{pp}_{\mathsf{SFPK}}$.

We briefly discuss the implications of a setup phase on the security notions. Usually, we require that the public parameters are generated by an honest and trusted party (i.e. by the challenger in definition 3.14 and definition 3.15). We can additionally consider those notions under maliciously generated parameters $\mathsf{pp}_{\mathsf{SFPK}}$. We call a scheme *class hiding under malicious parameters* if the class hiding definition holds even if in definition 3.14 the adversary is allowed to generate the public parameters $\mathsf{pp}_{\mathsf{SFPK}}$. Similarly, we call an SFPK scheme *unforgeable under malicious parameters* if the unforgeability definition 3.15 holds if $\mathsf{pp}_{\mathsf{SFPK}}$ is generated by the adversary.

3.4 Instantiating SFPK

In this section we present two efficient instantiations of signatures with flexible public keys. Both schemes support a generalized exponentiation relation $\mathcal{R}_{\mathsf{exp}}$ similar to the one we saw earlier for SPS-EQ. To recapitulate, we say that verification keys

$$\mathsf{fvk}_1 = (\mathsf{fvk}_{1,1}, \dots, \mathsf{fvk}_{1,\ell}) \text{ and}$$
$$\mathsf{fvk}_2 = (\mathsf{fvk}_{2,1}, \dots, \mathsf{fvk}_{2,\ell})$$

are equivalent, denoted $\mathsf{fvk}_1 \approx_{\mathsf{exp}} \mathsf{fvk}_2$, if and only if there exists a scalar $\delta \in \mathbb{Z}_p^*$ such that for all $i \in [\ell]$ we have

$$(\mathsf{fvk}_{1,i})^\delta = \mathsf{fvk}_{2,i}.$$

In this general variant, $\mathsf{fvk}_1, \mathsf{fvk}_2$ may contain a mix of elements from different (same order) groups as long as component-wise $\mathsf{fvk}_{1,i}, \mathsf{fvk}_{2,i}$ are in the same group for all i.

Remark

We remark that our construction requires a key homomorphism property in the programmable hash function that is used, namely for all $\delta \in \Delta$ it should hold

$$\mathsf{PHF.Eval}(K_{\mathsf{PHF}}^\delta, m) = \mathsf{PHF.Eval}(K_{\mathsf{PHF}}, m)^\delta.$$

It is easy to see that this property holds e.g. for construction 3.1.

3.4.1 Without Setup

The first instantiation of SFPK we present does not require a trusted setup. In comparison to the construction with setup in the following section, this leads to larger verification keys and a more time-consuming check procedure for representatives.

We assume that in the plain model scheme (i.e. without a common reference string) the verification key contains the implicit security parameter 1^λ and parameters BG. Since the bilinear-group generation algorithm $\mathsf{BGGen}(\lambda)$ is deterministic, it follows that this does not influence the class hiding property or the unforgeability property. Therefore, for readability we omit those parameters.

The first instantiation is based around an instantiation of the Waters group hash function presented in section 3.2.2. The scheme has the key recovery property.

Construction 3.3: SFPK **Without Setup**

```
SFPK.KeyGen($1^\lambda$)
  $K_{\mathsf{PHF}} \leftarrow \mathsf{PHF.Gen}(1^\lambda) \in (\mathbb{G}_1^*)^{\lambda+1}$
  $A, B, C, D, X \leftarrow \mathbb{G}_1^*$
  $y \leftarrow \mathbb{Z}_p^*$
  $t := e(X^y, g_2)$
  $\mathsf{fvk} := (t, A, B, C, D, K_{\mathsf{PHF}})$
  $\mathsf{fsk} := (y, X, \mathsf{fvk})$
  return $(\mathsf{fsk}, \mathsf{fvk})$
```

```
SFPK.tdGen($1^\lambda$)
  $a, x, y \leftarrow \mathbb{Z}_p^*$
  $t := e(g_1^{x \cdot y}, g_2)$
  $\mu_0 := a \cdot x$
  for $i = 1 \ldots (\lambda + 4)$ do
    $\mu_i \leftarrow \mathbb{Z}_p^*$
  $K_{\mathsf{PHF}} := (g_1^{\mu_i})_{i \in \{4,\ldots,\lambda+4\}}$
  $\mathsf{fvk} := (t, g_1^{\mu_0}, g_1^{\mu_1}, g_1^{\mu_2}, g_1^{\mu_3}, K_{\mathsf{PHF}})$
  $\mathsf{fsk} := (y, g_1^x, \mathsf{fvk})$
  $\mathsf{td} := (a, g_2^y, g_2^{\mu_0}, \ldots, g_2^{\mu_{\lambda+4}})$
  return $(\mathsf{fsk}, \mathsf{fvk}, \mathsf{td})$
```

```
SFPK.Sign($\mathsf{fsk}, m$)
  parse $\mathsf{fsk} = (y, X, \mathsf{fvk})$
  parse $\mathsf{fvk} = (t, A, B, C, D, K_{\mathsf{PHF}})$
  $r \leftarrow \mathbb{Z}_p^*$
  $h \leftarrow \mathsf{PHF.Eval}(K_{\mathsf{PHF}}, m)$
  $\sigma := (X^y \cdot h^r, g_1^r, g_2^r)$
  return $\sigma$
```

```
SFPK.Verify($\mathsf{fvk}, m, \sigma$)
  parse $\sigma = (\sigma^1, \sigma^2, \sigma^3)$
  parse $\mathsf{fvk} = (t, A, B, C, D, K_{\mathsf{PHF}})$
  $h \leftarrow \mathsf{PHF.Eval}(K_{\mathsf{PHF}}, m)$
  if $e(\sigma^2, g_2) = e(g_1, \sigma^3)$ and
    $e(\sigma^1, g_2) = t \cdot e\left(h, \sigma^3\right)$
   then return true
  else return false
```

```
SFPK.MoveVk($\mathsf{fvk}, \delta$)
  parse $\mathsf{fvk} = (t, A, B, C, D, K_{\mathsf{PHF}})$
  $\mathsf{fvk}' := (t^\delta, A^\delta, B^\delta, C^\delta, D^\delta, (K_{\mathsf{PHF}})^\delta)$
  return $\mathsf{fvk}'$
```

```
SFPK.MoveSk($\mathsf{fsk}, \delta$)
  parse $\mathsf{fsk} = (y, X, \mathsf{fvk})$
  $\mathsf{fvk}' \leftarrow \mathsf{MoveVk}(\mathsf{fvk}, \delta)$
  $\mathsf{fsk}' := (y, X^\delta, \mathsf{fvk}')$
  return $\mathsf{fsk}'$
```

```
SFPK.Check(td, fvk)

parse fvk = $(t, \mathsf{fvk}_0, \mathsf{fvk}_1, \mathsf{fvk}_2, \mathsf{fvk}_3, \ldots, \mathsf{fvk}_{\lambda+4})$
parse td = $(a, Y_2, \mathsf{td}_0, \ldots, \mathsf{td}_{\lambda+4})$
if $e(\mathsf{fvk}_0^{a^{-1}}, Y_2) = t$ and $\bigwedge_{i=0}^{\lambda+4} \bigwedge_{j=0}^{\lambda+4} e(\mathsf{fvk}_i, \mathsf{td}_j) = e(\mathsf{fvk}_j, \mathsf{fvk}_i)$
 then return true
else return false
```

```
SFPK.Recover(fsk, td, fvk')

parse fsk = $(y, X, \mathsf{fvk})$
parse td = $(a, Y_2, \mathsf{td}_0, \ldots \mathsf{td}_{\lambda+4})$
parse fvk' = $(t', A', B', C', D', K'_{\mathsf{PHF}})$
$X' := A'^{a^{-1}}$
$\mathsf{fsk}' := (y, X', \mathsf{fvk}')$
return fsk'
```

Correctness and Key Recovery. First, observe that SFPK.tdGen, instead of sampling group elements directly, samples in the exponent group and uses the exponents to compute the trapdoor in addition to signing and verification keys. Since the exponents are sampled uniformly at random, the group elements obtained by exponentiating the generator g_1 are equally uniformly distributed in $\mathbb{G}_1$. Thus, the distributions of key pairs generated by SFPK.tdGen and SFPK.KeyGen are identical.

Let $(\mathsf{fsk}, \mathsf{fvk}, \mathsf{td}) \leftarrow \mathsf{SFPK.tdGen}(\lambda)$. We have

$$\begin{aligned}
\mathsf{fvk} &= (t = e(g_1^{x \cdot y}, g_2), g_1^{\mu_0 = a \cdot x}, g_1^{\mu_1}, g_1^{\mu_2}, \ldots, g_1^{\mu_{\lambda+4}}) \\
\mathsf{fsk} &= (y, g_1^x, \mathsf{fvk}) \\
\mathsf{td} &= (a, Y_2 = g_2^y, \mathsf{td}_0, \ldots, \mathsf{td}_{\lambda+4})
\end{aligned}$$

where $\mathsf{td}_i = g_2^{\mu_i}$.

We will now show that checking of representatives is correct.

First, let $\mathsf{fvk}' = (t', \mathsf{fvk}'_0, \mathsf{fvk}'_1, \ldots, \mathsf{fvk}'_{\lambda+4})$, such that $\mathsf{fvk}' \approx_{\mathsf{exp}} \mathsf{fvk}$, i.e. there exists $\delta \in \mathbb{Z}_p^*$ such that $t' = t^\delta$ and $\mathsf{fvk}'_i = \mathsf{fvk}_i^\delta$ for all $i \in \{0, \ldots, \lambda + 4\}$.

Then we have

$$\begin{aligned} e(\mathsf{fvk}_0'^{a^{-1}}, Y_2) &= e(g_1^{ax\delta} \cdot a^{-1}, g_2^y) \\ &= e(g_1^{xy}, g_2)^\delta = t'. \end{aligned}$$

Further, let $i, j \in \{0, \ldots, \lambda + 4\}$. Then

$$\begin{aligned} e(\mathsf{fvk}_i', \mathsf{td}_j) &= e(g_1^{\delta\mu_i}, g_2^{\mu_j}) \\ &= e(g_1^{\delta\mu_j}, g_2^{\mu_i}) = e(\mathsf{fvk}_j', \mathsf{td}_i). \end{aligned}$$

Hence, $\mathsf{SFPK.Check}(\mathsf{td}, \mathsf{fvk}') = \mathsf{true}$. Observe further, that

$$\begin{aligned} \mathsf{SFPK.MoveSk}(\mathsf{fsk}, \delta) &= (y, g_1^{x\delta}, \mathsf{fvk}') \\ &= (y, g_1^{ax\delta \cdot a^{-1}}, \mathsf{fvk}') = \mathsf{SFPK.Recover}(\mathsf{fsk}, \mathsf{td}, \mathsf{fvk}'), \end{aligned}$$

hence key the key recovery algorithm is correct.

Second, let $\hat{\mathsf{fvk}}$ be such that a $\hat{\mathsf{fvk}} \not\approx_{\mathsf{exp}} \mathsf{fvk}$.

Then, for all $\delta \in \mathbb{Z}_p^*$ we have $\hat{t} \neq t^\delta$ or $\hat{\mathsf{fvk}}_i \neq \mathsf{fvk}_i^\delta$ for some $i \in \{0, \ldots, \lambda + 4\}$. Let $\delta \in \mathbb{Z}_p^*$ such that $\hat{\mathsf{fvk}}_0 = \mathsf{fvk}_0^\delta$. Hence, we have $\hat{t} = t^{\delta+\beta} \neq t^\delta$ or $\hat{\mathsf{fvk}}_i = \mathsf{fvk}_i^{\delta+\alpha} \neq \mathsf{fvk}_i^\delta$ for some $i \in \{1, \ldots, \lambda + 4\}$ and $\alpha, \beta \in \mathbb{Z}_p^*$. First, assume the second case. Then

$$\begin{aligned} e(\hat{\mathsf{fvk}}_i, \mathsf{td}_0) &= e(g_1^{\mu_i(\delta+\alpha)}, g_2^{\mu_0}) \\ &= e(g_1^{\mu_i\delta}, g_2^{\mu_0}) \cdot e(g_1^{\mu_i\alpha}, g_2^{\mu_0}) \\ &\neq e(g_1^{\mu_0\delta}, g_2^{\mu_i}) = e(\hat{\mathsf{fvk}}_0, \mathsf{td}_i) \end{aligned}$$

thus failing the second check.

Now assume $\hat{t} = t^{\delta+\beta} \neq t^\delta$, then

$$\begin{aligned} e(\hat{\mathsf{fvk}}_0^{a^{-1}}, Y_2) &= e(g_1^{ax\delta \cdot a^{-1}}, g_2^y) \\ &= e(g_1^{xy}, g_2)^\delta = t^\delta \\ &\neq t^{\delta+\beta} = \hat{t}, \end{aligned}$$

failing the first check. Thus, we can never have $\mathsf{SFPK.Check}(\mathsf{td}, \hat{\mathsf{fvk}}) = \mathsf{true}$ if $\hat{\mathsf{fvk}} \not\approx_{\mathsf{exp}}$ fvk.

For message m consider now a signature

$$\sigma = (X'^y \cdot h^r, g_1^r, g_2^r)$$

under fsk' corresponding to fvk' from above. We have

$$e(g_1^r, g_2) = e(g_1, g_2^r)$$

and

$$\begin{aligned} e(X'^{y} \cdot h^r, g_2) &= e(g_1^{xy\delta} \cdot h^r, g_2) \\ &= e(g_1^{xy\delta}, g_2) \cdot e(h, g_2^r) \\ &= t' \cdot e(h, g_2^r) \end{aligned}$$

Thus, if PHF is correct and compatible with the relation, we have

$$\mathsf{SFPK.Verify}(\mathsf{fvk}', m, \sigma) = \mathsf{true}.$$

Setting $\delta = 1$ shows that verification succeeds for canonical keys as well.

All together, this shows correctness of the construction.

Proof of Unforgeability

Theorem 3.4: Unforgeability of Construction 3.3

Construction 3.3 is existentially unforgeable under flexible public key, assuming the symmetric decisional linear assumption holds and that PHF is $(1, \mathsf{poly}(\lambda))$-programmable.

Proof. In this particular proof we assume that we can re-run PHF.tdGen using the same random coins on a different group, i.e. that we can generate key $K_{\mathsf{PHF}} = (g_1^{\mu_4}, \ldots, g_1^{\mu_{1^\lambda+4}}) \in \mathbb{G}_1^{1^\lambda+1}$ and a corresponding key $K'_{\mathsf{PHF}} = (g_2^{\mu_4}, \ldots, g_2^{\mu_{1^\lambda+4}}) \in \mathbb{G}_2^{1^\lambda+1}$. Note that this means that we make non-blackbox use of the underlying programmable hash function, but this re-running is possible for the hash function we use, i.e. the Waters hash function.

Let $(f_1, h_1, f_1^\alpha, h_1^\beta, f_2, h_2, f_2^\alpha, h_2^\beta, g_1^\gamma, g_2^\gamma)$ be an instance of the decisional linear problem and let $\mathcal{A}$ be a PPT adversary with non-negligible advantage $\mathsf{Adv}_{\mathcal{A},\mathsf{SFPK}}^{\mathsf{Flex\text{-}Unforgeability}}(\lambda)$. We will show an algorithm $\mathcal{R}$ that uses $\mathcal{A}$ to break the above problem instance.

In the first step, the reduction $\mathcal{R}$ prepares the verification key $\mathsf{fvk} = (t, A, B, C, D, K_{\mathsf{PHF}})$ as follows. It sets:

$$\begin{aligned} &X = g_1^\gamma && A = X^a && B = h_1^\beta \\ &C = h_1 && t = e(X, f_2) = e(X^\phi, g_2) && D = f_1^\alpha \end{aligned}$$

and $(K_{\mathsf{PHF}}, \tau_{\mathsf{PHF}}) \leftarrow \mathsf{PHF.tdGen}(1^\lambda, g_1^\gamma, g_1)$. The reduction also prepares the trapdoor $\tau = (a, f_2, f_2^\alpha, h_2^\beta, h_2, K'_{\mathsf{PHF}})$, where to generate K'_{PHF} we re-run the algorithm $\mathsf{PHF.tdGen}(1^\lambda, g_2^\gamma, g_2)$ as discussed above.

Let (m, l) be one of $\mathcal{A}'s$ signing queries. To answer it, $\mathcal{R}$

- chooses random values $t \leftarrow \mathbb{Z}_p^*$,
- it computes $(a_m, b_m) \leftarrow \mathsf{PHF.tdEval}(\tau_{\mathsf{PHF}}, m)$ and aborts if $a_m = 0$,
- it computes $\mathsf{fvk}' \leftarrow \mathsf{SFPK.MoveVk}(\mathsf{fvk}, l)$,
- it computes:

$$\sigma^1 = (g_1^{\gamma})^{t \cdot l \cdot a_m} \cdot ((f_1)^{(-a_m^{-1})} \cdot g_1^t))^{l \cdot b_m},$$
$$\sigma^2 = (f_1)^{-a_m^{-1}} \cdot g_1^t, \qquad \sigma^3 = (f_1)^{-a_m^{-1}} \cdot g_2^t,$$

- it returns the signature $\sigma = (\sigma^1, \sigma^2, \sigma^3)$.

Let $f_1 = g_1^{\phi}$. We will now show that this is a valid signature. Note that a valid signature is of the form $(f_1^{\gamma \cdot l} \cdot ((g_1^{\gamma})^{a_m} \cdot g_1^{b_m})^{l \cdot r}, g_1^r, g_2^r)$. In this case, the reduction has set $r = -a_m^{-1} \cdot \phi + t$ and this means that the $f_1^{\gamma \cdot l}$ cancels out and the reduction does not need to compute f_1^{γ}.

Finally, $\mathcal{A}$ will output a valid signature under message m^*:

$$\hat{\sigma} = (\hat{\sigma}^1, \hat{\sigma}^2, \hat{\sigma}^3) = ((g_1^{\gamma \cdot \phi} \mathsf{PHF.Eval}(K, m^*)^{r^*})^{l^*}, g_1^{r^*}, g_2^{r^*}),$$

for which we hope that $a_{m^*} = 0$, where $(a_{m^*}, b_{m^*}) \leftarrow \mathsf{PHF.tdEval}(\tau_{\mathsf{PHF}}, m^*)$. Moreover, since this should be a valid forgery then we have that this signature is under a verification key $\hat{\mathsf{fvk}}$ for which $(\mathsf{fvk}, \hat{\mathsf{fvk}}) \in \mathcal{R}$. Thus, we have $\hat{\sigma} = ((f_1^{\gamma}(g_1^{r^*})^{b_{m^*}})^{l^*}, g_1^{r^*}, g_2^{r^*})$, for some unknown r^* but known b_{m^*}. Since $(\mathsf{fvk}, \hat{\mathsf{fvk}}) \in \mathcal{R}$. This means that $\hat{\mathsf{fvk}} = (t^{l^*}, A^{l^*}, B^{l^*}, C^{l^*}, D^{l^*}, K_{\mathsf{PHF}}^{l^*}) = ((f_1^{\alpha})^{l^*}, (h_1^{\beta})^{l^*}, (h_1)^{l^*}, (g_1^{\gamma \cdot d})^{l^*}, t^{l^*}, K_{\mathsf{PHF}}^{l^*})$. We now compute

$$T_1 = e(\hat{\sigma}^1, h_2) = e(f_1^{\gamma}(g_1^{r^*})^{b_{m^*}}, h_2^{l^*}) \qquad T_2 = e(h_1^{l^*}, g_2^{r^*})^{b_{m^*}} = e(g_1^{r^* \cdot b_{m^*}}, h_2^{l^*})$$
$$T_3 = e((f_1^{\alpha})^{l^*}, h_2) = e(f_1^{\alpha}, h_2^{l^*}) \qquad T_4 = e((h_1^{\beta})^{l^*}, f_2) = e(f_1^{\beta}, h_2^{l^*})$$

Finally, the reduction $\mathcal{R}$ returns 1 if $T_1 \cdot T_2^{-1} = T_3 \cdot T_4$ and 0, otherwise. Note that $T_1 \cdot T_2^{-1} = e(f_1^{\gamma}, h_2^{l^*})$ and the above equation is correct only if $\gamma = \alpha + \beta$.

The success probability of the reduction $\mathcal{R}$ depends on whether it can answer all signing queries of $\mathcal{A}$ and on the returned forgery (i.e. for which we must have $a_{m^*} = 0$). However, since we assume that the used hash function is a $(1, \mathsf{poly}(\lambda))$-programmable hash function, it follows that $\mathcal{R}$ has a non-negligible advantage in solving the decisional linear problem. □

Proof of Class Hiding

Theorem 3.5: Class Hiding of Construction 3.3

Construction 3.3 is fully class hiding, assuming the decisional Diffie-Hellman assumption in $\mathbb{G}_1$ holds.

Proof. Consider the following series of hybrids. We will use S_i to denote the event that $\mathcal{H}_i$ outputs true. We will also use the vector $\vec{u}$ to denote the key for the programmable hash function K_{PHF}. Let $\mathsf{fvk}' = (t', A', B', C', D', \vec{u'})$ be the verification key given to the adversary as part of the challenge. Moreover, let $\mathsf{fvk}_0 = (t_0, A_0, B_0, C_0, D_0, \vec{u}_0)$ and $\mathsf{fvk}_1 = (t_1, A_1, B_1, C_1, D_1, \vec{u}_1)$ be the public keys that are returned by the KeyGen algorithm on input of random coins ω_0 and ω_1 given to the adversary and $\hat{b}$ be the bit chosen by the challenger.

$\mathcal{H}_0$: The Class-Hiding experiment.

$\mathcal{H}_1$: In this game we change the way we sample fvk_0 and fvk_1. Instead of sampling directly from $\mathbb{G}_1$, we sample $a, b, c, d, x, \nu_1, \dots, \nu_\lambda \leftarrow \mathbb{Z}_p^*$ and set $A = g_1^a$, $B = g_1^b$, $C = g_1^c$, $D = g_1^d$, $X = g_1^x$ and $\vec{u} = (g_1^{\nu_0}, \dots, g_1^{\nu_\lambda})$. We use the invertible sampling algorithm to obtain corresponding randomness ω_0, ω_1 to provide to the adversary.

Moreover, we change the way fsk' and fvk' are computed from $(\mathsf{fsk}_{\hat{b}}\ \mathsf{fvk}_{\hat{b}})$, i.e. $\mathsf{fvk}' = (e(Q^x, g_2^y), Q^a, Q^b, Q^c, Q^d, (Q^{\nu_0}, \dots, Q^{\nu_\lambda}))$, and $\mathsf{fsk}' = (y, Q^x, \mathsf{fvk}')$, where $Q \leftarrow \mathbb{G}_1$ is uniformly random. In other words, instead of using a value δ to move the verification and signing keys, we use a group element Q to do it.

Because the of the indistinguishability property of the invertible sampling algorithm and since the distribution of the keys does not change, it follows that $\Pr[S_1] = \Pr[S_0]$. Note that since the signing key fsk' is known, the signing oracle $\mathsf{OSign}(\cdot)$ can be perfectly simulated for any adversary.

$\mathcal{H}_2$: In this game instead of computing

$$\mathsf{fvk}' = (e(Q^{x_{\hat{b}}}, g_2^{y_{\hat{b}}}), Q^a, Q^b, Q^c, Q^d, (Q^{\nu_0}, \dots, Q^{\nu_\lambda}))$$

as in $\mathcal{H}_1$, we sample $A' \leftarrow \mathbb{G}_1$ and set

$$\mathsf{fvk}' = (e(Q^{x_{\hat{b}}}, g_2^{y_{\hat{b}}}), A', Q^b, Q^c, Q^d, (Q^{\nu_0}, \dots, Q^{\nu_\lambda})).$$

We will show that this transition only lowers the adversary's advantage by a negligible fraction. In particular, we will show a reduction $\mathcal{R}$ that uses an adversary $\mathcal{A}$ that can distinguish between those two games to break the decisional Diffie-Hellman assumption in $\mathbb{G}_1$. Let $(g_1^\alpha, g_1^\beta, g_1^\gamma)$ be an instance of this problem in $\mathbb{G}_1$. $\mathcal{R}$ samples $r_{0,A}, r_{1,A} \leftarrow \mathbb{Z}_p^*$ and sets $A_0 = (g_1^\alpha)^{r_{0,A}}$, $A_1 = (g_1^\alpha)^{r_{1,A}}$.

Additionally, the reduction uses $Q = g_1^\beta$ and the public key

$$\mathsf{fvk}' = (e(Q^{x_{\hat{b}}}, g_2^{y_{\hat{b}}}), (g_1^\gamma)^{r_{\hat{b},A}}, Q^b, Q^c, Q^d, (Q^{\nu_0}, \ldots, Q^{\nu_\lambda})).$$

Note that since $\mathcal{R}$ knows the signing key fsk' it can answer signing queries. Finally, notice, that if $\gamma = \alpha \cdot \beta$ then (fvk', σ) have the same distribution as in $\mathcal{H}_1$ and otherwise as in $\mathcal{H}_2$. Thus, we have $|\Pr[S_2] - \Pr[S_1]| \leq \mathsf{Adv}_{\mathcal{A}}^{\mathsf{DDH}}(\lambda)$.

$\mathcal{H}_3$ **(series of sub-games):** In this game instead of computing $\mathsf{fvk}' = (e(Q^{x_{\hat{b}}}, g_2^{y_{\hat{b}}}), A', Q^b, Q^c, Q^d, (Q^{\nu_1}, \ldots, Q^{\nu_\lambda}))$ as in $\mathcal{H}_2$, we sample B', C', $, D', u'_0, \ldots, u'_{1\lambda} \leftarrow \mathbb{G}_1$ and set $\mathsf{fvk}' = (e(Q^{x_{\hat{b}}}, g_2^{y_{\hat{b}}}), A', B', C', D', (u'_0, \ldots, u'_{1\lambda}))$.

This transition is composed of a number of sub-games, in which we change each element of the verification key fvk' separately. We can use the same reduction as above and show that each change lowers the adversary's advantage by at most $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{DDH}}(\lambda)$. It is worth noting, that the reduction can always create a valid signature, since the signing key $\mathsf{fsk}' = (y_{\hat{b}}, Q^{x_{\hat{b}}}, \mathsf{fvk}')$ can be computed by $\mathcal{R}$. Thus, we have $|\Pr[S_3] - \Pr[S_2]| \leq (4 + \lambda) \cdot \mathsf{Adv}_{\mathcal{A}}^{\mathsf{DDH}}(\lambda)$.

Let us now take a look at the randomized verification key and signature given to the adversary. Because of all the changes, we have:

$$\mathsf{fvk}' = (e(Q^{x_{\hat{b}} \cdot y_{\hat{b}}}, g_2), A', B', C', D'\vec{u'})$$

and signatures from the oracle are of the form

$$(Q^{x_{\hat{b}} \cdot y_{\hat{b}}}(\mathsf{PHF.Eval}(K, m))^r, g_1^r, g_2^r)$$

for some $r \in \mathbb{Z}_p^*$ and $A', B', C', D', \vec{u'}(= K_{\mathsf{PHF}}), Q$, which are independent of the bit $\hat{b}$ and the original public keys. Since the value Q is random and only appears as part of the term $Q^{x_{\hat{b}} \cdot y_{\hat{b}}}$, we can always restate this term to $Q'^{x_{1-\hat{b}} \cdot y_{1-\hat{b}}}$ where $Q' = Q^{(x_{1-\hat{b}} \cdot y_{1-\hat{b}}) \cdot (x_{\hat{b}} \cdot y_{\hat{b}})^{-1}}$ and Q' is a random value. It follows that the adversaries advantage is zero, i.e. $\Pr[S_3] = 0$.
Finally, we have $\mathsf{Adv}_{\mathcal{A},\mathsf{SFPK}}^{\mathsf{Class\text{-}Hiding}}(\lambda) = \Pr[S_0] \leq (5 + \lambda) \cdot \mathsf{Adv}_{\mathcal{A}}^{\mathsf{DDH}}(\lambda)$.

□

3.4.2 With Setup and Canonical Representatives

In this section we propose a signature scheme with flexible public keys in the common reference string model. Security relies on the bilinear decisional Diffie-Hellman assumption instead of the decisional linear assumption. Unlike the scheme in section 3.4.1 this scheme only provides class hiding with key corruption. We will see in section 3.5.1 and in chapter 4 that this is sufficient for group signature constructions.

We assume that both the SFPK.KeyGen and SFPK.tdGen algorithms output a verification key that is the canonical representative of its equivalence class. Further we assume that every user has access to a collision resistant hash function H, which we express by including it in the output of Setup. The SFPK.MoveVk and SFPK.MoveSk algorithms work, as before, by drawing uniformly at random an exponent $\delta \in \mathbb{Z}_p$ and raising every component of the verification key, or respectively the signing key to the power of δ. More details can be found in construction 3.4.

Construction 3.4: Canonical SFPK with Public Parameters

```
SFPK.Setup($1^\lambda$)
    $K_{\mathsf{PHF}} \leftarrow \mathsf{PHF.Gen}(1^\lambda)$
    $y, z \leftarrow \mathbb{Z}_p^*$
    $Y_1 \leftarrow g_1^y$
    $Y_2 \leftarrow g_2^y$
    $\hat{g} \leftarrow g_1^z$
    $\mathsf{pp_{SFPK}} := (\mathsf{BG}, Y_1, Y_2, K_{\mathsf{PHF}}, \hat{g}, \mathsf{H})$
    return $\mathsf{pp_{SFPK}}$
```

```
SFPK.Check($\mathsf{pp_{SFPK}}, \mathsf{td}, \mathsf{fvk}$)
    parse
      $\mathsf{fvk} = (\mathsf{fvk}_1, \mathsf{fvk}_2)$
    if $e(\mathsf{fvk}_1, \mathsf{td}) = e(\mathsf{fvk}_2, g_2)$
     then return true
    else return false
```

```
SFPK.KeyGen($\mathsf{pp_{SFPK}}, 1^\lambda$)
    parse
      $\mathsf{pp_{SFPK}} = (\mathsf{BG}, Y_1, Y_2, K_{\mathsf{PHF}}, \hat{g}, \mathsf{H})$
    $x \leftarrow \mathbb{Z}_p^*$
    $\mathsf{fvk} := (g_1, g_1^x)$
    $\mathsf{fsk} := (Y_1^x, \mathsf{fvk})$
    return $(\mathsf{fsk}, \mathsf{fvk})$
```

```
SFPK.tdGen($\mathsf{pp_{SFPK}}, 1^\lambda$)
    parse
      $\mathsf{pp_{SFPK}} = (\mathsf{BG}, Y_1, Y_2, K_{\mathsf{PHF}}, \hat{g}, \mathsf{H})$
    $x \leftarrow \mathbb{Z}_p^*$
    $\mathsf{fvk} := (g_1, g_1^x)$
    $\mathsf{fsk} := (Y_1^x, \mathsf{fvk})$
    $\mathsf{td} := (g_2^x)$
    return $(\mathsf{fsk}, \mathsf{fvk}, \mathsf{td})$
```

```
$\mathsf{SFPK.Sign}(\mathsf{pp}_{\mathsf{SFPK}}, \mathsf{fsk}, m)$

parse
    $\mathsf{pp}_{\mathsf{SFPK}} = (\mathsf{BG}, Y_1, Y_2, K_{\mathsf{PHF}}, \hat{g}, \mathsf{H})$
    $\mathsf{fsk} = (Z, \mathsf{fvk})$
$r, s \leftarrow \mathbb{Z}_p^*$
$\sigma^{(2)} := g_1^r$
$\sigma^{(3)} := g_2^r$
$\zeta \leftarrow \mathsf{H}(m||\sigma^2||\sigma^3||\mathsf{fvk})$
$M := g_1^{\zeta} \cdot \hat{g}^s$
$\sigma^{(1)} \leftarrow Z \cdot (\mathsf{PHF.Eval}(K_{\mathsf{PHF}}, M))^r$
$\sigma := (\sigma^{(1)}, \sigma^{(2)}, \sigma^{(3)}, s)$
return $\sigma$
```

```
$\mathsf{SFPK.Verify}(\mathsf{pp}_{\mathsf{SFPK}}, \mathsf{fvk}, m, \sigma)$

parse
    $\mathsf{pp}_{\mathsf{SFPK}} = (\mathsf{BG}, Y_1, Y_2, K_{\mathsf{PHF}}, \hat{g}, \mathsf{H})$
    $\mathsf{fvk} = (\mathsf{fvk}_1, \mathsf{fvk}_2)$
    $\sigma = (\sigma^1, \sigma^2, \sigma^3, s)$
$\zeta \leftarrow \mathsf{H}(m||\sigma^2||\sigma^3||\mathsf{fvk})$
$M := g_1^{\zeta} \cdot \hat{g}^s$
$h \leftarrow \mathsf{PHF.Eval}(K_{\mathsf{PHF}}, M)$
if $e(\sigma^2, g_2) = e(g_1, \sigma^3)$ and
    $e(\sigma^1, g_2) = e(\mathsf{fvk}_2, Y_2) \cdot e(h, \sigma^3)$
 then return true
else return false
```

```
$\mathsf{SFPK.MoveVk}(\mathsf{pp}_{\mathsf{SFPK}}, \mathsf{fvk}, \delta)$

parse $\mathsf{fvk} = (\mathsf{fvk}_1, \mathsf{fvk}_2)$
$\mathsf{fvk}' := (\mathsf{fvk}_1^{\delta}, \mathsf{fvk}_2^{\delta})$
return $\mathsf{fvk}'$
```

```
$\mathsf{SFPK.MoveSk}(\mathsf{pp}_{\mathsf{SFPK}}, \mathsf{fsk}, \delta)$

parse $\mathsf{fsk} = (Z, \mathsf{fvk})$
$\mathsf{fvk}' \leftarrow \mathsf{MoveVk}(\mathsf{fvk}, \delta)$
$\mathsf{fsk}' := (Z^{\delta}, \mathsf{fvk}')$
return $\mathsf{fsk}'$
```

Correctness. Observe first that SFPK.KeyGen and SFPK.tdGen generate signing and verification keys fsk, fvk in identical fashion, hence their distributions are also identical.

For $\lambda \in \mathbb{N}$ let

$$(\mathsf{BG}, Y_1 = g_1^y, Y_2 = g_2^y, K_{\mathsf{PHF}}, \hat{g}, \mathsf{H}) \leftarrow \mathsf{SFPK.Setup}(1^\lambda),$$
$$(\mathsf{fsk}, \mathsf{fvk}, \mathsf{td}) \leftarrow \mathsf{SFPK.tdGen}(1^\lambda),$$

where $\mathsf{fvk} = (g_1, g_1^x)$, $\mathsf{fsk} = Y_1^x, \mathsf{fvk}$ and $\mathsf{td} = g_2^x$ for some $x \in \mathbb{Z}_p^*$.

For any $\mathsf{fvk}'' = (g_1^a, g_1^b)$ for some $a, b \in \mathbb{Z}_p^*$ we have $\mathsf{SFPK.Check}(\mathsf{pp}_{\mathsf{SFPK}}, \mathsf{td}, \mathsf{fvk}'') =$ true if and only if $e(g_1^a, \mathsf{td}) = e(g_1^a, g_2^x) = e(g_1^b, g_2)$, i.e. if and only if $ax = b$, hence if and only if $\mathsf{fvk}_2^a = \mathsf{fvk}_2''$.

Then, if $\delta \in \Delta$, and $(\mathsf{fsk}'\mathsf{fvk}') \leftarrow \mathsf{SFPK.MoveKeys}(\mathsf{fsk}, \mathsf{fvk}, \delta)$ we have

$$\mathsf{fvk}' = (g_1^{\delta}, g_1^{x\delta})$$
$$\mathsf{fsk}' = (Y_1^{x\delta}, \mathsf{fvk}').$$

Consider

$$\sigma = (\sigma^1, \sigma^2, \sigma^3, s) = (Y_1^{x\delta} \cdot h^r, g_1^r, g_2^r, s),$$

a signature on message m under signing key fsk' where $h = \mathsf{PHF.Eval}(K_{\mathsf{PHF}}, M)$. We have

$$\begin{aligned} e(\sigma^2, g_2) &= e(g_1^r, g_2) \\ &= e(g_1, g_2^r) \\ &= e(g_1, \sigma^3). \end{aligned}$$

and

$$\begin{aligned} e(\sigma^1, g_2) &= e(Y_1^{x\delta} \cdot h^r, g_2) \\ &= e(Y_1^{x\delta}, g_2) \cdot e(h^r, g_2) \\ &= e(g_1^{yx\delta}, g_2) \cdot e(h^r, g_2) \\ &= e(g_1^{x\delta}, g_2^y) \cdot e(h, g_2^r) \\ &= e(g_1^{x\delta}, Y_2) \cdot e(h, g_2^r) \\ &= e(\mathsf{fvk}_2', Y_2) \cdot e(h, \sigma^3) \end{aligned}$$

Thus, if PHF is correct and compatible with the relation, we have

$$\mathsf{SFPK.Verify}(\mathsf{fvk}', m, \sigma) = \mathsf{true}.$$

Setting $\delta = 1$ shows that verification succeeds for canonical keys as well.

All together, this shows correctness of the construction.

Proof of Unforgeability

Theorem 3.6: Unforgeability of Construction 3.4

Construction 3.4 is strongly existentially unforgeable under flexible public key in the common reference string model, assuming the bilinear decisional Diffie-Hellman assumption holds in $\mathbb{G}_1$, that PHF is $(1, \mathsf{poly}(\lambda))$-programmable and H is collision-resistant.

Proof. Let $(\sigma^*, m^*, \mathsf{fvk}^*)$ be the forgery returned by an adversary $\mathcal{A}$, where $\sigma^* = (\sigma_1^*, \sigma_2^*, \sigma_3^*, s^*)$. We distinguish three types of strategies of the adversary:

Type 1: We call the adversary a type 1 adversary if there exists a verification key fvk and signature $\sigma = (\sigma_1, \sigma_2, \sigma_3, s)$ on message m generated by oracle OSign[1] or OSign[2], where

$$\mathsf{H}(m^* || \sigma_2^* || \sigma_3^* || \mathsf{fvk}^*) = \mathsf{H}(m || \sigma_2 || \sigma_3 || \mathsf{fvk}).$$

It is easy to see that the adversary broke the collision-resistance of function H, and we can build a reduction $\mathcal{R}$ that uses $\mathcal{A}_1$ to break collision-resistance of function H by simulating the system and returning

$$(m^*||\sigma_2^*||\sigma_3^*||\mathsf{fvk}^*, m||\sigma_2||\sigma_3||\mathsf{fvk})$$

as a valid collision.

Type 2: We call the adversary a type 2 adversary if there exists a verification key fvk and signature $\sigma = (\sigma_1, \sigma_2, \sigma_3, s)$ on message m generated by oracle OSign^1 or OSign^2, where $e^* = \mathsf{H}(m^*||\sigma_2^*||\sigma_3^*||\mathsf{fvk}^*) \neq \mathsf{H}(m||\sigma_2||\sigma_3||\mathsf{fvk}) = e$ but $M^* = g_1^{e^*} \cdot \hat{g}^{s^*} = g_1^e \cdot \hat{g}^s = M$.

In this case we show that a type 2 can be used to break the discrete logarithm assumption. We can apply the same reasoning as for Pedersen commitments, i.e. the reduction can set $\hat{g}$ as the element for which we want to compute the discrete logarithm in respect to g_1. The reduction can then simply simulate the whole system for $\mathcal{A}_2$ and output $(e - e^*)/(s^* - s)$.

Type 3: We call the adversary a type 3 adversary in all other cases. In particular, we ensure that M^* is distinct from all M's used in the oracles OSign^1 and OSign^2.

Let $(\mathsf{BG}, g_1^a, g_1^b, g_1^c, g_1^d, g_2^a, g_2^b, g_2^c, g_2^d)$ be an instance of the bilinear decisional Diffie-Hellman problem. We will show that any efficient adversary $\mathcal{A}_3$ can be used to break the above problem instance. To do so, we will build a reduction algorithm $\mathcal{R}$ that uses $\mathcal{A}_3$ in a black box manner, i.e. it plays the role of the challenger in the unforgeability experiment.

First $\mathcal{R}$ prepares the common reference string crs by setting $Y_1 = g_1^a$, $Y_2 = g_2^a$, $\hat{g} = g_1^z$, for some $z \leftarrow \mathbb{Z}_p^*$ and executes the trapdoor generation algorithm $(K_{\mathsf{PHF}}, \tau_{\mathsf{PHF}}) \leftarrow \mathsf{PHF.tdGen}(1^\lambda, g_1^a, g_1)$. Note that $\mathsf{td}_{\mathsf{crs}}$ is not publicly known, so $\mathcal{R}$ does not have to know the exponent a but still knows z. Next, $\mathcal{R}$ prepares the verification key fvk and the trapdoor τ. For this it uses the values g_1^b and g_2^b from the problem instance. It sets $\mathsf{fvk} = (g_1, g_1^b)$ and $\tau = (g_2^b)$.

To answer $\mathcal{A}$'s signing queries for message m and randomness t_1 (which is equal to 1 for oracle OSign^1), the reduction $\mathcal{R}$ follows the following steps:

1. it chooses random values $t_2 \leftarrow \mathbb{Z}_p^*$,
2. it computes $M = g_1^{e'} \cdot \hat{g}^{s'}$ for some $e', s' \leftarrow \mathbb{Z}_p^*$,
3. it computes $(a_m, b_m) \leftarrow \mathsf{PHF.tdEval}(\tau_{\mathsf{PHF}}, M)$ and aborts if $a_m = 0$,

4. it computes $\mathsf{fvk}' \leftarrow \mathsf{SFPK.MoveVk}(\mathsf{fvk}, t_1)$,
5. it computes:

$$\begin{aligned}\sigma^1 &= (g_1^a)^{t_2} \cdot ((g_1^b)^{(-a_m^{-1} \cdot t_1)} \cdot g_1^{t_2}))^{b_m},\\ \sigma^2 &= (g_1^b)^{-a_m^{-1} \cdot t_1} \cdot g_1^{t_2},\\ \sigma^3 &= (g_2^b)^{-a_m^{-1} \cdot t_1} \cdot g_2^{t_2}\\ e &= \mathsf{H}(m || \sigma^2, \sigma^3, \mathsf{fvk}'),\\ s &= ((e' - e) + s' \cdot z)/z,\end{aligned}$$

6. set the signature

$$\sigma := (\sigma^1, \sigma^2, \sigma^3, s).$$

It is easy to see that this is a valid signature. Note that a valid signature is of the form $(g_1^{a \cdot b \cdot t_1} \cdot ((g_1^a)^{a_m} \cdot g_1^{b_m})^r, g_1^r, g_2^r, s)$. In this case, the reduction has set $r = -a_m^{-1} \cdot b \cdot t_1 + t_2$ and this means that the $g_1^{a \cdot b \cdot t_1}$ cancels out and the reduction does not need to compute $g_1^{a \cdot b}$. Note that this only works because $a_m \neq 0$.

It follows that for the forgery $(\mathsf{fvk}^*, m^*, \sigma^*, s^*)$ of $\mathcal{A}$ we require that $(a_{m^*}, b_{m^*}) \leftarrow \mathsf{PHF.tdEval}(\tau_{\mathsf{PHF}}, M^*)$ and $a_{M^*} = 0$, where

$$M^* = g_1^{e^*} \hat{g}^{s^*} \text{ and } e^* = \mathsf{H}(m^* || \sigma^2 || \sigma^3 || \mathsf{fvk}^*).$$

In such a case, the reduction works as follows:

1. parse σ^* as $(\sigma^1, \sigma^2, \sigma^3, s^*)$,
2. compute

$$\begin{aligned}g_1^{a \cdot b \cdot t^*} &= \sigma^1 \cdot (\sigma^2)^{-b_{m^*}}\\ &= \left(g_1^{a \cdot b \cdot t^*} \cdot ((g_1^a)^{a_{m^*}} \cdot g_1^{b_{m^*}})^{r^*}\right) \cdot (g_1^{r^*})^{-b_{m^*}},\end{aligned}$$

3. parse fvk^*, and since for a valid forgery we have $\mathsf{fvk}^* \approx_{\mathsf{exp}} \mathsf{fvk}$, we have $\mathsf{fvk}^* = (g_1^{t^*}, (g_1^b)^{t^*})$ and $\mathcal{R}$ can use $g_1^{t^*}$,
4. output 1 iff $e(g_1^{a \cdot b \cdot t^*}, g_2^c) = e(g_1^{t^*}, g_2^d)$.

The probability that $\mathcal{R}$ successfully solves the bilinear decisional Diffie-Hellman problem depends on the advantage of $\mathcal{A}$ and the probability that $\mathcal{R}$'s simulation succeeds. Since the programmable hash function PHF is $(1, \mathsf{poly}(\lambda))$-programmable and because this is a type 3 adversary, we conclude that this probability is non-negligible. Note that since in this case we use $\mathcal{A}_3$, M^* is distinct from all M's used in OSign^1 and OSign^2, which is not the case for type 1 and type 2 adversaries.

□

Proof of Class Hiding

Theorem 3.7: Class Hiding with Key Corruption of Construction 3.4

Construction 3.4 is class hiding with key corruption in the common reference string model, assuming the decisional Diffie-Hellman assumption holds in $\mathbb{G}_1$.

Proof. We start with $\mathcal{H}_0$ which is the original class hiding experiment and let S_0 be the event that the experiment evaluates to true, i.e. the adversary wins. We will use S_i to denote the event that the adversary wins the class hiding experiment in $\mathcal{H}_i$.

Let $\mathsf{fvk} = (A, B)$ be the verification key given to the adversary, $\mathsf{fvk}_0 = (A_0, B_0) = (g_1, g_1^{x_0})$ and $\mathsf{fvk}_1 = (A_0, B_1) = (g_1, g_1^{x_1})$ be the public keys that are returned by $\mathsf{SFPK.KeyGen}$, $\mathsf{fsk}_0 = (Y_1^{x_0}, \mathsf{fvk}_0)$ and $\mathsf{fsk}_1 = (Y_1^{x_1}, \mathsf{fvk}_1)$ the corresponding secret keys and $\hat{b}$ be the bit chosen by the challenger.

$\mathcal{H}_0$: The original class hiding game.

$\mathcal{H}_1$: In this game we do not use the $\mathsf{SFPK.MoveSk}$ algorithm to compute fsk and fvk but compute them as $\mathsf{fvk} = (Q, Q^{x_{\hat{b}}})$, and $\mathsf{fsk} = ((Q^{x_{\hat{b}}})^y, \mathsf{fvk})$, where $Y_1 = g_1^y$ is part of the common reference string crs generated by the challenger. In other words, instead of using the exponent r to randomize the verification key and signing key, we use a group element Q to do it.

Since the distribution of the keys does not change, it follows that $\Pr[S_1] = \Pr[S_0]$. Note that the oracle can still use fsk to compute valid signatures.

$\mathcal{H}_1$: In this game instead of computing $\mathsf{fvk} = (Q, Q^{x_{\hat{b}}})$ as in $\mathcal{H}_1$, we sample $B' \leftarrow \mathbb{G}_1$ and set $\mathsf{fvk} = (Q, B')$.

We will show that this transition only lowers the adversaries advantage by a negligible fraction. This can be show by construction using a reduction $\mathcal{R}$ that uses an adversary $\mathcal{A}$ that can distinguish between those two games to break the decisional Diffie-Hellman assumption in $\mathbb{G}_1$.

Let $(g_1^{\alpha}, g_1^{\beta}, g_1^{\gamma})$ be an instance of this problem in $\mathbb{G}_1$. $\mathcal{R}$ samples $r_0, r_1 \leftarrow \mathbb{Z}_p^*$ and sets $B_0 = (g_1^{\alpha})^{r_0}$, $B_1 = (g_1^{\alpha})^{r_1}$. Note that in such a case, we also have to set $\mathsf{fsk}_0 = ((B_0)^y, \mathsf{fvk}_0)$ and $\mathsf{fsk}_1 = ((B_1)^y, \mathsf{fvk}_1)$. Additionally, the reduction uses $Q = g_1^{\beta}$ and the public key $\mathsf{fvk} = (Q, (g_1^{\gamma})^{r_{\hat{b}}})$. Note that the reduction can use the signing key $\mathsf{fsk} = (((g_1^{\gamma})^{r_{\hat{b}}})^y, \mathsf{fvk})$ to generate signatures and answer signing queries. Now $\gamma = \alpha \cdot \beta$ then fvk has the same distribution as in $\mathcal{H}_1$ and otherwise as in $\mathcal{H}_2$. Thus, it follows that $|\Pr[S_2] - \Pr[S_1]| \leq \mathsf{Adv}_{\mathcal{A}}^{\mathsf{DDH-1}}(\lambda)$.

We will now show that we have $\Pr[S_2] = \frac{1}{2}$. This follow from the fact that we have $\mathsf{fvk} = (Q, B')$ and signatures of the form $\sigma = ((B')^y \cdot (\mathsf{PHF.Eval}(K_{\mathsf{PHF}}, m))^r, g_1^r, g_2^r, s)$ for some $r \in \mathbb{Z}_p^*$ and Q, B', which are independent of the bit $\hat{b}$. Thus, we have $\mathsf{Adv}_{\mathcal{A},\mathsf{SFPK}}^{\mathsf{Class\text{-}Hiding}}(\lambda) = \Pr[S_0] \leq \mathsf{Adv}_{\mathcal{A}}^{\mathsf{DDH-1}}(\lambda)$. □

3.5 Applications of SFPK to Privacy-Preserving Signatures

3.5.1 Static Group Signatures

We now present an efficient generic construction of static group signatures that uses SFPK as a building block and which is secure in the model by Bellare, Micciancio and Warinschi [BMW03] that we have restated in section 2.3. Let

- SPS = (Setup, KeyGen, Sign, Verify, Move, ValKey) be a structure-preserving signature scheme on equivalence classes.
- SFPK = (Setup, KeyGen, tdGenSign, Verify, MoveKeys, Check) be a signature scheme with flexible public keys.

The core idea of the scheme is to use a compatible set of signatures with flexible public keys and structure-preserving signatures on equivalence classes. To generate the group keys, the group manager generates a pair of SFPK signing and verification keys for each user and "certifies" the verification keys with an SPS-EQ signature under a SPS-EQ verification key, which will be part of the group public key. To give a group signature on a message, a user changes the representative of their SFPK keys, and also changes the representation of the SPS-EQ certificate using the same randomizer. The message is signed with the randomized SFPK signing key. The group signature is then composed of the SFPK signature, the randomized verification key and the randomized SPS-EQ certificate.

To enable subsequent opening, the group manager generates the SFPK keys using SFPK.tdGen and stores the trapdoors. Opening is then performed using the stored trapdoors with the SFPK.Check algorithm.[2] Since the group manager is trusted, it may

[2]If the SFPK.KeyGen algorithm is used instead of SFPK.tdGen to compute the SFPK key pairs, there is no efficient opening procedure and the combination of SFPK and SPS-EQ signature scheme yields a self-blindable certificate scheme [Ver01].

also generate $\mathsf{pp}_{\mathsf{SFPK}} \leftarrow \mathsf{SFPK.Setup}$ for the SFPK signatures and use it as part of the group public key. This allows us to use schemes which are secure in the multi-user setting, e.g. construction 3.4.

Full-anonymity of the scheme follows from the perfect adaptation and unforgeability of SPS-EQ signatures, the class hiding property of the SFPK scheme and its strong existential unforgeability. On the other hand, full-traceability follows from the unforgeability of SPS-EQ and the existential unforgeability of the SFPK.

Construction 3.5: Static Group Signature Scheme

```
$\mathsf{GS.KeyGen}(1^\lambda, n)$

$\mathsf{pp}_{\mathsf{SPS}} \leftarrow \mathsf{SPS.Setup}(1^\lambda)$
$\mathsf{pp}_{\mathsf{SFPK}} \leftarrow \mathsf{SFPK.Setup}(1^\lambda)$
$(\mathsf{sk}_{\mathsf{SPS}}, \mathsf{vk}_{\mathsf{SPS}}) \leftarrow \mathsf{SPS.KeyGen}(\mathsf{pp}_{\mathsf{SPS}})$
**foreach** user $i \in [n]$ :
 $(\mathsf{fsk}^i, \mathsf{fvk}^i, \mathsf{td}^i) \leftarrow \mathsf{SFPK.tdGen}(\mathsf{pp}_{\mathsf{SFPK}})$
 $\sigma^i_{\mathsf{EQ}} \leftarrow \mathsf{SPS.Sign}(\mathsf{sk}_{\mathsf{SPS}}, \mathsf{fvk}^i)$
$\mathsf{gpk} := (\mathsf{pp}_{\mathsf{SPS}}, \mathsf{pp}_{\mathsf{SFPK}}, \mathsf{vk}_{\mathsf{SPS}})$
$\mathsf{gmsk} := \{(\mathsf{td}^i, \mathsf{fvk}^i)\}_{i=1}^n$
$\vec{\mathsf{gsk}} := \{(\mathsf{fvk}^i, \mathsf{fsk}^i, \sigma^i_{\mathsf{EQ}})\}_{i=1}^n$
**return** $(\mathsf{gpk}, \mathsf{gmsk}, \vec{\mathsf{gsk}})$
```

```
$\mathsf{GS.Sign}(\vec{\mathsf{gsk}}[i], m)$

**parse** $\vec{\mathsf{gsk}}[i] = (\mathsf{fvk}, \mathsf{fsk}, \sigma_{\mathsf{EQ}})$
$\delta \leftarrow \Delta$
$\mathsf{fvk}' \leftarrow \mathsf{SFPK.MoveVk}(\mathsf{fvk}, \delta)$
$\mathsf{fsk}' \leftarrow \mathsf{SFPK.MoveSk}(\mathsf{fsk}, \delta)$
$(\mathsf{fvk}', \sigma'_{\mathsf{EQ}}) \leftarrow \mathsf{SPS.Move}(\mathsf{vk}_{\mathsf{SPS}}, \mathsf{fvk}, \sigma_{\mathsf{EQ}}, \delta)$
$M := m \| \sigma'_{\mathsf{EQ}} \| \mathsf{fvk}'$
$\sigma_{\mathsf{SFPK}} \leftarrow \mathsf{SFPK.Sign}(\mathsf{fsk}', M)$
$\sigma := (\mathsf{fvk}', \sigma_{\mathsf{SFPK}}, \sigma'_{\mathsf{EQ}})$
**return** $\sigma$
```

```
$\mathsf{GS.Verify}(\mathsf{gpk}, m, \sigma)$

**parse**
    $\sigma = (\mathsf{fvk}, \sigma_{\mathsf{SFPK}}, \sigma_{\mathsf{EQ}})$
    $\mathsf{gpk} = (\mathsf{pp}_{\mathsf{SPS}}, \mathsf{pp}_{\mathsf{SFPK}}, \mathsf{vk}_{\mathsf{SPS}})$
**if** $\mathsf{SPS.Verify}(\mathsf{vk}_{\mathsf{SPS}}, \mathsf{fvk}, \sigma_{\mathsf{EQ}}) = \mathsf{false}$
 **then** **return** false
$M := m \| \sigma_{\mathsf{EQ}} \| \mathsf{fvk}$
**return** $\mathsf{SFPK.Verify}(\mathsf{fvk}, M, \sigma_{\mathsf{SFPK}})$
```

```
$\mathsf{GS.Open}(\mathsf{gmsk}, m, \sigma)$

**parse**
    $\sigma = (\mathsf{fvk}, \sigma_{\mathsf{SFPK}}, \sigma_{\mathsf{EQ}})$
    $\mathsf{gmsk} = \{(\mathsf{td}^i, \mathsf{fvk}^i)\}_{i=1}^n$
**if** $\mathsf{GS.Verify}(\mathsf{gpk}, m, \sigma) = \mathsf{false}$
 **then** **return** $\perp$
**if** $\forall i \in [n]$ :
    $\mathsf{SFPK.Check}(\mathsf{td}^i, \mathsf{fvk}) = \mathsf{false}$
 **then** **return** $\perp$
**else let** $i$ **be s.t.**
    $\mathsf{SFPK.Check}(\mathsf{td}^i, \mathsf{fvk}) = \mathsf{true}$
**return** $i$
```

Correctness. Let $\lambda, n \in \mathbb{N}$, $m \in \mathcal{M}$. Given $(\mathsf{gpk}, \mathsf{gmsk}, \vec{\mathsf{gsk}}) \leftarrow \mathsf{GS.KeyGen}(1^\lambda, n)$, we have

$$\begin{aligned}\mathsf{gpk} &= (\mathsf{pp}_{\mathsf{SPS}}, \mathsf{pp}_{\mathsf{SFPK}}, \mathsf{vk}_{\mathsf{SPS}})\\ \mathsf{gmsk} &= \left\{(\mathsf{td}^i, \mathsf{fvk}^i)\right\}_{i=1}^n\\ \vec{\mathsf{gsk}} &= \left\{(\mathsf{fvk}^i, \mathsf{fsk}^i, \sigma_{\mathsf{EQ}}^i)\right\}_{i=1}^n .\end{aligned}$$

For any $i \in [n]$ a signature on m under $\vec{\mathsf{gsk}}[i]$ has the form

$$\sigma = (\mathsf{fvk}', \sigma_{\mathsf{SFPK}}, \sigma'_{\mathsf{EQ}}).$$

From the correctness of the SPS-EQ and SFPK schemes, it follows that

$$\mathsf{SPS.Verify}(\mathsf{vk}_{\mathsf{SPS}}, \mathsf{fvk}', \sigma'_{\mathsf{EQ}}) = \mathsf{true},$$

since σ'_{EQ} was derived from σ^i_{EQ} with the same randomizer that was used to move fvk^i to fvk'. Then, again from the correctness of the SFPK scheme, it follows that $\mathsf{SFPK.Verify}(\mathsf{fvk}', M, \sigma_{\mathsf{SFPK}}) = \mathsf{true}$, since M was signed with the equally transformed fsk'. This shows correctness of signing and verification.

From the correctness of SFPK it also follows that for any valid signature as above,

$$\mathsf{SFPK.Check}(\mathsf{td}^i, \mathsf{fvk}') = \mathsf{true}$$

and $\mathsf{SFPK.Check}(\mathsf{td}^j, \mathsf{fvk}') = \mathsf{false}$ for any $j \in [n] \setminus \{i\}$. This shows correctness of opening.

Proof of Full Traceability

Theorem 3.8: Full Traceability of Construction 3.5

Construction 3.5 is fully traceable if the SPS-EQ and the SFPK signature schemes are each existentially unforgeable under chosen-message attack.

The proof relies on the fact that the only way for an adversary to win the full traceability game is by either creating a new group member (thus directly breaking the unforgeability of the SPS-EQ scheme) or by creating a forged signature for an existing group member (thus breaking the unforgeability of the SFPK scheme).

Proof. We will use the game-based approach. Let us denote by S_i the event that the adversary wins the full traceability experiment in $\mathcal{H}_i$. Let $(m^*, \sigma^* = (\mathsf{fvk}^*, \sigma^*, \sigma^*_{\mathsf{EQ}}))$ be the forgery output by the adversary.

$\mathcal{H}_0$: The original traceability experiment.

$\mathcal{H}_1$: We abort in case $\mathsf{GS.Open}(\mathsf{gmsk}, m^*, \sigma^*) = \perp$ but $\mathsf{GS.Verify}(\mathsf{gpk}, m^*, \sigma^*) = 1$. Informally, we exclude the case that the adversary creates a new user from outside the group, i.e. a new SPS-EQ signature.

We will show that this only decreases the adversary's advantage by a negligible fraction. In particular, we will show that any adversary $\mathcal{A}$ returns a forgery for which we abort, can be used to break the existential unforgeability of the SPS-EQ signature scheme. The reduction algorithm uses the signing oracle to compute all signature σ^i_{EQ} of honest users. Finally, if the adversary returns $(m^*, \sigma^* = (\mathsf{fvk}^*, \sigma^*, \sigma^*_{\mathsf{EQ}}))$, the reduction algorithm returns $(\mathsf{fvk}^*, \sigma^*_{\mathsf{EQ}})$ as a valid forgery. We note that by correctness of the SFPK scheme, if fvk^* is in a relation to a verification key of an honest user, then we can always open this signature. It follows that fvk^* is from a different equivalence class and the values returned by the reduction algorithm are a valid forgery against the SPS-EQ signature scheme.

It follows that $|\Pr[S_1] - \Pr[S_0]| \leq \mathsf{Adv}^{\ell,\mathsf{EQ\text{-}EUF\text{-}CMA}}_{\mathsf{SPS\text{-}EQ},\mathcal{A}}(\lambda)$.

$\mathcal{H}_2$: We choose a random user identifier $j \leftarrow [n]$ and abort in case $\mathsf{GS.Open}(\mathsf{gmsk}, m^*, \sigma^*) \neq j$

It is easy to see that $\Pr[S_1] = n \cdot \Pr[S_2]$.

We now show that any adversary $\mathcal{A}$ that has non-negligible advantage in winning full-traceability experiment in $\mathcal{H}_2$ can be used by a reduction algorithm $\mathcal{R}$ to break the existential unforgeability of the SFPK scheme.

$\mathcal{R}$ computes all the public keys of group members according to protocol, except for user j. For this user, the algorithm sets fvk^j to the verification key given to $\mathcal{R}$ by the challenger in the unforgeability experiment of the SFPK scheme. It is worth noting, that the adversary $\mathcal{A}$ is given the group manager's secret key $\mathsf{gmsk} = ([(\tau^i, \mathsf{fvk}^i)]^n_{i=1})$. Fortunately, the reduction $\mathcal{R}$ is also given τ^j by the challenger and can compute a valid secret key gmsk that it gives as input to $\mathcal{A}$. To simulate signing queries for the j-th user, $\mathcal{R}$ uses its own signing oracle. By the change made in $\mathcal{H}_2$, $\mathcal{A}$ will never ask for the secret key of the j-th user, for which $\mathcal{R}$ is unable to answer (unlike for the other users).

Finally, $\mathcal{A}$ outputs a valid group signature $(m^*, \sigma^* = (\mathsf{fvk}^*, \sigma^*, \sigma^*_{\mathsf{EQ}}))$ and the reduction algorithm outputs $(m^*||\sigma^*_{\mathsf{EQ}}||\mathsf{fvk}^*, \sigma^*)$ as a valid SFPK forgery. By the changes made in the previous games we know that fvk^* and fvk^j must be in a relation. Moreover, the message m^* could not be used by $\mathcal{A}$ in any signing query made to $\mathcal{R}$. Thus, we know that $(m^*||\sigma^*_{\mathsf{EQ}}||\mathsf{fvk}^*)$ was never queried by $\mathcal{R}$ to its signing oracle, which show that $\mathcal{R}$ returns a valid forgery against the unforgeability of the SFPK scheme.

Finally, we have

$$\Pr[S_0] \leq n \cdot \mathsf{Adv}^{\mathsf{Flex\text{-}Unforgeability}}_{\mathcal{A},\mathsf{SFPK}}(\lambda) + \mathsf{Adv}^{\ell,\mathsf{EQ\text{-}EUF\text{-}CMA}}_{\mathsf{SPS\text{-}EQ},\mathcal{A}}(\lambda).$$

□

Proof of Anonymity

Theorem 3.9: Anonymity of Construction 3.5

Construction 3.5 is fully anonymous if the SPS-EQ signature scheme perfectly adapts signatures and is existentially unforgeable under chosen-message attacks and the SFPK scheme is class hiding and strongly existentially unforgeable.

We first use the perfect adaptation of SPS-EQ signatures to re-sign the verification key fvk' used in the challenge signature. Then we exclude the case that the adversary issues an open query that cannot be opened. This means that the adversary created a new group member and can be used to break the unforgeability of the SPS-EQ scheme. In the next step we choose one of the users (and abort if they are not part of the query issued by the adversary to the challenge oracle) for which we change the way we generate the signing key. Instead of using SFPK.tdGen, we use the standard key generation algorithm SFPK.KeyGen. Note that in such a case, the open oracle cannot identify signatures created by this user. However, since signatures cannot be opened by the oracle for this user we can identify such a case and return their identifier. Finally, we replace the SFPK verification key and signature in the challenge group signature by a random one (which is indistinguishable by class hiding). In the end the challenge signature is independent of the bit $\hat{b}$. However, the adversary still has non-zero advantage. This follows from the fact that it can randomize the challenge signature and our oracle will output $i_{\hat{b}}$ (because the SFPK verification key is random in the signature, the oracle will fail to open and return the user's identifier). However, if the adversary is able to submit such a query we can break the strong existential unforgeability of the SFPK scheme.

Proof. We will use the game-based approach. Let us denote by S_i the event that the adversary is successful in experiment $\mathcal{H}_i$.

$\mathcal{H}_0$: The original full-anonymity experiment.

$\mathcal{H}_1$: In this game we change the way we compute the challenge signature $\sigma^* \leftarrow \mathsf{GS.Sign}(\vec{\mathsf{gsk}}[i_b], m^*)$. Let $\sigma^* = (\mathsf{fvk}', \sigma, \sigma'_{\mathsf{EQ}})$. We compute (fvk', σ) as in the original experiment but instead of randomizing the SPS-EQ signature σ_{EQ}, we compute $\sigma_{\mathsf{EQ}} \leftarrow \mathsf{SPS.Sign}(\mathsf{fvk}', \mathsf{sk}_{\mathsf{SPS}})$.

Because the SPS-EQ signature scheme perfectly adapts signatures, we have $\Pr[S_1] = \Pr[S_0]$.

$\mathcal{H}_2$: We pick a random user identifier $j \leftarrow [n]$ and abort in case $j \neq i_b$.

It is easy to see that $\Pr[S_1] = n \cdot \Pr[S_2]$.

$\mathcal{H}_3$: We now abort in case the adversary queries a valid signature $(m, \sigma = (\mathsf{fvk}', \sigma, \sigma'_{\mathsf{EQ}}))$ to the Open oracle and it fails to open, i.e. the opening algorithm returns $\perp$.

By perfect correctness of the SFPK scheme, it follows that the only way an adversary can make the experiment abort if it is able to create a new user, i.e. create a valid SPS-EQ signature under a public key fvk^* that is not in relation with any of the honest public keys. It follows that we can use such an adversary to break the existential unforgeability of the SPS-EQ signature scheme, i.e. we just use the signing oracle to generate all σ^i_{EQ} and return $(\mathsf{fvk}', \sigma'_{\mathsf{EQ}})$ as a valid SPS-EQ forgery.

It follows that $|\Pr[S_3] - \Pr[S_2]| \leq \mathsf{Adv}^{\ell,\mathsf{EQ\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{SPS\text{-}EQ}}(\lambda)$.

$\mathcal{H}_4$: We now change the way, we compute the signing key for user i_b. Instead of using $(\mathsf{fvk}^j, \mathsf{fsk}^j, \tau^j) \leftarrow \mathsf{SFPK.tdGen}(1^\lambda)$, we use $(\mathsf{fvk}^j, \mathsf{fsk}^j) \leftarrow \mathsf{SFPK.KeyGen}(1^\lambda)$.

Obviously, in such a case we cannot answer the Open queries for user j, as the value τ^j is unknown. However, we note that if the adversary's query (m, σ) is a valid group signature, then the Open must return a valid user identifier (because of the change in $\mathcal{H}_3$, we do not return $\perp$ in such a case). Therefore, if there exists no identifier $i \in [n]/\{j\}$ for which $\mathsf{SFPK.Check}(\tau^i, \mathsf{fvk}^i, \mathsf{fvk}') = 1$, we return j.

It is easy to note that this is just a conceptual change (because of the change in $\mathcal{H}_3$) and we have $\Pr[S_4] = \Pr[S_3]$.

$\mathcal{H}_5$: We now compute a random SFPK key pair $(\mathsf{fvk}, \mathsf{fsk}) \leftarrow \mathsf{SFPK.KeyGen}(1^\lambda)$, choose a randomizer r, compute verification key $\mathsf{fvk}' \leftarrow \mathsf{SFPK.MoveVk}(\mathsf{fvk}, r)$, signing key $\mathsf{fsk}' \leftarrow \mathsf{SFPK.MoveSk}(\mathsf{fsk}, r)$ and change the way we compute the challenged signature $\sigma = (\mathsf{fvk}', \sigma, \sigma_{\mathsf{EQ}})$ under message m. We set $M = m||\sigma_{\mathsf{EQ}}||\mathsf{fvk}'$ and run $\sigma \leftarrow \mathsf{SFPK.Sign}(\mathsf{fsk}', M)$. In other words, instead of using the secret of user i_b to generate the signature σ, we use a fresh key pair for this (i.e. a user from outside the system).

We note that any adversary that is able to distinguish between $\mathcal{H}_4$ and $\mathcal{H}_5$, can be used to break the class hiding property of the SFPK signature scheme. The reduction algorithm can just set one of the public keys from the class hiding challenge to be part of the verification key of the j-th user. In case, the signature given by the challenger in the class hiding game was created by this user, we are in $\mathcal{H}_4$. If it was created by the second user, then we are in $\mathcal{H}_5$. Of course, it might happen that the one of the users in the other group member (other than the j-th user) has a verification key from the same relation as the second user in the class hiding experiment. However, this event occurs with negligible probability and we omit it.

Lastly, we notice that the challenger in the class hiding experiment is given the random coins used to generate the signing key to the adversary. Thus, our reduction can reuse those coins and compute the signing key, which it can give to the distinguishing algorithm, as required to fully simulate the anonymity experiment.

It follows that $|\Pr[S_5] - \Pr[S_4]| \leq \mathsf{Adv}^{\mathsf{Class\text{-}Hiding}}_{\mathcal{A},\mathsf{SFPK}}(\lambda)$.

The above changes ensure that the challenged signature is independent of the user i_b, i.e. we use a random SFPK verification key and a freshly generated SPS-EQ signature on it. However, an adversary $\mathcal{A}$ can still use the way we implemented the Open in $\mathcal{H}_4$. Note that in case it is somehow able to randomize the signature $\sigma = (\mathsf{fvk}, \sigma, \sigma_{\mathsf{EQ}})$ and ask the Open oracle, then we will return i_b as the answer.

We will now show that the adversary cannot create a valid and distinct signature from $\sigma = (\mathsf{fvk}, \sigma, \sigma_{\mathsf{EQ}})$. Let $(m^*, \sigma^* = (\mathsf{fvk}^*, \sigma^*, \sigma^*_{\mathsf{EQ}}))$ be the query made by the adversary and σ^* is a randomized version of σ.

The first observation is that by the change made in $\mathcal{H}_5$, we must have that fvk and fvk^* are in a relation, otherwise the above attack does not work. Thus, we can use such an adversary to break the strong existential unforgeability of the SFPK signature scheme. Note that by the change made in $\mathcal{H}_5$, fvk is a fresh verification key and the reduction algorithm can use the one from the strong existential unforgeability game. Moreover, in order to generate σ, the reduction algorithm uses its signing oracle. Finally, the reduction algorithm returns $((m^*||\sigma^*_{\mathsf{EQ}}||\mathsf{fvk}^*), \sigma^*)$ as a valid forgery.

It is easy to see that in case $\mathsf{fvk} \neq \mathsf{fvk}^*$ or $\sigma_{\mathsf{EQ}} \neq \sigma^*_{\mathsf{EQ}}$, the reduction algorithm wins the strong existential unforgeability game. Thus, the only part of the group signature that the adversary could potentially change is σ. This is the SFPK signature and would mean that the adversary was able to create a new signature under the message asked by the reduction algorithm to the signing algorithm. However, the case that $\sigma \neq \sigma^*$ also means that the reduction algorithm breaks the strong existential unforgeability of the SFPK scheme. We conclude, $\Pr[S_5] = \mathsf{Adv}^{\mathsf{Flex\text{-}sUnforgeability}}_{\mathcal{A},\mathsf{SFPK}}(\lambda)$.

Finally, we have

$$\Pr[S_0] \leq n \cdot \left(\mathsf{Adv}^{\ell,\mathsf{EQ\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{SPS\text{-}EQ}}(\lambda) + \mathsf{Adv}^{\mathsf{Class\text{-}Hiding}}_{\mathcal{A},\mathsf{SFPK}}(\lambda) + \mathsf{Adv}^{\mathsf{Flex\text{-}sUnforgeability}}_{\mathcal{A},\mathsf{SFPK}}(\lambda)\right).$$

□

3.5.2 Ring Signatures

In ring signatures there is no trusted entity such as a group manager and groups are chosen ad hoc by the signers themselves. Thus, to certify ring members we use a non-interactive membership proof instead of a SPS-EQ signature. We require this proof to be perfectly sound even if the common reference string is generated the prover, in our case the signer. In other words, the actual ring signature is a SFPK signature (fvk', σ) and a proof π that there exists a verification key $\mathsf{fvk} \in \mathsf{R}$ that is in relation to the verification key fvk', i.e. the signer proves knowledge of the randomizer used to get fvk'. The signature's anonymity relies on the class hiding property of SFPK. Unfortunately, in the proof of anonymity, the reduction does not know a valid witness for proof π, since it does not choose the randomizer for the challenge signature. Thus, we extend the signer's public keys by a tuple of three group elements (A, B, C) and prove a disjunctive statement which allows the reduction to compute a valid proof π if (A, B, C) is a non-DDH tuple.

We can instantiate this scheme with a membership proof based on the $\mathcal{O}(\sqrt{\ell})$ size ring signatures by Chandran, Groth, Sahai [CGS07] and the perfectly sound proof system for NP languages by Groth, Ostrovsky, Sahai [GOS06]. The resulting membership proof is perfectly sound and of sub-linear size in the size of the set.

Let $\mathcal{L}_{\mathsf{pk}}$ be the set of statements of the form:

$$\{(\mathsf{fvk}', \mathsf{R}) \mid \exists (i, \delta, \mathsf{fvk}).(i, \mathsf{fvk}_i, \cdot) \in \mathsf{R} \wedge \mathsf{SFPK.MoveVk}(\mathsf{fvk}_i, \delta) = \mathsf{fvk}\}$$

Let further $\mathcal{L}_{\mathsf{non-DDH}}$ be the set of statements of the form:

$$\{\ \mathsf{R} \mid \exists(i, I).(i, \cdot, I) \in \mathsf{R} \wedge I \text{ is not a DDH tuple}\}$$

Then we define the following witness relation:

$$\mathcal{R}_{\mathsf{RS}} := \{((\mathsf{fvk}', \mathsf{R}), (i, \delta, \mathsf{fvk}, I)) \mid ((\mathsf{fvk}', \mathsf{R}), (i, \delta, \mathsf{fvk})) \in \mathcal{L}_{\mathsf{pk}} \vee (\mathsf{R}, (i, I)) \in \mathcal{L}_{\mathsf{non-DDH}}\}$$

Let

- $\mathsf{NIP}_{\mathsf{RS}} = (\mathsf{Prove}, \mathsf{Verify})$ be a non-interactive proof system for $\mathcal{R}_{\mathsf{RS}}$.
- $\mathsf{SFPK} = (\mathsf{KeyGen}, \mathsf{tdGenSign}, \mathsf{Verify}, \mathsf{MoveKeys}, \mathsf{Check})$ be a signature scheme with flexible public keys without setup.

Construction 3.6: Generic Ring Signature Scheme

```
RS.KeyGen($1^\lambda$)
  $(\mathsf{fsk}, \mathsf{fvk}) \leftarrow \mathsf{SFPK.KeyGen}(1^\lambda)$
  $I := (A, B, C) \leftarrow \mathbb{G}_1^3$
  $\mathsf{rsk} := \mathsf{fsk}$
  $\mathsf{rvk} := (\mathsf{fvk}, I)$
  return $(\mathsf{rsk}, \mathsf{rvk})$
```

```
RS.Verify($m, \varsigma, \mathsf{R}$)
  parse $\varsigma = (\mathsf{fvk}', \sigma_{\mathsf{SFPK}}, \pi, \mathsf{crs})$
  $\mathsf{x} := (\mathsf{fvk}', \mathsf{R})$
  if $\mathsf{NIP}_{\mathsf{RS}}.\mathsf{Verify}(\mathsf{x}, \pi)$ and
    $\mathsf{SFPK.Verify}(\mathsf{fvk}', m, \sigma_{\mathsf{SFPK}})$
   then return true
  else return false
```

```
RS.Sign($m, \mathsf{rsk}, \mathsf{R}$)
  $\delta \leftarrow \Delta$
  $\mathsf{fsk}' \leftarrow \mathsf{SFPK.MoveSk}(\mathsf{fsk}, \delta)$
  $\mathsf{fvk}' \leftarrow \mathsf{SFPK.MoveVk}(\mathsf{fvk}, \delta)$
  $\sigma_{\mathsf{SFPK}} \leftarrow \mathsf{SFPK.Sign}(\mathsf{fsk}', m||\mathsf{R})$
  $\mathsf{x} := (\mathsf{fvk}', \mathsf{R})$
  $\mathsf{w} := (i_{\mathsf{rsk}}, \delta, \mathsf{fvk}, \emptyset)$
  $\pi \leftarrow \mathsf{NIP}_{\mathsf{RS}}.\mathsf{Prove}(\mathsf{x}, \mathsf{w})$
  $\varsigma := (\mathsf{fvk}', \sigma_{\mathsf{SFPK}}, \pi)$
  return $\varsigma$
```

Correctness. Let $\lambda, \ell \in \mathbb{N}$, $m \in \mathcal{M}$ and $(\mathsf{rsk}^i, \mathsf{rvk}^i) \leftarrow \mathsf{RS.KeyGen}(1^\lambda)$ for all $i \in [\ell]$. Let $\mathsf{R} = (\mathsf{rvk}^1, \ldots, \mathsf{rvk}^\ell)$ and in particular

$$\mathsf{rsk}^s = \mathsf{fsk}$$
$$\mathsf{rvk}^s = (\mathsf{fvk}, (A, B, C)).$$

for some $\mathsf{rvk}^s \in \mathsf{R}$. A signature on m under rsk^s has the form

$$\varsigma = (\mathsf{fvk}', \sigma, \pi)$$

Because of the soundness of the proof system, π shows that $\mathsf{fvk}' \approx_{\mathcal{R}} \mathsf{fvk}$ via some $\delta \in \Delta$. From the correctness of the SFPK scheme it then follows that $\mathsf{SFPK.Verify}(\mathsf{fvk}', m, \sigma) =$ true, hence verification is successful.

Proof of Unforgeability

Theorem 3.10: Unforgeability of Construction 3.6

The generic construction of ring signatures presented in construction 3.6 is unforgeable with respect to insider corruption assuming the SFPK scheme is existentially unforgeable, $\mathsf{NIP}_{\mathsf{RS}}$ is perfectly sound and the decisional Diffie-Hellman assumption holds in $\mathbb{G}_1$.

In the proof we proceed as follows. We first fix all verification keys of honest users to contain only DDH tuples. This ensures that the forgery $\varsigma^* = (\mathsf{fvk}^*, \sigma^*, \pi^*)$ includes a perfectly sound proof for the first clause of the statement, i.e. there exists a verification key $\mathsf{fvk} \in \mathsf{R}$, which is in relation to fvk^* (all users in R must be honest). This enables us to break existential unforgeability of the SFPK scheme. Note that we have to guess the correct user to execute a successful reduction. Thus, the reduction has a loss of $1/n$, where n is the number of honest users.

Proof. We will use the game based approach to prove this theorem. The first change we do is to fix the instance I to be a DDH tuple. This way our reduction algorithm (as well as the adversary) must use a witness that fulfills the first part of the statement proven by Π. The next step is simple. The reduction algorithm translates this game to the existential unforgeability experiment of the SFPK scheme. Note that the reduction algorithm will choose one of the users at random and use the challenged verification key as the user's public key. For the other users, the reduction algorithm will use a randomly choose key pair. This allows the reduction to answer all corruption queries. More formally. Let us denote by S_i the event that the adversary wins the unforgeability w.r.t insider corruption experiment in $\mathcal{H}_i$.

$\mathcal{H}_0$: The experiment.

$\mathcal{H}_1$: We make a small change in the way we generate the instance I for the public keys of users. Instead of generating A, B, C as random elements of $\mathbb{G}_1$, we first chose

$a, b \leftarrow \mathbb{Z}_p^*$ and then set $A = g_1^a$, $B = g_1^b$ and $C = g^{a \cdot b}$.

It is obvious that this change only decreases the adversary's advantage by a negligible fraction. In particular any distinguishing adversary can be used to break the decisional Diffie-Hellman assumption. Moreover, note that since any DDH instance can be randomized (i.e. (A^r, B^r, C^r) is a DDH tuple if and only if (A, B, C) is a DDH tuple) we can apply this change to all honest users at once. Thus, we get $|\Pr[S_1] - \Pr[S_0]| \leq \mathsf{Adv}_{\mathcal{A}}^{\mathsf{DDH}-1}(\lambda)$.

We now show how to use any adversary $\mathcal{A}$ that has non-negligible advantage in winning the unforgeability w.r.t insider corruption experiment in $\mathcal{H}_1$ to create a reduction algorithm $\mathcal{R}$ that has non-negligible advantage in winning the existential unforgeability experiment of the SFPK scheme. Let us by l denote the total number of users in the unforgeability w.r.t insider corruption experiment. The reduction algorithm works as follows.

In the first step $\mathcal{R}$ chooses a random $j \leftarrow [l]$ and generates $(\mathsf{rsk}_i, \mathsf{rvk}_i) \leftarrow \mathsf{RS.KeyGen}(1^\lambda)$ for all $i \in [l]/\{j\}$. For the j-th user it uses the verification key $\mathsf{rvk}_j = \mathsf{fvk}_j$ given to it by the challenger in the existential unforgeability experiment for the SFPK scheme for relation $\mathcal{R}$. $\mathcal{R}$ is able to answer all corruption queries of $\mathcal{A}$, beside for the j-th user. However, we hope that the adversary chooses this user to be part of the ring R^* for which it has to output a forgery. In such a case the adversary cannot ask the corruption query for the secret key of this user. We will later calculate the corresponding probability of the adversary asking for the j-th user's key, but now we assume that in such a case the reduction $\mathcal{R}$ aborts. The reduction algorithm is also able to answer all signing queries. Note that for the j-th user instead of using the Sign algorithm, we choose a random $r \leftarrow \mathbb{Z}_p^*$ and query the signing oracle OSign^2 with input (m, r).

Finally, the adversary $\mathcal{A}$ outputs a ring signature $\varsigma^* = (\mathsf{fvk}^*, \sigma^*, \Pi^*, \mathsf{crs}_\Pi^*)$ under message m^* for ring R^*. The reduction returns (m^*, ς^*) as its forgery for the SFPK scheme. We will now calculate the success probability of $\mathcal{R}$. We first notice that by the change made in $\mathcal{H}_1$ and since the proof Π^* is perfectly sound, it follows that there exists a verification key $\mathsf{fvk} \in \mathsf{R}^*$ for which $(\mathsf{fvk}, \mathsf{fvk}^*) \in \mathcal{R}$. Finally, we have that the probability that $\mathsf{fvk} = \mathsf{fvk}_j$ is $1/l$, i.e. from the j-th user's public key. Note that in such a case the adversary will not ask for the j-th user public key.

It follows that

$$\begin{aligned}\Pr[S_1] \leq & l \cdot \mathsf{Flex\text{-}Unforgeability}_{\mathsf{SFPK},\mathcal{R}}^{\mathcal{A}}(\lambda), \text{ and}\\ \Pr[S_0] \leq & l \cdot \mathsf{Flex\text{-}Unforgeability}_{\mathsf{SFPK},\mathcal{R}}^{\mathcal{A}}(\lambda) + \mathsf{Adv}_{\mathcal{A}}^{\mathsf{DDH}}(\lambda).\end{aligned}$$

□

Proof of Anonymity

Theorem 3.11: Anonymity of Construction 3.6

The generic construction of ring signatures presented in construction 3.6 is anonymous against full key exposure assuming the SFPK scheme is fully class hiding and $\mathsf{NIP}_{\mathsf{RS}}$ system is computationally witness-indistinguishable.

In the proof we proceed as follows. We first fix all verification keys of honest users to contain only non-DDH tuples I. In the next step we randomly choose a fresh bit $\hat{b} \leftarrow \{0,1\}$ and use the witness for the tuple $I_{i_{\hat{b}}}$ in the challenge signature. Note that the proof is valid for both values of $\hat{b}$, but now the proof part is independent of the bit b. Next we change the SFPK verification key fvk' and signature σ returned as part of the challenge signature $\varsigma = (\mathsf{fvk}', \sigma', \pi)$. Again we choose a fresh bit $\hat{b} \leftarrow \{0,1\}$ and compute them using $\mathsf{fvk}' \leftarrow \mathsf{SFPK.MoveVk}(\mathsf{fvk}_{i_{\hat{b}}}, \delta)$, $\mathsf{fsk}' \leftarrow \mathsf{SFPK.MoveSk}(\mathsf{fsk}_{i_{\hat{b}}}, \delta)$ and $\sigma \leftarrow \mathsf{SFPK.Sign}(\mathsf{fsk}', m||\mathsf{R})$. Any adversary distinguishing this change can be used to break the class hiding property of the SFPK scheme. Finally, all elements of ς are independent of b and the adversary's advantage is zero.

Proof. Let us denote by S_i the event that the adversary wins the anonymity experiment in $\mathcal{H}_i$.

$\mathcal{H}_0$: The original experiment.

$\mathcal{H}_1$: We make a small change we compute the instance $I = (A, B, C)$ in all the public keys of users. Instead of choosing A, B, C at random from $\mathbb{G}_1$, we first choose $a, b \leftarrow \mathbb{Z}_p^*$ and then compute $A = g_1^a$, $B = g_1^b$, $C = g_1^{a \cdot b - 1}$. In other words, we make sure that I is not a DDH tuple.

Similar as in the proof for unforgeability, $|\Pr[S_1] - \Pr[S_0]| \leq \mathsf{Adv}_{\mathcal{A}}^{\mathsf{DDH\text{-}1}}(\lambda)$.

$\mathcal{H}_2$: We now change the witness that we use to compute the proof Π in the challenged signature ς. Instead of using the verification key fvk_{i_b}, we will use a witness for the second part of the statement. Note that by the change made in the previous game, all instances I in the public keys of honest users are non-DDH tuples. Moreover, instead of using the witness for the instance I_{i_b} (where b is the challenged bit b and i_b is the identifier of the user for which the experiment generates the signature), we will choose a random bit $\hat{b}$ and use the witness for instance $I_{i_{\hat{b}}}$. Note that the proof inside the signature ς is now valid and independent of the bit b.

Because the proof system is computational witness-indistinguishable, it follows that $|\Pr[S_2] - \Pr[S_1]| \leq \mathsf{Adv}^{\mathsf{WI}}_{\Pi,\mathcal{A}}(\lambda)$.

$\mathcal{H}_3$: We will now change the way we compute the signature $\varsigma = (\mathsf{fvk}', \sigma', \Pi, \mathsf{crs}_\Pi)$. In particular, we will change the way we compute fvk' and σ'. Instead of computing it them using

$$\begin{aligned} \mathsf{fvk}' &\leftarrow \mathsf{SFPK.MoveVk}(\mathsf{fvk}_{i_b}, r), \\ \mathsf{fsk}' &\leftarrow \mathsf{SFPK.MoveSk}(\mathsf{fsk}_{i_b}, r), \\ \sigma &\leftarrow \mathsf{SFPK.Sign}(\mathsf{fsk}', m||\mathsf{R}), \end{aligned}$$

we will choose a fresh random bit $\hat{b}$ and compute it as

$$\begin{aligned} \mathsf{fvk}' &\leftarrow \mathsf{SFPK.MoveVk}(\mathsf{fvk}_{i_{\hat{b}}}, r), \\ \mathsf{fsk}' &\leftarrow \mathsf{SFPK.MoveSk}(\mathsf{fsk}_{i_{\hat{b}}}, r), \\ \sigma &\leftarrow \mathsf{SFPK.Sign}(\mathsf{fsk}', m||\mathsf{R}). \end{aligned}$$

We now show that any adversary $\mathcal{A}$ that has non-negligible advantage in distinguishing the difference between games 2 and 3, can be used as part of a reduction algorithm $\mathcal{R}$ that breaks the class hiding property of the SFPK scheme. Let us by l denote the total number of users in the anonymity experiment. The reduction first chooses $j, k \leftarrow [l]$ and generates $(\mathsf{rsk}_i, \mathsf{rvk}_i) \leftarrow \mathsf{RS.KeyGen}(1^\lambda, r_i)$ for all $i \in [l]/\{j, k\}$. Let (ω_0^*, ω_1^*) be the random coins given to $\mathcal{A}$ by the class hiding challenger. The reduction $\mathcal{R}$ runs $(\mathsf{fsk}_0, \mathsf{fvk}_0) \leftarrow \mathsf{SFPK.KeyGen}(1^\lambda, \omega_0^*)$ and $(\mathsf{fsk}_1, \mathsf{fvk}_1) \leftarrow \mathsf{SFPK.KeyGen}(1^\lambda, \omega_1^*)$. Then it computes random (A_0, B_0, C_0) and (A_1, B_1, C_1) as in $\mathcal{H}_1$ and the corresponding random coins ω_{I_0} and ω_{I_1}. It then sets $r_i = (\omega_0^*, \omega_{I_0})$, $r_k = (\omega_1^*, \omega_{I_1})$ and gives $\{r_i\}_{i=1}^{l}$ to $\mathcal{A}$. The adversary now outputs $(m, i_0, i_1, \mathsf{R})$. The reduction $\mathcal{R}$ aborts in case $i_0, i_1 \notin \{j, k\}$. Note that since, $\mathcal{A}$ advantage is non-negligible, we have that $i_0 \neq i_1$, $i_0 \in \mathsf{R}$ and $i_1 \in \mathsf{R}$. $\mathcal{R}$ then forwards $m||\mathsf{R}$ to the class hiding challenger and receives a SFPK signature σ' under the randomized verification key fvk'. The reduction computes the ring signature as $\varsigma = (\mathsf{fvk}', \sigma', \Pi, \mathsf{crs}_\Pi)$, where Π is a proof computed as in $\mathcal{H}_2$. Obviously, the success of $\mathcal{R}$ depends on the probability of guessing the correct identifiers i_0 and i_1. The probability is greater than $\frac{2}{l^2}$.

It follows that $|\Pr[S_3] - \Pr[S_2]| \leq \frac{l^2}{2} \cdot \mathsf{Adv}^{\mathsf{Class\text{-}Hiding}}_{\mathcal{A},\mathsf{SFPK}}(\lambda)$.

We now notice that the only value that depends on the challenged bit b in the original game is the ring signature $\varsigma = (\mathsf{fvk}', \sigma', \Pi, \mathsf{crs}_\Pi)$. By the changes we made in $\mathcal{H}_2$, the values (Π, crs_Π) are independent of b. What is more, by the changes made in $\mathcal{H}_3$, the values (fvk', σ') are also independent of b. It follows that:

$$\Pr[S_3] = 0$$
$$\Pr[S_0] \leq \frac{l^2}{2} \cdot \mathsf{Adv}^{\mathsf{Class\text{-}Hiding}}_{\mathcal{A},\mathsf{SFPK}}(\lambda) + \mathsf{Adv}^{\mathsf{WI}}_{\Pi,\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{DDH-1}}_{\mathcal{A}}(\lambda).$$

□

3.5.3 Practical Instantiations

In this section we discuss how to instantiate the generic group signature construction 3.5 and the generic ring signature construction 3.6 with our SFPK instantiations.

Note that in the case of group signatures we can use a SFPK scheme that is strongly existentially unforgeable in the multi-user setting, since the group manager can be trusted to perform a setup of public parameters. Thus, a natural candidate is construction 3.4. We also require a SPS-EQ signature scheme, which we instantiate using the scheme presented in [FG18]. A caveat to this scheme is that it only supports a one-time adaptation of signatures to a different representative. This does not impact our use of the scheme since in our application the group member performs the adaptation only once per signing. Further, the scheme is only unforgeable under adaptive chosen-*open*-message attacks, hence we require the following lemma.

Lemma 3.12: Security of Construction 3.5

Let the verification key of the SFPK scheme consist only of elements sampled directly from $\mathbb{G}_1$ or computed as g_1^x, where $x \leftarrow \mathbb{Z}_p^*$. Theorem 3.8 and theorem 3.9 still hold if the SPS-EQ scheme is only existential unforgeable under adaptive chosen-open-message attacks.

Sketch. In the proof of theorem 3.8, instead of excluding the case where the adversary creates a new user, we can toss a coin and chose the adversary's strategy (forging the SPS-EQ or SFPK signature). In case we end up choosing the SPS-EQ, we can freely choose the SFPK public keys and issue signing oracles to get all σ^i_{EQ}. In the proof of theorem 3.9 we use the unforgeability of SPS-EQ to exclude the case that the adversary issues an open query for a new user. Because this is the first change, we can again freely choose the SFPK public keys and issue signing oracles to get all σ^i_{EQ}. Finally, we note that in such proofs we make a non-blackbox use of the SFPK scheme. □

For message space $(\mathbb{G}_1^*)^\ell$ the size of the SPS-EQ signature is $(4 \cdot \ell + 2)$ elements in $\mathbb{G}_1$ and 4 elements in $\mathbb{G}_2$. The security of the SPS-EQ scheme relies on the decisional linear assumption and the decisional Diffie-Hellman assumption in $\mathbb{G}_2$. The security of our SFPK relies on the bilinear decisional Diffie-Hellman assumption. All in all, the proposed instantiation yields a static group signature scheme that is secure under standard assumptions and has a signature size of 15 elements in $\mathbb{G}_1$ (counting elements in $\mathbb{Z}_q^*$ as $\mathbb{G}_1$) and 5 elements in $\mathbb{G}_2$. It therefore has shorter signatures than the current state-of-the-art scheme in [LPY15].

Even shorter signatures can be achieved at the expense of introducing stronger assumptions without relying on Lemma theorem 3.12, by using the scheme found in [FHS14], which is unforgeable in the generic group model and has signatures of size 2 elements in $\mathbb{G}_1$ and 1 element in $\mathbb{G}_2$.

We now focus on instantiating our ring signatures construction. Combining construction 3.3 with a generic perfectly sound proof system would result in a ring signature scheme that is unlikely to be of interest, as there are already more efficient schemes with or without a trusted setup. However, using the results presented by Chandran, Groth and Sahai [CGS07] we can make the membership proof efficient. In the context of a ring signature scheme with setup, they propose a perfectly sound proof of size $\mathcal{O}(\sqrt{n})$ that a verification key $\mathsf{rvk} \in \mathbb{G}_1$ (or $\mathsf{rvk} \in \mathbb{G}_2$), is contained in a ring R of size n. The proof itself does not require trusted setup, however, hence this idea can be applied to arbitrary public keys (i.e. consisting of group elements in different groups) in combination with a perfectly sound proof system for NP languages. A verification key of construction 3.3 contains an element in $\mathbb{G}_T$ and therefore cannot be used with the proof system from section 3.2.3, which is based on the efficient Groth-Sahai proofs for pairing product equations. We solve this problem in the following way:

Lemma 3.13: Extended Public Keys

Construction 3.3 is unforgeable and class hiding even if $X = g_1^x$, $Y = g_2^y$ are publicly known, where $t = e(X^y, g_2) = e(X, Y)$ is part of the signer's public key. Moreover, knowing the signing key one can compute such values.

Proof. Class hiding still holds, because the adversary is given the secret keys fsk_i for $i \in \{0, 1\}$, which contain X_i and y_i, so it can compute X_i and Y_i by itself already. To show that unforgeability still holds, we first have to note that Y is part of the trapdoor τ and does not provide new information for the adversary. Finally, in the proof of unforgeability of construction 3.3 X is set to be g_1^γ, where g_1^γ is part of the decisional linear problem instance. This element is not given to the adversary directly but the same proof works if this value would be given to the adversary. $\square$

The idea is that instead of putting the verification key $\mathsf{fvk} = (t, A, B, C, D, K_{\mathsf{PHF}})$ into the ring, we put $(A, B, C, D, X, Y, K_{\mathsf{PHF}})$. Finally, we modify the first part of the statement proven during signing, i.e. we use

$$\begin{aligned}\exists_{A,B,C,D,X,X',Y,K_{\mathsf{PHF}},r}\ (i,(A,B,C,D,X,Y,K_{\mathsf{PHF}}),\cdot)\in\mathsf{R}\ \wedge\ e(X,g_2^r)=e(X',g_2)\ \wedge\\ e(X',Y)=t'\ \wedge\ e(A,g_2^r)=e(A',g_2)\ \wedge\\ e(B,g_2^r)=e(B',g_2)\ \wedge\ e(C,g_2^r)=e(C',g_2)\ \wedge\\ e(D,g_2^r)=e(D',g_2)\ \wedge\ e(K_{\mathsf{PHF}},g_2^r)=e(K'_{\mathsf{PHF}},g_2),\end{aligned}$$

instead of $\exists_{\mathsf{fvk},r}\ ((i,\mathsf{fvk},\cdot)\in\mathsf{R}\ \wedge\ \mathsf{SFPK.MoveVk}(\mathsf{fvk},r)=\mathsf{fvk}')$, where $\mathsf{fvk}'=(t',A',B',C',D',K'_{\mathsf{PHF}})$ is the randomized SFPK verification key used as part of the ring signature. Since all elements in the ring are now elements in $\mathbb{G}_1$ or $\mathbb{G}_2$, we can use the proof system from section 3.2.3 to efficiently instantiate the proof used in our ring signature construction. What is more, we can also apply the trick from [CGS07] and create a membership proof of length only $\mathcal{O}(\sqrt{n})$. The resulting ring signature scheme is the first efficient scheme that is secure under falsifiable assumptions, without a trusted party and with signature size that depends sub-linearly on the number of ring members. This solves the open problem stated by Malavolta and Schröder [MS17].

3.6 Related Work

There exist many primitives that allow for a limited malleability of the signed message. Homomorphic signatures [Bon+09] allow to sign any subspace of a vector space. In particular, given a number of signatures σ_i for vectors $\vec{v}_i$, everyone can compute a signature of $\sum_i \beta_i \cdot \vec{v}_i$ for scalars β_i.

Chase et al. [Cha+14] discussed malleable signatures, which allow any party knowing a signature of message m to construct a signature of message $m' = T(m)$ for some defined transformation T. One can consider malleable signatures as a generalization of quotable [ALP13] and redactable signatures [Joh+02].

Signatures on randomized ciphertexts by Blazy et al. [Bla+11] allow any party that is given a signature on a ciphertext to randomize the ciphertext and adapt the signature to maintain public verifiability.

Verheul [Ver01] introduces so-called self-blindable certificates. The idea is to use the same scalar to randomize the signature and corresponding message. Verheul proposed that one can view the message as a public key, which allows to preserve the validity of this “certificate” under randomization/blinding. However, the construction does not yield a secure signature scheme.

As noted above, all the mentioned works consider malleability of the message space. In our case we consider malleability of the key space. In this regard, signatures with re-randomizable keys introduced by Fleischhacker et al. [Fle+16] are a related primitive. They allow a re-randomization of signing and verification keys such that re-randomized keys share the same distribution as freshly generated keys and a signature created under a randomized key can be verified using an analogously randomized verification key.

They also define a notion of unforgeability under re-randomized keys, which allows an adversary to learn signatures under the adversaries' choice of randomization of the signing key under attack. The goal of the adversary is to output a forgery under the original key or under one of its randomizations. Regular existential unforgeability for signature schemes is a special case of this notion, where the attacker does not make use of the re-randomization oracle.

The difference to signatures with flexible public keys is that re-randomization in [Fle+16] is akin to sampling a fresh key from the space of all public keys, while changing the representative in our case is restricted to the particular key's equivalence class. Note that one might intuitively think that signatures under re-randomizable keys are just signatures with flexible keys where there is only one class of keys since re-randomizing is indistinguishable from fresh sampling. In this case class hiding would be perfect. However, such a scheme cannot achieve unforgeability under flexible keys, since it would be enough for an attacker to sample a fresh key pair and use a signature under that key as the forgery.

In a concurrent and independent line of work to [Bac+18], Lysyanskaya and Crites [CL19] develop mercurial signatures which are also similar to SPS-EQ and allow randomization of both messages and public keys. They show its applications to anonymous credentials. The key difference between mercurial signatures and our work is that mercurial signatures do not consider the possibility of a trapdoor key generation as described here for signatures with flexible public keys. As a result the only known constructions for mercurial signatures have security guarantees in the generic group model, whereas we show signatures with flexible public keys from standard assumptions.

4 Membership Privacy for Fully Dynamic Group Signatures

In this chapter:

Publication History

The contributions of this chapter are based on and were first published in

> Michael Backes, Lucjan Hanzlik, and Jonas Schneider-Bensch. "Membership Privacy for Fully Dynamic Group Signatures." In: *ACM CCS 2019: 26th Conference on Computer and Communications Security*. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. ACM Press, Nov. 2019, pp. 2181–2198. DOI: 10.1145/3319535.3354257.

In this chapter, we have modified the presentation of the fully dynamic group signature model to ease understanding.

4.1 Introduction

The first formal security model for group signatures, called the *static model*, was given by Bellare, Micciancio and Warinschi (BMW) in [BMW03], who also provided a construction from general assumptions. In the static model, which we have restated in chapter 2, all members of the group have to be specified during the setup phase of the scheme and the group manager generates and distributes their signing keys. Further, the group manager is also responsible for opening signatures.

Later models, notably the ones due to Bellare, Shi, and Zhang (BSZ) [BSZ05], as well as Kiayias, and Yung (KY) [KY05b; KY06], which we will subsume under the term *dynamic models*, generalized the static model in terms of functionality as well as security considerations:

- Where the static model requires all potential group members to be known at setup time, the dynamic models allow dynamic enrollment to the group after the group has been created. Unlike the static model, user keys are generated via a join/issue protocol, where users jointly generate their signing keys in interaction with the authorities. This means in particular that the issuing authority may not learn the user signing key.

- Where the static model places strong trust assumptions on the group manager by letting them handle users' signing key generation and opening, the dynamic models split the group manager into separate issuing and opening (or tracing) authorities. This strengthens security guarantees by allowing the model to incorporate malicious behavior on the part of either of the authorities.

Regarding the opening authority, it has to be ensured that the opening is honestly created. Otherwise, a malicious opener or even a malicious member of the group could produce a dishonest opening, that identifies a wrong signer either to claim a specific signature for themselves, or blame a user for a signature which they did not create. To this end, the opening is no longer simply a pointer to an entry in a group membership list, but is typically realized via a proof, that can be publicly verified. The validity of this proof is covered by the non-frameability requirement of the dynamic models. Sakai et al. [Sak+12] additionally define a property called opening soundness, which, if achieved, ensures that it is infeasible to create an opening which points to any but the actual signer of a valid signature.

A further extension of the dynamic models was recently proposed by Bootle et al. [Boo+16]. Their model, which we subsequently call the *fully dynamic model*, additionally addresses revocation of group membership, incorporates opening soundness and considers security even under maliciously generated keys. To model the dynamic nature of addition and revocation of members, the scheme's lifetime is partitioned into a series of epochs such that changes in the group membership can only happen during the transition from one epoch to the next. Since the issuing authority still decides who may join the group and who has to leave, the group's public information is updated by the issuing authority for each new epoch. The authors show that their model is general enough to capture all previously proposed notions, making it the most expressive model of the security of group signatures to date.

4.1.1 Contributions in This Chapter

In this chapter we revisit the fully-dynamic group signature framework by Bootle et al. [Boo+16]. We observe that the epoch information published with each modification of the group (joining or leaving of a member) may leak the identities of members. For instance in the scheme proposed in [Boo+15], where the epoch information contains a list of active members, this information is required to verify a signature. This is a major issue that limits the applications of group signatures and introduces real-world privacy risks that are not captured by the security model. In particular, let us consider the use of group signatures as part of a corporate/governmental access control system. In such a scenario group signatures protect access patterns. On the other hand, leaking a list of active members of the group can be used by potential adversaries to perform targeted attacks, e.g. bribery attempts, phishing attacks on private emails or denial of service attacks. An application that was impossible to formalize using previous definitions are private groups that can be used to create an electronic authentication method for private club members. Members of the club are unknown to the public and other members of the club but the group signatures allows a way to prove membership if required.

Therefore, as our first contribution for this chapter, we propose a new security notion for fully dynamic group signatures, namely *membership privacy*. Informally, when a group signature scheme offers membership privacy it means that an external observer cannot tell who joined or left the group in a given epoch, even if a subset of the group's members is controlled by the observer.[1] The possibility of membership privacy changes the meaning of a group signature to the external public compared to the previous models. The public may still verify that the signature was created by a party which received signing capabilities from the issuing authority, but not only is there no indication who the signer was specifically, but even the group of potential signers is hidden. As a consequence, to an external viewer, the group signature scheme is a way for the issuing authority to dynamically delegate signing capabilities to anonymous signers, who can be held *privately accountable* by the opening authority. In extending the model of Bootle et al. [Boo+16] we give formal definitions of join and leave privacy, which taken

[1]A similar property was recently put forward by Baldimtsi et al. [Bal+17] for the security of cryptographic accumulators, which are one of the building block of revocation systems for anonymous credentials. Although based on similar real-world concerns, their definition is specific to cryptographic accumulators and cannot be easily applied to group signatures.

together achieve membership privacy in the most expressive model of group signature security to date.

Our second contribution is a generic construction of fully-dynamic group signatures with membership privacy. Our scheme is built upon novel techniques in the area of signature with flexible public keys (SFPK) and their fruitful combination with signatures on equivalence classes (SPS-EQ). The former primitive allows signing keys to be re-randomized within a system of equivalence classes, while the second allows the same for messages and signatures. We build upon the idea, introduced in chapter 3, to use the combination of SFPK and SPS-EQ schemes with *compatible* systems of equivalence classes in the construction of highly efficient privacy-preserving signature schemes. Each epoch the issuing authority uses a fresh instance of SPS-EQ to certify the public keys of members, which live in SFPK equivalence classes. However, instead of using the original public keys in the epoch information, the group manager first randomizes the verification key and encrypts the randomization using the signer's public key for an encryption scheme. Members can decrypt the randomization and use the SPS-EQ signature from the epoch information. Additionally, the signer creates a proof of knowledge of a unique representative of the equivalence class and the randomness used by the signer. This unique representative can be extracted by the tracing authority and used to identify the signer because the unique representative is also used as the signer's global public key. Membership privacy is ensured because the issuing authority randomizes the published verification key list.

Lastly we show how to optimize our generic construction and efficiently instantiate it under standard assumptions without relying on the random oracle model. The resulting scheme has shorter signatures than state-of-the-art schemes [LPY15; GS08] that are secure under similar assumptions but only allow for partially-dynamic groups. To achieve this efficiency we make use of SFPK scheme that provides *canonical representative* (cf. definition 3.18).

The results presented in this chapter can thus be summarized as follows.

- We extend the existing definitions of Bootle et al. [Boo+16] and show that membership privacy can be seamlessly integrated in the previous security models for fully dynamic group signatures.

- We devise a generic construction of fully-dynamic group signatures with membership privacy that can be instantiated in the standard model.

- We employ a novel technique for the conjunction of SFPK and SPS-EQ, allowing us to build a highly efficient standard model group signature schemes along the lines of our generic construction but with shorter signature size than even state-of-the-art non-private schemes with comparable assumption. This underlines that membership privacy need not come at additional cost.

4.2 Chapter Preliminaries

In this section, we introduce preliminaries relevant in the constructions and arguments of this chapter. In particular, we give a detailed description of the formal model of fully dynamic group signatures as established by Bootle et al. [Boo+16].

4.2.1 The Fully Dynamic Group Signature Model

Before giving the formal definitions, we give an informal overview of the operation of a fully dynamic group signature scheme.

Key Management and Group Authorities. A fully dynamic group signature scheme has two designated authorities, an issuing authority $\mathcal{G}$ and a tracing authority $\mathcal{T}$, as well as arbitrarily many users, identified by numeric uids. The issuing and tracing authorities (possibly non-interactively, possibly after completing a common setup procedure $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$) compute their key pairs

$$((\mathsf{mpk}, \mathsf{msk}), (\mathsf{tpk}, \mathsf{tsk})) \leftarrow \langle \mathsf{KeyGen}_{\mathcal{G}}(\mathsf{pp}), \mathsf{KeyGen}_{\mathcal{T}}(\mathsf{pp})\rangle$$

and users compute their own keys

$$(\mathsf{usk}[\mathsf{uid}], \mathsf{upk}[\mathsf{uid}]) \leftarrow \mathsf{UserKeyGen}(1^\lambda).$$

Dynamic Group Membership via Epochs. Group membership is segmented into epochs η which are managed by the issuing authority, which publishes a piece of information info_η for each new epoch. We assume the epoch number τ is encoded in this information. A user can join the group for the next epoch by executing a

$$\langle \mathsf{Join}(\mathsf{info}_\eta, \mathsf{gpk}, \mathsf{uid}, \mathsf{usk}[\mathsf{uid}]), \mathsf{Issue}(\mathsf{info}_\eta, \mathsf{reg}, \mathsf{msk}, \mathsf{uid}, \mathsf{upk}[\mathsf{uid}])\rangle$$

procedure with the issuing authority, where $\mathsf{gpk} = (\mathsf{pp}, \mathsf{mpk}, \mathsf{tpk})$ is the group public key, thereby obtaining secret group signing keys $\vec{\mathsf{gsk}}[\mathsf{uid}]$ and advancing the current epoch to η_+.

Signing for an epoch. The user may then create signatures for the new epoch using the $\sigma \leftarrow \mathsf{Sign}(\mathsf{gpk}, \vec{\mathsf{gsk}}[\mathsf{uid}], \mathsf{info}_{\eta_+}, m)$ algorithm, and this signature may be publicly verified for η_+ using $\mathsf{Verify}(\mathsf{gpk}, \mathsf{info}_{\eta_+}, m, \sigma)$.

Publicly Verifiable Tracing. In case of abuse, the tracing authority is equipped with a registration table reg to produce a tracing proof

$$(\mathsf{uid}, \pi) \leftarrow \mathsf{Trace}(\mathsf{gpk}, \mathsf{tsk}, \mathsf{info}_\eta, \mathsf{reg}, m, \sigma)$$

which can be publicly verified using $\mathsf{Judge}(\mathsf{gpk}, \mathsf{uid}, \mathsf{info}_{\eta_+}, \pi_{\text{Trace}}, \mathsf{upk}[\mathsf{uid}], m, \sigma)$.

The interfaces of all parties described above are defined as follows. We recall the framework of definitions for fully dynamic group signatures established in [Boo+16].

Definition 4.1: Fully Dynamic Group Signature

A fully dynamic group signature scheme DGS is defined by the following set of efficient algorithms

$\mathsf{Setup}(1^\lambda)$:

On input a security parameter, the setup algorithm outputs public parameters pp and initializes the user registration table reg.

$\langle \mathsf{KeyGen}_{\mathcal{G}}(\mathsf{pp}), \mathsf{KeyGen}_{\mathcal{T}}(\mathsf{pp})\rangle$:

Given the public parameters pp the issuing authority $\mathcal{G}$ and tracing authority $\mathcal{T}$ jointly execute a key generation protocol.

- The private output of the issuing authority is a secret manager key msk, its public output a manager public key mpk and the initial group information info.
- The private output of the tracing authority is a secret tracing key tsk and a tracing authority public key tpk.

The public outputs together are referred to as the group public key gpk $:= (\mathsf{pp}, \mathsf{mpk}, \mathsf{tpk})$.

$\mathsf{UserKeyGen}(1^\lambda)$:

On input the public parameters, the user key generation algorithm outputs a pair of user secret and user public key $(\mathsf{usk}[\mathsf{uid}], \mathsf{upk}[\mathsf{uid}])$, bound to a fresh user ID uid > 0.

$\langle \mathsf{Join}(\mathsf{info}_\tau, \mathsf{gpk}, \mathsf{uid}, \mathsf{usk}[\mathsf{uid}]), \mathsf{Issue}(\mathsf{info}_\tau, \mathsf{reg}, \mathsf{msk}, \mathsf{uid}, \mathsf{upk}[\mathsf{uid}])\rangle$:

A user who has executed UserKeyGen, obtaining a user ID uid and key pair $(\mathsf{usk}[\mathsf{uid}], \mathsf{upk}[\mathsf{uid}])$ may, given the group public key and information regarding the current epoch info_τ engage the issuing authority in a join-issue procedure to become a member of the group. If successful, the output of the Issue algorithm is user registration information which is stored in reg[uid] the user signing key $\vec{\mathsf{gsk}}[\mathsf{uid}]$ is updated with the output of Join.

$\mathsf{Update}(\mathsf{gpk}, \mathsf{msk}, \mathsf{info}_{\eta_{now}}, R, \mathsf{reg})$:

The issuing authority may advance the current epoch η_{now} to the next epoch η_+, at the same time revoking membership of a subset R of the set of active group members. If any uid $\in R$ is not assigned to an active member of the group, i.e. was not assigned in a run of the join-issue procedure, the algorithm aborts. The output is the new group information info_{η_+} and a possibly updated registration table reg. If the group information does not change, the algorithm outputs $\perp$.

$\mathsf{Sign}(\mathsf{gpk}, \vec{\mathsf{gsk}}[\mathsf{uid}], \mathsf{info}_\eta, m)$:
: Given their group signing key, current group information and the group public key, a user may sign a message, producing a signature σ. If the user-ID uid is not assigned to an active group member in the current epoch η_{now}, the algorithm outputs $\perp$ instead.

$\mathsf{Verify}(\mathsf{gpk}, \mathsf{info}_\eta, m, \sigma)$:
: If the given signature σ is valid for message m in epoch η, verification outputs true, otherwise false.

$\mathsf{Trace}(\mathsf{gpk}, \mathsf{tsk}, \mathsf{info}_\eta, \mathsf{reg}, m, \sigma)$:
: Given a signature, message, group information for epoch η and a registration table, the tracing authority may output a pair (uid, π) where uid > 0 identifies the user-ID of the group member who produced the signature and π is a proof of this fact. If tracing is not successful the algorithm will output a pair $(0, \pi)$ indicating the failure via the special user-ID 0, which is not assigned to any regular user.

$\mathsf{Judge}(\mathsf{gpk}, \mathsf{uid}, \mathsf{info}_\eta, \pi_{\mathsf{Trace}}, \mathsf{upk}[\mathsf{uid}], m, \sigma)$:
: Given a signature for epoch η, the corresponding group information and a tracing output (uid, π), anyone in possession of the group public key can deterministically judge the validity of π w.r.t. to the statement, that σ was created using $\vec{\mathsf{gsk}}[\mathsf{uid}]$, in which case the algorithm outputs true, otherwise false.

Remark

In addition to the interface above, we make explicit an algorithm for determining if a give user was active in a certain epoch that was treated as implicit in previous works.

$\mathsf{Active?}(\mathsf{info}_\eta, \mathsf{reg}, \mathsf{uid})$:
: If user uid is a member of the group in epoch η, return true, otherwise false.

Note that this algorithm is only available to parties with access to the registration table.

A fully dynamic group signature scheme is secure if it achieves the following properties:

- Correctness,
- Traceability,
- Anonymity,
- Non-frameability,
- and Tracing Soundness.

These properties are formally defined in the experiment-based style that is used throughout this thesis. Due to the large number of oracles available to the adversaries in each of the experiments we give below an informal overview of the functionality of these oracles instead of reprinting them next to each security experiment.

The experiments may further be stateful and keep lists of the attackers' actions to subsequently determine whether the attacker was successful or not. For reference, we give a short summary of the lists and their purposes below.

H: Honest users added via the AddUser oracle.

C: Users with maliciously generated keys, added via the CorruptUser oracle.

B: Users whose secret keys were revealed to the adversary via the Reveal oracle.

Q: Signature queries, this list is populated by the Sign oracle.

Q*: Signatures created by the challenge users, this list is populated by the Challenge oracle.

The full pseudocode for the oracles can be found in section 6.2.2.

AddUser(uid):
If uid is new to the system, run DGS.UserKeyGen(1^λ) to honestly generate the user's keys $(\mathsf{usk}[\mathsf{uid}], \mathsf{upk}[\mathsf{uid}])$ and add uid to **H**. Afterwards the honest key generation is run using DGS.Join and DGS.Issue. This sets the user group signing key $\vec{\mathsf{gsk}}[\mathsf{uid}]$ and the contents of the registration table reg[uid]. The oracle's output is the new epoch information info_τ and the user's public key upk[uid].

CorruptUser(uid, pk):
If uid is new to the system, set upk[uid] to the supplied key pk and add uid to **C**. Initiates a join-issue session for uid by setting $\mathsf{decision}^{\mathsf{uid}}_{\mathsf{Issue}} := \texttt{continue}$.

$\mathsf{SendM}(\mathsf{uid}, M_{\mathsf{in}})$:
If a join-issue session is running for corrupted user uid, compute the new session state and response $(\mathsf{state}^{\mathsf{uid}}_{\mathsf{Issue}}, M_{\mathsf{out}}, \mathsf{decision}^{\mathsf{uid}}_{\mathsf{Issue}})$ from the issuing authority using $\mathsf{Issue}(\tau, \mathsf{reg}, \mathsf{msk}, \mathsf{uid}, \mathsf{upk}[\mathsf{uid}], M_{\mathsf{in}})$. If the output of Issue contains $\mathsf{decision}^{\mathsf{uid}}_{\mathsf{Issue}} = \mathtt{accept}$, update the registration table $\mathsf{reg}[\mathsf{uid}] := \mathsf{state}^{\mathsf{uid}}_{\mathsf{Issue}}$. The oracle's output is $(M_{\mathsf{out}}, \mathsf{decision}^{\mathsf{uid}}_{\mathsf{Issue}})$.

$\mathsf{SendU}(\mathsf{uid}, M_{\mathsf{in}})$:
If not already running initiate a join-issue session for a user, compute the new session state and response $(\mathsf{state}^{\mathsf{uid}}_{\mathsf{Join}}, M_{\mathsf{out}}, \mathsf{decision}^{\mathsf{uid}}_{\mathsf{Join}})$ from the user using $\mathsf{Join}(\tau, \mathsf{gpk}, \mathsf{uid}, \mathsf{usk}[\mathsf{uid}], M_{\mathsf{in}})$. If the output of Join contains $\mathsf{decision}^{\mathsf{uid}}_{\mathsf{Join}} = \mathtt{accept}$, set $\vec{\mathsf{gsk}}[\mathsf{uid}] := \mathsf{state}^{\mathsf{uid}}_{\mathsf{Join}}$. The oracle's output is $(M_{\mathsf{out}}, \mathsf{decision}^{\mathsf{uid}}_{\mathsf{Join}})$.

$\mathsf{ReadReg}(\mathsf{uid})$:
Return the registration table entry $\mathsf{reg}[\mathsf{uid}]$.

$\mathsf{ModifyReg}(\mathsf{uid}, \mathit{val})$:
Set registration table entry $\mathsf{reg}[\mathsf{uid}] := \mathit{val}$.

$\mathsf{Reveal}(\mathsf{uid})$:
Return the user secret keys $(\mathsf{usk}[\mathsf{uid}], \vec{\mathsf{gsk}}[\mathsf{uid}])$ and add uid to the set of bad users $\mathbf{B}$.

$\mathsf{Sign}(\mathsf{uid}, m, \eta)$:
If η is a valid epoch, and uid is active in that epoch, create a signature $\sigma \leftarrow \mathsf{DGS.Sign}(\mathsf{gpk}, \vec{\mathsf{gsk}}[uid], \mathsf{info}_\eta, m)$ and add $(\mathsf{uid}, m, \sigma, \eta)$ to the set of queried signatures $\mathbf{Q}$ and return σ.

$\mathsf{Trace}(m, \sigma, \mathsf{info}_\eta)$:
If the signature is valid in epoch η and is not part of the challenge set $\mathbf{Q}^*$, return the output of $\mathsf{DGS.Trace}(\mathsf{gpk}, \mathsf{tsk}, \mathsf{info}_\eta, \mathsf{reg}, m, \sigma)$.

$\mathsf{UpdateGroup}(R)$:
Run and return the output of $\mathsf{DGS.Update}(\mathsf{gpk}, \mathsf{msk}, \mathsf{info}_\tau, R, \mathsf{reg})$.

$\mathsf{Challenge}_b(\mathsf{info}_\eta, \mathsf{uid}_0, \mathsf{uid}_1, m)$:
If uid_0 and uid_1 are both active and honest in η, run the signing algorithm $\mathsf{DGS.Sign}(\mathsf{gpk}, \vec{\mathsf{gsk}}[\mathsf{uid}_b], \mathsf{info}_\eta, m)$ to obtain signature σ, adding (m, σ, η) to the challenge signature set $\mathbf{Q}^*$ and returning the signature.

We are now ready to give formal definitions for the security notions outlined above.

Correctness. Rather unusually, we have to consider correctness against adversarial manipulation of the group. In particular an adversary could, via some interleaving of joins and revocations corrupt the group configuration, such that verification correctness does not hold or tracing of honestly generated signatures is no longer possible.

A scheme is correct if this is not possible.

Definition 4.2: Correctness

For a fully dynamic group signature scheme DGS, consider the correctness experiment between a challenger and an adversary $\mathcal{A}$:

$\mathsf{Correctness}_{\mathsf{DGS}}^{\mathcal{A}}(1^\lambda)$

```
$\mathbf{H} := \emptyset$
$(\mathsf{reg}, \mathsf{pp}) \leftarrow \mathsf{DGS.Setup}(1^\lambda)$
$(\mathsf{msk}, \mathsf{mpk}, \mathsf{info}, \mathsf{tsk}, \mathsf{tpk}) \leftarrow \langle \mathsf{DGS.KeyGen}_{\mathcal{G}}(\mathsf{pp}), \mathsf{DGS.KeyGen}_{\mathcal{T}}(\mathsf{pp}) \rangle$
$\mathsf{gpk} := (\mathsf{pp}, \mathsf{mpk}, \mathsf{tpk})$
$(\mathsf{uid}, m, \eta) \leftarrow \mathcal{A}(\mathsf{gpk}, \mathsf{info})$
if $\mathsf{uid} \notin \mathbf{H}$ or $\vec{\mathsf{gsk}}[\mathsf{uid}] = \bot$ or $\mathsf{info}_\eta = \bot$
 or $\mathsf{DGS.Active?}(\mathsf{info}_\eta, \mathsf{reg}, \mathsf{uid}) = \mathsf{false}$
    then return false
$\sigma \leftarrow \mathsf{DGS.Sign}(\mathsf{gpk}, \vec{\mathsf{gsk}}[\mathsf{uid}], \mathsf{info}_\eta, m)$
if $\mathsf{DGS.Verify}(\mathsf{gpk}, \mathsf{info}_\eta, m, \sigma) = \mathsf{false}$
    then return true
$(\mathsf{uid}^*, \pi) \leftarrow \mathsf{DGS.Trace}(\mathsf{gpk}, \mathsf{tsk}, \mathsf{info}_\eta, \mathsf{reg}, m, \sigma)$
if $\mathsf{uid} \neq \mathsf{uid}^*$ then return true
if $\mathsf{DGS.Judge}(\mathsf{gpk}, \mathsf{uid}, \mathsf{info}_\eta, \pi, \mathsf{upk}[\mathsf{uid}], m, \sigma) = \mathsf{false}$
 then return false
else return true
```

$\mathcal{A}$ may query oracles AddUser, ReadReg, UpdateGroup at any point during its runtime.

We define the advantage of $\mathcal{A}$ in the above experiment as

$$\mathsf{Adv}_{\mathcal{A},\mathsf{DGS}}^{\mathsf{Correctness}}(\lambda) := \Pr\left[\mathsf{Correctness}_{\mathsf{DGS}}^{\mathcal{A}}(1^\lambda) \Rightarrow \mathsf{true}\right].$$

We say that DGS is *correct* if for all PPT adversaries $\mathcal{A}$, we have

$$\mathsf{Adv}_{\mathcal{A},\mathsf{DGS}}^{\mathsf{Correctness}}(\lambda) \leq \mathsf{negl}(\lambda).$$

Traceability. Any coalition of group members and the opening authority cannot produce a signature which would open to an identity not generated in the setup phase or an identity that was not active in the epoch for which the signature was created. Note, that this implies correctness.

Definition 4.3: Traceability

For a fully dynamic group signature scheme DGS, consider the traceability experiment between a challenger and a two-stage adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$:

```
$\mathsf{Tracing}^{\mathcal{A}}_{\mathsf{DGS}}(1^\lambda)$

$\mathbf{H}, \mathbf{C}, \mathbf{B}, \mathbf{Q} := \emptyset$
$(\mathsf{reg}, \mathsf{pp}) \leftarrow \mathsf{DGS.Setup}(1^\lambda)$
$(\mathsf{state}, \mathsf{tsk}, \mathsf{tpk}) \leftarrow \mathcal{A}_0^{\langle \mathsf{DGS.KeyGen}_{\mathcal{G}}(\mathsf{pp}), \cdot \rangle}(\mathsf{pp})$
if $\bot \leftarrow \mathsf{DGS.KeyGen}_{\mathcal{G}}(\mathsf{pp})$ or $\mathcal{A}$'s output invalid
    then return false
$(\mathsf{msk}, \mathsf{mpk}, \mathsf{info}) \leftarrow \mathsf{DGS.KeyGen}_{\mathcal{G}}(\mathsf{pp})$;
$\mathsf{gpk} := (\mathsf{pp}, \mathsf{mpk}, \mathsf{tpk})$
$(m, \sigma, \eta) \leftarrow \mathcal{A}_1(\mathsf{state}, \mathsf{gpk}, \mathsf{info})$
if $\mathsf{DGS.Verify}(\mathsf{gpk}, \mathsf{info}_\eta, m, \sigma) = \mathsf{false}$
 then return false
$(\mathsf{uid}, \pi) \leftarrow \mathsf{DGS.Trace}(\mathsf{gpk}, \mathsf{tsk}, \mathsf{info}_\eta, \mathsf{reg}, m, \sigma)$
if $\mathsf{DGS.Active?}(\mathsf{info}_\eta, \mathsf{reg}, \mathsf{uid}) = \mathsf{false}$ or $\mathsf{uid} = 0$
 or $\mathsf{DGS.Judge}(\mathsf{gpk}, \mathsf{uid}, \mathsf{info}_\eta, \pi, \mathsf{upk}[\mathsf{uid}], m, \sigma) = \mathsf{false}$
 then return true
else return false
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
$\mathcal{A}_1$ may query oracles AddUser, CorruptUser, SendM, Reveal, Sign,
ModifyReg, UpdateGroup at any point during its runtime.
```

We define the advantage of $\mathcal{A}$ in the above experiment as

$$\mathsf{Adv}^{\mathsf{Tracing}}_{\mathcal{A},\mathsf{DGS}}(\lambda) := \Pr\left[\mathsf{Tracing}^{\mathcal{A}}_{\mathsf{DGS}}(1^\lambda) \Rightarrow \mathsf{true}\right].$$

We say that DGS has *traceable signatures* if for all PPT adversaries $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathsf{Tracing}}_{\mathcal{A},\mathsf{DGS}}(\lambda) \leq \mathsf{negl}(\lambda).$$

Non-frameability. Any coalition of group members, the issuing authority and the opening authority cannot produce a signature which opens to an identity of an honest user from outside the coalition.

Definition 4.4: Non-frameability

For a fully dynamic group signature scheme DGS, consider the non-frameability experiment between a challenger and a two-stage adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$:

$\mathsf{Non\text{-}Frameability}^{\mathcal{A}}_{\mathsf{DGS}}(1^\lambda)$

```
$\mathbf{H}, \mathbf{C}, \mathbf{B}, \mathbf{Q} := \emptyset$
$(\mathsf{reg}, \mathsf{pp}) \leftarrow \mathsf{DGS.Setup}(1^\lambda)$
$(\mathsf{state}, \mathsf{info}, \mathsf{msk}, \mathsf{mpk}, \mathsf{tsk}, \mathsf{tpk}) \leftarrow \mathcal{A}_0(\mathsf{pp})$
if $\mathsf{msk} = \bot$ or $\mathsf{mpk} = \bot$
 then return false
$\mathsf{gpk} := (\mathsf{pp}, \mathsf{mpk}, \mathsf{tpk})$
$(m, \sigma, \mathsf{uid}, \pi, \mathsf{info}_\eta) \leftarrow \mathcal{A}_1(\mathsf{state}, \mathsf{gpk})$
if $\mathsf{DGS.Verify}(\mathsf{gpk}, \mathsf{info}_\eta, m, \sigma) = \mathsf{false}$
 or $\mathsf{DGS.Judge}(\mathsf{gpk}, \mathsf{uid}, \mathsf{info}_\eta, \pi, \mathsf{upk}[\mathsf{uid}], m, \sigma) = \mathsf{false}$
    then return false
if $\mathsf{uid} \in \mathbf{H} \setminus \mathbf{B}$ and $(\mathsf{uid}, m, \sigma, \eta) \notin \mathbf{Q}$
    then return true else return false
```

$\mathcal{A}_1$ may query oracles $\mathsf{CorruptUser}, \mathsf{Sign}, \mathsf{SendU}, \mathsf{Reveal}, \mathsf{ModifyReg}$ at any point during its runtime.

We define the advantage of $\mathcal{A}$ in the above experiment as

$$\mathsf{Adv}^{\mathsf{Non\text{-}Frameability}}_{\mathcal{A},\mathsf{DGS}}(\lambda) := \Pr\left[\mathsf{Non\text{-}Frameability}^{\mathcal{A}}_{\mathsf{DGS}}(1^\lambda) \Rightarrow \mathsf{true}\right]$$

We say that DGS provides *non-frameability* if for all PPT adversaries $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathsf{Non\text{-}Frameability}}_{\mathcal{A},\mathsf{DGS}}(\lambda) \leq \mathsf{negl}(\lambda)\,.$$

Anonymity. Given a signature, it is infeasible, without a secret trapdoor information, to distinguish which signer created the signatures.

Definition 4.5: Anonymity

For a fully dynamic group signature scheme DGS, consider the anonymity experiment between a challenger and a two-stage adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$:

$\mathsf{Anonymity}_{\mathsf{DGS}}^{\mathcal{A}}(1^\lambda)$

```
$(\mathsf{reg}, \mathsf{pp}) \leftarrow \mathsf{DGS.Setup}(1^\lambda); \mathbf{H}, \mathbf{C}, \mathbf{B}, \mathbf{Q}, \mathbf{Q}^* := \emptyset$
$(\mathsf{state}, \mathsf{msk}, \mathsf{mpk}, \mathsf{info}) \leftarrow \mathcal{A}_0^{\langle \cdot, \mathsf{DGS.KeyGen}_{\mathcal{T}}(\mathsf{pp}) \rangle}(\mathsf{pp})$
if $\perp \leftarrow \mathsf{DGS.KeyGen}_{\mathcal{T}}(\mathsf{pp})$ or $\mathcal{A}$'s output invalid
    then return false
$(\mathsf{tsk}, \mathsf{tpk}) \leftarrow \mathsf{DGS.KeyGen}_{\mathcal{T}}(\mathsf{pp})$; $\mathsf{gpk} := (\mathsf{pp}, \mathsf{mpk}, \mathsf{tpk})$
$d \leftarrow \mathcal{A}_1(\mathsf{state}, \mathsf{gpk})$
return $d = b$
```

$\mathcal{A}_1$ may query oracles $\mathsf{AddUser}, \mathsf{CorruptUser}, \mathsf{SendU}, \mathsf{Reveal}, \mathsf{Trace}$, $\mathsf{ModifyReg}$, $\mathsf{Challenge}_b$ at any point during its runtime.

We define the advantage of $\mathcal{A}$ in the above experiment as

$$\mathsf{Adv}_{\mathcal{A},\mathsf{DGS}}^{\mathsf{Anonymity}}(\lambda) := \left| \Pr\left[\mathsf{Anonymity}_{\mathsf{DGS}}^{\mathcal{A}}(1^\lambda) \Rightarrow \mathsf{true}\right] - \frac{1}{2} \right| .$$

We say that DGS is *anonymous* if for all PPT adversaries $\mathcal{A}$, we have

$$\mathsf{Adv}_{\mathcal{A},\mathsf{DGS}}^{\mathsf{Anonymity}}(\lambda) \leq \mathsf{negl}(\lambda) .$$

4.3 Extensions to the Fully Dynamic Model

In this section we present our extensions to the model presented in [Boo+16] and restated in section 4.2.

4.3.1 Functional Tracing Soundness

Even if all parties in the group collude, they cannot produce a valid signature that traces to two different members. This property is also called *opening soundness*.

A subtle point arises in the definition of tracing soundness, namely how is the uniqueness of group members established? If the adversary controls several users, they may share the same public key, hence their signatures cannot be distinguished by an opening which reveals the public key of the signer. Because of this, the opening instead leads to a specific user identity in the public registration table. This has two-fold consequences:

1. The user registration table has to be public, otherwise the opening is meaningless.
2. To verify an opening or even a signature, it has to be verified as well that the group at the time of the creation of the signature was well-formed, i.e. every member occupies exactly one slot in the registration table.

We propose a relaxation of this notion, which allows us to avoid these implications. Our notion, *functional tracing soundness* distinguishes members by their public keys, i.e. it should not be possible, even in a fully corrupted group to create a valid signature and two openings for it which indicate conflicting public keys.

We observe that the fully dynamic group signature scheme based on accountable ring signatures presented in [Boo+16] adheres to this definition already, since its proof of tracing soundness relies on the tracing soundness of the underlying accountable ring signature scheme. The property for accountable ring signature schemes requires that the verification keys provided in the two openings be different.

Note that the construction of fully dynamic group signatures presented later in this work can be made to achieve the original version of tracing soundness, albeit at the cost of the above-mentioned group integrity checks and any kind of group membership privacy.

Definition 4.6: Functional Tracing Soundness

For a fully dynamic group signature scheme DGS consider the tracing soundness experiment between a challenger and a two-stage adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$.

$\mathsf{Tracing\text{-}Soundness}^{\mathcal{A}}_{\mathsf{DGS}}(1^\lambda)$

```
$(\mathsf{reg}, \mathsf{pp}) \leftarrow \mathsf{Setup}(1^\lambda); \mathbf{C} := \emptyset$
$(\mathsf{state}, \mathsf{info}, \mathsf{msk}, \mathsf{mpk}, \mathsf{tsk}, \mathsf{tpk}) \leftarrow \mathcal{A}_0(\mathsf{pp})$
**if** $\mathsf{msk} = \bot$ **or** $\mathsf{mpk} = \bot$
    **then** **return** false
$\mathsf{gpk} := (\mathsf{pp}, \mathsf{mpk}, \mathsf{tpk})$
$(m, \sigma, \{\mathsf{uid}_i, \pi_i\}_{i=1}^{2}, \mathsf{info}_\eta) \leftarrow \mathcal{A}_1(\mathsf{state}, \mathsf{gpk})$
**if** $\mathsf{Verify}(\mathsf{gpk}, \mathsf{info}_\eta, m, \sigma) = \mathsf{false}$
    **then** **return** false
**if** $\mathsf{upk}[\mathsf{uid}_1] = \mathsf{upk}[\mathsf{uid}_2]$ **or** $\exists i \in \{1, 2\}$ s.t. $\mathsf{upk}[\mathsf{uid}_i] = \bot$
 **or** $\mathsf{Judge}(\mathsf{gpk}, \mathsf{uid}_i, \mathsf{info}_\eta, \pi_i, \mathsf{upk}[\mathsf{uid}_i], m, \sigma) = \mathsf{false}$
    **then** **return** false **else return** false
```

$\mathcal{A}_1$ may invoke the oracles CorruptUser, ModifyReg, at any point during its runtime.

We define the adversary's advantage in the Tracing-Soundness experiment as

$$\mathsf{Adv}^{\mathsf{Tracing\text{-}Soundness}}_{\mathcal{A},\mathsf{DGS}}(\lambda) := \Pr\left[\mathsf{Tracing\text{-}Soundness}^{\mathcal{A}}_{\mathsf{DGS}}(1^\lambda) \Rightarrow \mathsf{true}\right].$$

A group signature scheme DGS has *tracing soundness* if for all PPT adversaries $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathsf{Tracing\text{-}Soundness}}_{\mathcal{A},\mathsf{DGS}}(\lambda) \leq \mathsf{negl}(\lambda).$$

4.3.2 Membership Privacy in the Fully Dynamic Model

Formal models of dynamic group signatures thus far implicitly assumed that group membership is public information. Usually, a registration table is published, such that the entries are bound to public keys of the members. This is in line with one of the main application of group signatures: authenticating messages with the authority of a known group, certifying that *someone* within the group has seen the signed message and has taken responsibility on behalf of the group.

In their seminal work Chaum and van Heyst [Cv91], however, did not specify this as an essential requirement. In fact, they point out that group signatures can be used for access control, where knowing members of the group is an obvious privacy leak that could for instance lead to targeted DoS attacks on the group. Therefore, it seems natural that in some applications we want to hide the identities of active group members.

To address this issue we discuss for the first time *membership privacy* for group signatures. Informally, we will say that a group signature scheme has membership privacy if it protects the identity of users that join or leave the system. This means that we consider a scenario in which some kind of public identifier about users is known independently of the scheme (e.g. public key) but it is unknown to a third party who is part of the group. Moreover, we take into account that some users can be corrupted or can collude to infer information about the membership status of other users. Even in these cases, membership should remain private.

To formally define this notion, we propose a pair of security experiments which are expressed in the fully dynamic framework put forth by [Boo+16]. However, one can easily specify similar experiments for the partially dynamic models [BSZ05; KY05b; KY06]. The first one describes *join privacy*, since it considers the case that two non-members are known in one epoch and in the next epoch one of them joins the system and the task is to distinguish who joined the group. The second experiment describes *leave privacy* and models the case that there are two known members in one epoch and in the next epoch one of them leaves the group. Note that this assumes that the adversary knows out of band that the two users had previously joined the group. In both cases we allow an adversary to corrupt members of the group, but we consider both authorities to be honest: The issuing authority always knows who is part of the group and the tracing authority can open all signatures to extract the identities of members. In particular, this implies that the registration table reg may not be public because one could easily infer current members from it. Fortunately, this seems a fairly natural assumption. This registration table is not necessary in any of the user-centric algorithms, and it is easier to keep it local to the authorities than publishing it online. An exception is the scheme [Sak+12] mentioned above, where the registration table is part of the verification algorithm to ensure that tracing soundness holds with respect to public user identities rather than in the functional sense we describe.

A different question is whether additionally to the identities of users, we can hide the size of the group. Unfortunately, since the fully dynamic model in [Boo+16] allows joining and leaving the group, all efficient constructions fail to hide the size of the group. Whitelisting immediately leaks the size of the group and can only be alleviated using dummy users, which incurs large overhead and fixes a constant upper bound on the group size. This is even the case for cryptographic accumulators, where it is required by members to update their witness with every epoch. Thus, some kind of information that is linear is the number of active/inactive members must be published together with the accumulator.

We formally define join and leave privacy in terms of the two experiments shown below. Note, that we introduce a new set of privacy challenge users **U**. In the two experiments, **U** is used to restrict the function of oracles which would allow trivial success for the adversary:

- Privacy challenge users may not be removed from the group, i.e. UpdateGroup returns $\perp$ if $R \cap \mathbf{U} \neq \emptyset$. This is because Update is defined to return $\perp$ if the group information does not change as result of the revocation, which would be the case if the user was already removed from the group.

- Privacy challenge users may not be corrupted or have their keys revealed. Note, that this also prevents an adversary from re-enrolling a challenge user by initiating a join-issue session for them.

- The signing oracle treats signature requests for user IDs in the privacy challenge set differently. In the case of join privacy, a signature request for any privacy challenge user, i.e. uid_0 or uid_1 will be treated like a signature request for user uid_b who joined the system. In the case of leave privacy, it will be treated like a signature request for user $\mathsf{uid}_{(1-b)}$ who did not leave the group. Additionally, the queries will be added to the set of challenge queries $\mathbf{Q}^*$, which prevents the adversary from using the Trace oracle to produce an opening for them.

Definition 4.7: Join Privacy

For a fully dynamic group signature scheme DGS consider the join-privacy experiment between a challenger and a two-stage adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$.

$\mathsf{Join\text{-}Privacy}^{\mathcal{A}}_{\mathsf{DGS}}(1^\lambda)$

```
$(\mathsf{reg}, \mathsf{pp}) \leftarrow \mathsf{DGS.Setup}(1^\lambda)$
$(\mathsf{msk}, \mathsf{mpk}, \mathsf{info}, \mathsf{tsk}, \mathsf{tpk}) \leftarrow \langle \mathsf{KeyGen}_{\mathcal{G}}(\mathsf{pp}), \mathsf{KeyGen}_{\mathcal{T}}(\mathsf{pp}) \rangle$
$\mathsf{gpk} := (\mathsf{pp}, \mathsf{mpk}, \mathsf{tpk})$
$(\mathsf{state}, \mathsf{uid}_0, \mathsf{uid}_1) \leftarrow \mathcal{A}_0(\mathsf{gpk}, \mathsf{info})$
**if** $\{\mathsf{uid}_0, \mathsf{uid}_1\} \cap \mathbf{C} \neq \emptyset$ **then return** false
$b \leftarrow \{0,1\}$; $(\mathsf{info}^*, \mathsf{upk}[\mathsf{uid}_b]) \leftarrow \mathsf{AddUser}(\mathsf{uid}_b)$;
$(\mathsf{usk}[\mathsf{uid}_{1-b}], \mathsf{upk}[\mathsf{uid}_{1-b}]) \leftarrow \mathsf{UserKeyGen}(1^\lambda)$
$\eta^* := \eta_{now}$; $\mathbf{U} := \{\mathsf{uid}_0, \mathsf{uid}_1\}$
$d \leftarrow \mathcal{A}_1(\mathsf{state}, \mathsf{info}^*, \mathsf{upk}[\mathsf{uid}_0], \mathsf{upk}[\mathsf{uid}_1])$
**return** $b = d$
```

Both $\mathcal{A}_0$ and $\mathcal{A}_1$ have access to the oracles AddUser, Reveal, CorruptUser, SendM, Sign, Trace, UpdateGroup at any point during their runtime.

We define the adversary's advantage in the Join-Privacy experiment as

$$\mathsf{Adv}_{\mathcal{A},\mathsf{DGS}}^{\mathsf{Join\text{-}Privacy}}(\lambda) := \left|\Pr\left[\mathsf{Join\text{-}Privacy}_{\mathsf{DGS}}^{\mathcal{A}}(1^\lambda) \Rightarrow \mathsf{true}\right] - \frac{1}{2}\right|.$$

A group signature scheme DGS has *join privacy* if for all PPT adversaries $\mathcal{A}$, we have

$$\mathsf{Adv}_{\mathcal{A},\mathsf{DGS}}^{\mathsf{Join\text{-}Privacy}}(\lambda) \leq \mathsf{negl}(\lambda)\,.$$

Definition 4.8: Leave Privacy

For a fully dynamic group signature scheme DGS consider the join-privacy experiment between a challenger and a two-stage adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$.

$\mathsf{Leave\text{-}Privacy}_{\mathsf{DGS}}^{\mathcal{A}}(1^\lambda)$

```
$(\mathsf{reg}, \mathsf{pp}) \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{msk}, \mathsf{mpk}, \mathsf{info}, \mathsf{tsk}, \mathsf{tpk}) \leftarrow \langle \mathsf{KeyGen}_{\mathcal{G}}(\mathsf{pp}), \mathsf{KeyGen}_{\mathcal{T}}(\mathsf{pp})\rangle$
$\mathsf{gpk} := (\mathsf{pp}, \mathsf{mpk}, \mathsf{tpk})$
$(\mathsf{state}, \mathsf{uid}_0, \mathsf{uid}_1) \leftarrow \mathcal{A}_0(\mathsf{gpk}, \mathsf{info})$
$\textbf{if } \{\mathsf{uid}_0, \mathsf{uid}_1\} \cap \mathbf{H} \setminus (\mathbf{C} \cup \mathbf{B}) \neq \{\mathsf{uid}_0, \mathsf{uid}_1\} \textbf{ then return } \mathsf{false}$
$b \leftarrow \{0,1\}; \mathbf{U} := \{\mathsf{uid}_0, \mathsf{uid}_1\}; \mathsf{invert} := \textbf{true}; \eta^* := \eta_{now}$
$\mathsf{info}^* \leftarrow \mathsf{Update}(\mathsf{gpk}, \mathsf{msk}, \mathsf{info}_{\eta^*}, \mathsf{uid}_b, \mathsf{reg})$
$d \leftarrow \mathcal{A}_1(\mathsf{state}, \mathsf{info}^*)$
$\textbf{return } b = d$
```

Both $\mathcal{A}_0$ and $\mathcal{A}_1$ have access to the oracles AddUser, Reveal, Sign,Trace, UpdateGroup at any point during their runtime.

We define the adversary's advantage in the Leave-Privacy experiment as

$$\mathsf{Adv}_{\mathcal{A},\mathsf{DGS}}^{\mathsf{Leave\text{-}Privacy}}(\lambda) := \left|\Pr\left[\mathsf{Leave\text{-}Privacy}_{\mathsf{DGS}}^{\mathcal{A}}(1^\lambda) \Rightarrow \mathsf{true}\right] - \frac{1}{2}\right|.$$

A group signature scheme DGS has *leave privacy* if for all PPT adversaries $\mathcal{A}$, we have

$$\mathsf{Adv}_{\mathcal{A},\mathsf{DGS}}^{\mathsf{Leave\text{-}Privacy}}(\lambda) \leq \mathsf{negl}(\lambda)\,.$$

Definition 4.9: Membership Privacy

We say a fully dynamic group signature scheme has *membership privacy*, if it has both join- and leave privacy.

Remark

Note that leave privacy as stated above only seems to ensure privacy when a single user leaves the group, however, the Update algorithm allows simultaneous membership revocation for a whole set of users R. A simple hybrid argument should suffice to extend the join privacy property from one revocation to many revocations.

4.4 Generic Construction of Membership-Private Group Signatures

In this section we formalize the group signature proposed in the introduction. We present the full algorithms in construction 4.1.

The idea of our construction is as follows. The issuer uses signatures on equivalence classes to certify group members SFPK public keys. As already noted in chapter 3 this forms self-blindable certificates, i.e. each member can randomize the certificate and their public key which is computationally indistinguishable from the original public key used during the issuing procedure. To add and revoke members, each epoch the issuer generates a new SPS-EQ key pair and puts the verification key in the epoch information. To prevent malicious epoch information, the issuer signs the SPS-EQ verification key using a standard digital signature scheme. To protect the identities of members, the issuer does not directly publish the new certificates but uses a randomization, i.e. certificates for public keys that are in relation to keys of members. To allow the members to restore the right certificate, the issuer encrypts the random coins that can be used to restore the original certificate. The encryption is done under the member's encryption key. What is more, key-privacy ensures that the ciphertexts do not leak the identities.

Statement x_{Sign}:

$$\begin{aligned}
&\exists\,(\mathsf{fvk}, r) \text{ s. t.}\\
&\quad \mathsf{SFPK.MoveVk}(\mathsf{fvk}, r) = \mathsf{fvk}'\\
&\quad \wedge \quad \mathsf{SFPK.Canonical?}(\mathsf{fvk})\\
&\quad \vee \quad \mathsf{ctx} = \mathsf{PKE.Enc}(\mathsf{tpk}, \mathsf{fvk})
\end{aligned}$$

To allow tracing of users we use the canonical representative of SFPK, i.e. while signing, the group member encrypts this canonical representative under the tracing authority public key and uses proof system $\mathsf{NIP}_{\mathsf{Sign}}$ to prove in statement x_{Sign} that the randomized SFPK verification key is in relation to this encrypted key.

In the end, the group signature is composed of a randomized signature on equivalence classes from the issuer on the randomized SFPK verification key of the member, a ciphertext of the canonical representative, a proof that this ciphertext is sound and a SFPK signature on all those values and the message.

Statement x_{Trace}:

$$\begin{aligned}
&\exists\,(\mathsf{tsk}) \text{ s. t.}\\
&\quad (\mathsf{upk}[\mathsf{uid}]) \leftarrow \mathsf{PKE.Dec}(\mathsf{tsk}, \mathsf{ctx})\\
&\quad \wedge \quad \mathsf{tsk} \leftrightarrow \mathsf{tpk}
\end{aligned}$$

Finally, the proof system $\mathsf{NIP}_{\mathsf{Trace}}$ is used by the tracing authority to prove in statement x_{Trace} that the decrypted public key corresponds to public keys used during the issuing procedure.

All in all let

- $\mathsf{SPS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Move}, \mathsf{ValKey})$ be a structure-preserving signature scheme on equivalence classes.

- $\mathsf{SFPK} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{tdGenSign}, \mathsf{Verify}, \mathsf{MoveKeys}, \mathsf{Check})$ be a signature scheme with flexible public keys.

- $\mathsf{DS} = (\mathsf{Sign}, \mathsf{Verify})$ be a digital signature scheme.

- $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a public key encryption scheme.

- $\mathsf{NIP}_{\mathsf{Sign}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ be a non-interactive proof system for $\mathcal{R}_{\mathsf{Trace}}$.

- $\mathsf{NIP}_{\mathsf{Trace}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$be a non-interactive proof system for $\mathcal{R}_{\mathsf{Judge}}$.

Construction 4.1: Fully Dynamic Group Signature

```
DGS.Setup($1^\lambda$)
    $(\mathsf{crs}_{\mathsf{SFPK}}, \cdot) \leftarrow \mathsf{SFPK.Setup}(1^\lambda)$
    $\mathsf{BG} \leftarrow \mathsf{BGGen}(1^\lambda)$
    $\mathsf{crs}_J \leftarrow \mathsf{NIP}_{\mathsf{Trace}}.\mathsf{Setup}(1^\lambda)$
    $\mathsf{crs}_T \leftarrow \mathsf{NIP}_{\mathsf{Sign}}.\mathsf{Setup}(1^\lambda)$
    $\eta := 0;\ \mathsf{reg} := \emptyset$
    return $\mathsf{pp} := (\mathsf{BG}, \mathsf{crs}_{\mathsf{SFPK}}, \mathsf{crs}_J, \mathsf{crs}_T)$
```

```
DGS.KeyGen$_{\mathcal{G}}$(pp)
    $(\mathsf{sk}_{\mathsf{DS}}, \mathsf{vk}_{\mathsf{DS}}) \leftarrow \mathsf{DS.KeyGen}(1^\lambda)$
    $(\mathsf{sk}_{\mathsf{SPS}}, \mathsf{vk}_{\mathsf{SPS}}) \leftarrow \mathsf{SPS.KeyGen}(\mathsf{BG}, \ell)$
    $\sigma_{\mathsf{DS}} \leftarrow \mathsf{DS.Sign}(\mathsf{sk}_{\mathsf{DS}}, \mathsf{vk}_{\mathsf{SPS}})$
    $\mathsf{info} := (\mathsf{vk}_{\mathsf{SPS}}, \sigma_{\mathsf{DS}}, \emptyset)$
    return $(\mathsf{msk} := (\mathsf{sk}_{\mathsf{DS}}, \mathsf{sk}_{\mathsf{SPS}}),$
            $\mathsf{mpk} := \mathsf{vk}_{\mathsf{DS}}, \mathsf{info})$
```

```
DGS.UserKeyGen($1^\lambda$)
    $(\mathsf{fsk}, \mathsf{fvk}) \leftarrow \mathsf{SFPK.KeyGen}(1^\lambda)$
    $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$
    return $(\mathsf{usk}[\mathsf{uid}] := (\mathsf{fsk}, \mathsf{dk}),$
            $\mathsf{upk}[\mathsf{uid}] := (\mathsf{fvk}, \mathsf{ek}))$
```

```
DGS.KeyGen$_{\mathcal{T}}$(pp)
    $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$
    $\mathsf{tsk} := \mathsf{dk}$
    $\mathsf{tpk} := \mathsf{ek}$
    return $(\mathsf{tsk}, \mathsf{tpk})$
```

```
DGS.Trace(gpk,tsk,info,reg,$m$,$\sigma$)
    parse
      $\sigma = (\mathsf{fvk}, \sigma_{\mathsf{EQ}}, \Pi_{\mathsf{SFPK}}, \mathsf{ctx}\sigma)$
    $(\mathsf{fvk}) \leftarrow \mathsf{PKE.Dec}(\mathsf{tsk}, \mathsf{ctx})$
    abort if $\neg\exists \mathsf{uid}$ s. t. $\mathsf{reg}[\mathsf{uid}] = (\mathsf{fvk}, \cdot)$
    $\mathsf{w}; = \mathsf{tsk}$
    $\pi \leftarrow \mathsf{NIP}_{\mathsf{Trace}}.\mathsf{Prove}(\mathsf{crs}_J, x_{\mathsf{Trace}}, \mathsf{w})$
    return $(\mathsf{uid}, \pi)$
```

```
DGS.Judge(gpk,info,$\pi$,upk[uid],$m$,$\sigma$)
    if $\mathsf{DGS.Verify}(\mathsf{gpk}, \mathsf{info}, m, \sigma) = \mathsf{false}$
     then return false
    $\sigma = (\cdot, \cdot, \Pi_{\mathsf{SFPK}}, \mathsf{ctx}\cdot)$
    $\mathsf{upk}[\mathsf{uid}] = (\mathsf{fvk}, \cdot)$
    return $\mathsf{NIP}_{\mathsf{Trace}}.\mathsf{Verify}(\mathsf{crs}_J, x_{\mathsf{Trace}}, \pi)$
```

```
$\mathsf{DGS.Issue}(\mathsf{info}, \mathsf{msk}, \mathsf{uid}, \mathsf{upk}[\mathsf{uid}])$

**parse**   $\mathsf{msk} = (\mathsf{sk}_{\mathsf{DS}}, \mathsf{sk}_{\mathsf{SPS}})$
    $\mathsf{upk}[\mathsf{uid}] = (\mathsf{fvk}^{\mathsf{uid}}, \mathsf{ek}^{\mathsf{uid}})$
    $\mathsf{info}[\mathsf{uid}] = (\mathsf{vk}_{\mathsf{SPS}}, \sigma_{\mathsf{DS}}, \mathsf{Active})$
**if** $\mathsf{SFPK.Canonical?}(\mathsf{fvk}) = \mathsf{false}$ **then** **abort**
$\delta \leftarrow \Delta$
$\mathsf{fvk}' \leftarrow \mathsf{SFPK.MoveVk}(\mathsf{fvk}^{\mathsf{uid}}, \delta)$
$\mathsf{ctx}^{\mathsf{uid}} \leftarrow \mathsf{PKE.Enc}(\mathsf{ek}^{\mathsf{uid}}, \delta);$
$\sigma_{\mathsf{EQ}}^{\mathsf{uid}} \leftarrow \mathsf{SPS.Sign}(\mathsf{sk}_{\mathsf{SPS}}, \mathsf{fvk}')$
$\mathsf{Active}' := \mathsf{Active} \cup \{(\mathsf{ctx}^{\mathsf{uid}}, \sigma_{\mathsf{EQ}}^{\mathsf{uid}})\}$
$\mathsf{info}[\mathsf{uid}] := (\mathsf{vk}_{\mathsf{SPS}}, \sigma_{\mathsf{DS}}, \mathsf{Active}')$
$\mathsf{reg}[\mathsf{uid}] := \mathsf{upk}[\mathsf{uid}]$
**return**
```

```
$\mathsf{DGS.Update}(\mathsf{gpk}, \mathsf{msk}, \mathsf{info}, R, \mathsf{reg})$

**parse**   $\mathsf{msk} = (\mathsf{sk}_{\mathsf{DS}}, \mathsf{sk}_{\mathsf{SPS}})$
    $\mathsf{info} = (\mathsf{vk}_{\mathsf{SPS}}, \sigma_{\mathsf{DS}}, \mathsf{Active})$
$(\mathsf{sk}_{\mathsf{SPS}}', \mathsf{vk}_{\mathsf{SPS}}') \leftarrow \mathsf{SPS.KeyGen}(\mathsf{pp}_{\mathsf{SPS}})$
$\sigma_{\mathsf{DS}}' \leftarrow \mathsf{DS.Sign}(\mathsf{sk}_{\mathsf{DS}}, \mathsf{vk}_{\mathsf{SPS}}')$
$\mathsf{msk} := (\mathsf{sk}_{\mathsf{DS}}, \mathsf{sk}_{\mathsf{SPS}}')$
$A := \{\mathsf{uid} \mid \mathsf{uid} \text{ is active}\}$
**foreach** $\mathsf{uid} \in A \setminus R$
    **parse** $\mathsf{reg}[\mathsf{uid}] = (\mathsf{fvk}^{\mathsf{uid}}, \mathsf{ek}^{\mathsf{uid}})$
    $\delta \leftarrow \Delta$
    $\mathsf{ctx}^{\mathsf{uid}} \leftarrow \mathsf{Enc}(\mathsf{ek}^{\mathsf{uid}}, \delta)$
    $\mathsf{fvk}' \leftarrow \mathsf{SFPK.MoveVk}(\mathsf{fvk}^{\mathsf{uid}}, \delta)$
    $\sigma_{\mathsf{EQ}}^{\mathsf{uid}} \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{SPS}}', \mathsf{fvk}')$
    $\mathsf{Active}' := \mathsf{Active}' \cup (\mathsf{ctx}^{\mathsf{uid}}, \sigma_{\mathsf{EQ}}^{\mathsf{uid}})$
$\mathsf{info}_{\eta+} := (\mathsf{vk}_{\mathsf{SPS}}', \sigma_{\mathsf{DS}}', \mathsf{Active}')$
**return** $\mathsf{info}_{\eta+}$
```

```
$\mathsf{DGS.Sign}(\mathsf{gpk}, \vec{\mathsf{gsk}}[\mathsf{uid}], \mathsf{info}_\eta, m)$

**parse** $\mathsf{info}_{\eta_{now}} = (\mathsf{vk_{SPS}}, \sigma_{\mathsf{DS}}, \mathsf{Active})$
    $\vec{\mathsf{gsk}}[\mathsf{uid}] = (\mathsf{fsk}, \mathsf{dk})$
    $\mathsf{gpk}[\mathsf{uid}] = (\mathsf{fvk}, \mathsf{ek})$
**if** $\neg\exists\ (\mathsf{ctx}, \sigma_{\mathsf{EQ}}) \in \mathsf{Active}$ **s.t.**
            $\delta \leftarrow \mathsf{Dec}(\mathsf{dk}, \mathsf{ctx})$
     **and** $\mathsf{fvk}' \leftarrow \mathsf{SFPK.MoveVk}(\mathsf{fvk}, \delta)$
     **and** $\mathsf{SPS.Verify}(\mathsf{vk_{SPS}}, \mathsf{fvk}', \sigma_{\mathsf{EQ}}) = \mathsf{true}$
 **then abort**
$\gamma \leftarrow \Delta$
$(\mathsf{fsk}', \mathsf{fvk}') \leftarrow \mathsf{SFPK.MoveKeys}(\mathsf{fsk}, \mathsf{fvk}, \gamma')$
$\sigma'_{\mathsf{EQ}} \leftarrow \mathsf{SPS.Move}(\mathsf{vk_{SPS}}, \mathsf{fvk}, \sigma_{\mathsf{EQ}}, \gamma \cdot \delta^{-1})$
$\mathsf{ctx}' \leftarrow \mathsf{PKE.Enc}(\mathsf{tpk}, \mathsf{fvk})$
$\mathsf{w} := (\mathsf{fvk}, r)$
$\Pi_{\mathsf{SFPK}} \leftarrow \mathsf{NIP_{Sign}.Prove}(\mathsf{crs}_T, \mathsf{x_{Sign}}, \mathsf{w}))$
$\sigma \leftarrow \mathsf{SFPK.Sign}(\mathsf{fsk}', m||\eta_{now}||\mathsf{fvk}'||\sigma'_{\mathsf{EQ}}||\Pi_{\mathsf{SFPK}}||\mathsf{ctx})$
**return** $\sigma := (\mathsf{fvk}', \sigma'_{\mathsf{EQ}}, \Pi_{\mathsf{SFPK}}, \mathsf{ctx}, \sigma)$
```

```
$\mathsf{DGS.Verify}(\mathsf{gpk}, \mathsf{info}_\eta, m, \sigma)$

**parse** $\mathsf{info}_\eta = (\mathsf{vk_{SPS}}, \sigma_{\mathsf{DS}}, \cdot)$;
    $\mathsf{mpk} = \mathsf{vk}$
$\sigma = (\mathsf{fvk}, \sigma_{\mathsf{EQ}}, \Pi_{\mathsf{SFPK}}, \mathsf{ctx}, \sigma)$
**if** $\mathsf{DS.Verify}(\mathsf{vk}, \mathsf{vk_{SPS}}, \sigma_{\mathsf{DS}}) = \mathsf{false}$ **or**
    $\mathsf{NIP.Verify}(\mathsf{crs}_\Pi, x_{\mathsf{Sign}}, \Pi_{\mathsf{SFPK}}) = \mathsf{false}$ **or**
    $\mathsf{SPS.Verify}(\mathsf{vk_{SPS}}, \mathsf{fvk}, \sigma_{\mathsf{EQ}}) = \mathsf{false}$
 **then return** false
$M := m||\eta||\mathsf{fvk}||\sigma_{\mathsf{EQ}}||\Pi_{\mathsf{SFPK}}||\mathsf{ctx}$
**return** $\mathsf{SFPK.Verify}(\mathsf{fvk}, M, \sigma)$
```

4.4.1 Proof of Traceability

Theorem 4.1: Traceability

Our construction is traceable if the SPS-EQ scheme is existential unforgeable under chosen-message attacks, the SFPK scheme is existential unforgeable and the signature scheme used by the Issuer is existential unforgeable under chosen-message attacks.

Proof. Let us denote by S_i the event that the adversary wins the traceability experiment in $\mathcal{H}_i$. Let

$$\begin{aligned}&m^*,\\&\sigma^* = (\mathsf{fvk}^*, \sigma^*_{\mathsf{EQ}}, \Pi^*_{\mathsf{SFPK}}, \mathsf{ctx}^*_{\mathsf{SFPK}}, \sigma^*),\\&\mathsf{info}^*_{\eta} = ({\mathsf{vk}_{\mathsf{SPS}}}^*, \sigma^*, \mathsf{Active}^*)\end{aligned}$$

be the forgery outputted by the adversary. Moreover, let u be the maximum number of oracle queries to UpdateGroup made by the adversary and n the number of queries to the AddUser oracle.

$\mathcal{H}_0$: The original experiment.

$\mathcal{H}_1$: We abort in the case that $\mathsf{DS.Verify}(\mathsf{vk}, {\mathsf{vk}_{\mathsf{SPS}}}^*, \sigma^*) = \mathsf{true}$ but the signature σ^* was not created by the UpdateGroup oracle. Informally, we exclude the case that the adversary creates a custom SPS-EQ verification key and uses it to create its own epoch information.

It is easy to see that this change only decreases the adversary's advantage by a negligible fraction. In particular, we can simply use any adversary $\mathcal{A}$ to break the existential unforgeability of the digital signature scheme used by the Issuer. Thus, it follows that $|\Pr[S_1] - \Pr[S_0]| \leq \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A},\mathsf{DS}}(\lambda)$.

$\mathcal{H}_2$: We abort in case the proof for statement x_{Sign} is invalid, i.e. there exists no random coins r such that $\mathsf{SFPK.MoveVk}(\mathsf{fvk}, r) = \mathsf{fvk}'$.

It is easy to see that this would mean that we can use an adversary outputting such proof to break the soundness property of the proof system $\mathsf{NIP}_{\mathsf{Sign}}$. We have shown that $|\Pr[S_2] - \Pr[S_1]| \leq \mathsf{Adv}^{\mathsf{Soundness}}_{\mathsf{NIP}_{\mathsf{Sign}},\mathcal{A}}(\lambda)$.

$\mathcal{H}_3$: Choose $u^* \leftarrow \{1, \ldots, u\}$ and abort if info^*_η is not the output of the u^*-th call to the UpdateGroup oracle.

Because of the changes made by the previous game we know that the adversary can only use epoch information outputted by this oracle. Thus, we have $\Pr[S_2] = u \cdot \Pr[S_3]$.

$\mathcal{H}_4$: We abort in case $\mathsf{DGS.Trace}(\mathsf{gpk}, \mathsf{tsk}, \mathsf{info}^*_\eta, \mathsf{reg}, m^*, \sigma^*) = \perp$ but $\mathsf{DGS.Verify}(\mathsf{gpk}, \mathsf{info}^*_\eta, m^*, \sigma^*) = \mathsf{true}$. Informally, we exclude the case that the adversary creates a new user from outside the group, i.e. a new SPS-EQ signature.

We will show that any adversary $\mathcal{A}$ returns a forgery for which we abort, can be used to break the existential unforgeability of the SPS-EQ signature scheme. The reduction $\mathcal{R}$ algorithm on input of the verification key $\mathsf{vk}_{\mathsf{SPS}}$ performs as follows. It first sets $\mathsf{info}_{u^*} = (\mathsf{vk}_{\mathsf{SPS}}, \mathsf{DS.Sign}(\mathsf{sk}, \mathsf{vk}_{\mathsf{SPS}}), \mathsf{Active})$. For every active user i is this epoch, Active contains a tuple $(\mathsf{PKE.Enc}(\mathsf{ek}^i, k_i), \sigma^i_{\mathsf{EQ}})$, where σ^i_{EQ} is a signature generated for $\mathcal{R}$ by the signing oracle on input $\mathsf{SFPK.MoveVk}(\mathsf{fvk}^i, k_i)$. It then runs the system for $\mathcal{A}$ according to description.

After some interactions, the adversary returns the forgery. Note that because of the changes in the previous games, we know that $\mathsf{vk}_{\mathsf{SPS}}{}^* = \mathsf{vk}_{\mathsf{SPS}}$, i.e. the forgery is created for an epoch that uses our challenged SPS-EQ public key to certify members. Finally, the reduction $\mathcal{R}$ returns $(\mathsf{fvk}^*, \sigma^*_{\mathsf{EQ}})$ as a valid forgery. It is easy to see that this is a valid solution. Note that since opening failed, this means that the trusted authority decrypted a verification key fvk that is not a verification key of any honest user.
We conclude that $|\Pr[S_4] - \Pr[S_3]| \leq \mathsf{Adv}^{\ell,\mathsf{EQ\text{-}EUF\text{-}CMA}}_{\mathsf{SPS\text{-}EQ},\mathcal{A}}(\lambda)$.

Finally, we will show hat any adversary $\mathcal{A}$ that has non-negligible advantage in winning traceability experiment in $\mathcal{H}_3$ can be used by a reduction algorithm $\mathcal{R}$ to break the existential unforgeability of the SFPK scheme for a verification key fvk.

The reduction simulator works as follows. It generates all values according to description but for $i \leftarrow [n]$ the reduction answers the i-th queries of the adversary to AddUser by setting $\mathsf{upk}[] = (\mathsf{fvk}, \mathsf{ek})$ for some $(\mathsf{dk}, \mathsf{ek}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$. The reduction aborts if at some point the adversary asks for the group signing key of this member.

To answer signing queries $\mathsf{Sign}(i, m, \eta)$ for this member, the reduction parses $\mathsf{info}_\eta = (\mathsf{vk}_{\mathsf{SPS}}, \cdot, \mathsf{Active})$. Then it returns $\perp$ if for all tuples $(E, \sigma_{\mathsf{EQ}})$ in Active the decryption $k \leftarrow \mathsf{PKE.Dec}(\mathsf{dk}, E)$ fails. $\mathcal{R}$ chooses random coins $r \leftarrow r$, randomizes the flexible public key $\mathsf{fvk}' \leftarrow \mathsf{SFPK.MoveVk}(\mathsf{fvk}, r)$, the signature $\sigma'_{\mathsf{EQ}} \leftarrow \mathsf{SPS.Move}(\mathsf{vk}_{\mathsf{SPS}}, \mathsf{fvk}, \sigma_{\mathsf{EQ}}, r \cdot k^{-1})$ and computes ciphertext $\mathsf{ctx}_{\mathsf{SFPK}} \leftarrow \mathsf{Enc}(\mathsf{tpk}, \mathsf{fvk})$. It then creates a proof Π_{SFPK} for the statement:

$$\begin{aligned} x = \{\ \exists_{\mathsf{fvk},r}\ & \mathsf{SFPK.MoveVk}(\mathsf{fvk}, r) = \mathsf{fvk}' \\ \wedge\ & \mathsf{SFPK.Canonical?}(\mathsf{fvk}) = \mathsf{true} \\ \wedge\ & \mathsf{ctx}_{\mathsf{SFPK}} = \mathsf{Enc}(\mathsf{tpk}, \mathsf{fvk})\ \} \end{aligned}$$

using witness $w = (\mathsf{fvk}, r)$. It then uses its own signing oracle $\mathsf{OSign}^2((m||\eta|| \mathsf{fvk}'||\sigma'_{\mathsf{EQ}}||\Pi_{\mathsf{SFPK}}||\mathsf{ctx}_{\mathsf{SFPK}}), r)$, receiving signature σ. Finally, it outputs $\sigma = (\mathsf{fvk}', \sigma'_{\mathsf{EQ}}, \Pi_{\mathsf{SFPK}}, \mathsf{ctx}_{\mathsf{SFPK}}, \sigma)$. Note that since values required to perform the above computations are known to $\mathcal{R}$, it can efficiently compute valid group signatures for this member.

Finally, $\mathcal{A}$ outputs a valid group signature

$$\begin{aligned} & m^*, \\ & \sigma^* = (\mathsf{fvk}^*, \sigma^*_{\mathsf{EQ}}, \Pi^*_{\mathsf{SFPK}}, \mathsf{ctx}^*_{\mathsf{SFPK}}, \sigma^*), \\ & \mathsf{info}^*_\eta = (\mathsf{vk}_{\mathsf{SPS}}{}^*, \sigma^*, \mathsf{Active}^*) \end{aligned}$$

and the reduction algorithm outputs

$$((m^*||\eta^*||\mathsf{fvk}^*||\sigma^*_{\mathsf{EQ}}||\Pi^*_{\mathsf{SFPK}}), \sigma^*)$$

as a valid SFPK forgery. Note that this is only true if fvk^* and fvk are in the same equivalence class. By the changes made in the previous games we know that fvk^* is in a relation with a public key of an honest user and with probability $1/n$ we guessed the correct member for which $\mathsf{Trace}(m^*, \sigma^*, \mathsf{info}^*_\eta) = i$, and we have set their verification key to fvk. Note that also in such a case we do not have to worry about a corruption query for this member, since the forgery must be for non-corrupted users. We conclude that since m^* was never queried previously, the reduction also never used the prefix m^* in its oracle queries. In the end we have:

$$\begin{aligned} \Pr[S_0] \leq & u \cdot \left(n \cdot \mathsf{Adv}^{\mathsf{Flex\text{-}Unforgeability}}_{\mathcal{A},\mathsf{SFPK}}(\lambda) + \mathsf{Adv}^{\ell,\mathsf{EQ\text{-}EUF\text{-}CMA}}_{\mathsf{SPS\text{-}EQ},\mathcal{A}}(\lambda) \right) \\ & + \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A},\mathsf{DS}}(\lambda) + \mathsf{Adv}^{\mathsf{Soundness}}_{\mathsf{NIP}_{\mathsf{Sign}},\mathcal{A}}(\lambda). \end{aligned}$$

□

Corollary 4.2: Correctness

Since our construction fulfills traceability, and traceability implies correctness, our construction is also correct.

4.4.2 Proof of Anonymity

Theorem 4.3: Anonymity

Our construction is anonymous if the SPS-EQ signature scheme perfectly adapts signatures, the SFPK scheme is adaptively class hiding with key corruption and strongly existential unforgeable, the proof system used by signers is witness-indistinguishable and the proof system used by the tracing authority is zero-knowledge.

Proof. We will use the game base approach. Let us denote by S_i the event that the adversary wins the anonymity experiment in $\mathcal{H}_i$. Moreover, let n be the number of queries to the AddUser oracle made by the adversary and let $(\mathsf{info}^*_\eta, \mathsf{uid}^*_1, \mathsf{uid}^*_2, m^*)$ be the query made to the $\mathsf{Challenge}_b$ oracle, which outputs

$$\sigma^* = (\mathsf{fvk}^*, \sigma^*_{\mathsf{EQ}}, \Pi^*_{\mathsf{SFPK}}, \mathsf{ctx}^*_{\mathsf{SFPK}}, \sigma^*).$$

$\mathcal{H}_0$: The original experiment.

$\mathcal{H}_1$: We simulate the proof generated in Trace by the tracing authority.

Obviously, we only lower the advantage of the adversary by a negligible fraction because of the zero-knowledge property of this proof. Thus, we have $|\Pr[S_1] - \Pr[S_0]| \leq \mathsf{Adv}^{\mathsf{ZK}}_{\mathcal{A},\mathsf{NIP}_{\mathsf{Trace}}}(\lambda)$.

$\mathcal{H}_2$: We change the way the Trace oracle works. Instead of using tsk to decrypt fvk from $\mathsf{ctx}_{\mathsf{SFPK}}$, we first extract the witness (fvk, r) and use fvk instead. What is more, we simulate the proof Π^*_{SFPK}, which is part of the challenge signature.

Note that since the proof system $\mathsf{NIP}_{\mathsf{Sign}}$ is simulation-sound extractable it follows that $|\Pr[S_2] - \Pr[S_1]| \leq \mathsf{Adv}^{\mathsf{SSE}}_{\mathcal{A},\mathsf{NIP}_{\mathsf{Sign}}}(\lambda)$.

$\mathcal{H}_3$: We change the way the ciphertext $\mathsf{ctx}^*_{\mathsf{SFPK}}$ is computed. Instead of encrypting the canonical representative, we encrypt the value0.

Note that because of the changes made in the previous game, the Trace oracle works as in $\mathcal{H}_2$. Thus, we have that $|\Pr[S_3] - \Pr[S_2]| \leq \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKE},\mathcal{A}}(\lambda)$

$\mathcal{H}_4$: We now change the way we compute σ^*_{EQ}. Instead of using the Move algorithm to change representation of an old signature, we compute the SPS-EQ signature directly on fvk^*.

Since the SPS-EQ signature scheme perfectly adapts signatures, we have $\Pr[S_4] = \Pr[S_3]$

$\mathcal{H}_5$: Given the experiments bit b, we choose index $i \leftarrow [n]$ and abort if uid_b does not correspond to the user created in the i-th query of the adversary to AddUser.

We have $\Pr[S_4] = n \cdot \Pr[S_5]$.

$\mathcal{H}_6$: Let fvk be the SFPK verification key of the user chosen in the previous game. We now instead of using fvk to create fvk^*, we use a fresh key generated using SFPK.KeyGen.

We will now show that any adversary $\mathcal{A}$ that can distinguish those games, can be used to brake the weak class hiding of the SFPK scheme. We will show how to build a reduction $\mathcal{R}$ that does this. Let $(\mathsf{fsk}^0, \mathsf{fvk}^0)$, $(\mathsf{fsk}^1, \mathsf{fvk}^1)$ and fvk' be the inputs given to $\mathcal{R}$ by the challenger in the adaptive class hiding experiment. The reduction then sets fvk^0 as the i-th honest user SFPK public key. All other key material for those users is constructed as described in the scheme. Now in order to answer the query $(\mathsf{info}^*_\eta, \mathsf{uid}^*_1, \mathsf{uid}^*_2, m^*)$ to the $\mathsf{Challenge}_b$ oracle, the reduction: sets $\mathsf{fvk}^* = \mathsf{fvk}'$, computes σ^*_{EQ} as in $\mathcal{H}_3$, computes Π^*_{SFPK} as in $\mathcal{H}_2$, computes $\mathsf{ctx}^*_{\mathsf{SFPK}} \leftarrow \mathsf{PKE.Enc}(\mathsf{tpk}, \mathsf{fvk}')$, asks its signing oracle for σ^* under message $m^*||\eta^*||\mathsf{fvk}^*||\sigma^*_{\mathsf{EQ}}||\Pi^*_{\mathsf{SFPK}}||\mathsf{ctx}^*_{\mathsf{SFPK}}$, and returns $\sigma^* = (\mathsf{fvk}^*, \sigma^*_{\mathsf{EQ}}, \Pi^*_{\mathsf{SFPK}}, \mathsf{ctx}^*_{\mathsf{SFPK}}, \sigma^*)$. Note that since it knows fsk^0 and fsk^1 it can easily answer all corruption queries made by $\mathcal{A}$. In the end $\mathcal{A}$ outputs a bit b, which is also returned by $\mathcal{R}$. It follows that we have $|\Pr[S_6] - \Pr[S_5]| \leq \mathsf{Adv}^{\mathsf{Class\text{-}Hiding}}_{\mathcal{A},\mathsf{SFPK}}(\lambda)$.

We now argue that the only way the adversary $\mathcal{A}$ can break anonymity is by creating a randomization

$$\sigma' = (\mathsf{fvk}', \sigma'_{\mathsf{EQ}}, \Pi'_{\mathsf{SFPK}}, \mathsf{ctx}'_{\mathsf{SFPK}}, \sigma')$$

of the signature $\sigma^* = (\mathsf{fvk}^*, \sigma^*_{\mathsf{EQ}}, \Pi^*_{\mathsf{SFPK}}, \mathsf{ctx}^*_{\mathsf{SFPK}}, \sigma^*)$ and use σ' in a query to the Trace oracle. Since in $\mathcal{H}_5$ we changed the verification key fvk^* to a random one, this is the only part of the simulation, where the adversary can notice something. Thus, for this to work the adversary must use a valid signature σ' for $\mathsf{fvk}' \in [\mathsf{fvk}^*]_{\mathcal{R}}$. We distinguish two cases: $\sigma' = \sigma^*$ and $\sigma' \neq \sigma^*$. If $\sigma' = \sigma^*$

this means that $\mathsf{fvk}' = \mathsf{fvk}^*$ and either $\sigma'_{\mathsf{EQ}} \neq \sigma^*_{\mathsf{EQ}}$ or $\Pi'_{\mathsf{SFPK}} \neq \Pi^*_{\mathsf{SFPK}}$. Since fvk^* is set to random verification key in $\mathcal{H}_6$ we can use an adversary that creates such a signature σ' to break strong existential unforgeability of the SFPK scheme. In case $\sigma' \neq \sigma^*$, we notice that in order for the adversary to see that this is a simulation the verification key fvk' must be in relation to fvk^*. Thus, we can again use the adversary to break the strong existential unforgeability of the SFPK scheme, even if $\sigma'_{\mathsf{EQ}} = \sigma^*_{\mathsf{EQ}}$, $\Pi'_{\mathsf{SFPK}} = \Pi^*_{\mathsf{SFPK}}$ and $\mathsf{fvk}' = \mathsf{fvk}^*$.

In other words, the only way the adversary can randomize the challenged signature is by randomizing the SFPK signature because the other values are signed. However, since the scheme is strongly unforgeable the adversary has negligible chances to do so. It follows that $\Pr[S_6] = \mathsf{Adv}^{\mathsf{Flex\text{-}sUnforgeability}}_{\mathcal{A},\mathsf{SFPK}}(\lambda)$. In the end we have:

$$\begin{aligned}\Pr[S_0] \leq & n \cdot \left(\mathsf{Adv}^{\mathsf{Class\text{-}Hiding}}_{\mathcal{A},\mathsf{SFPK}}(\lambda) + \mathsf{Adv}^{\mathsf{Flex\text{-}sUnforgeability}}_{\mathcal{A},\mathsf{SFPK}}(\lambda)\right) \\ & + \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKE},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{SSE}}_{\mathcal{A},\mathsf{NIP_{Sign}}}(\lambda) + \mathsf{Adv}^{\mathsf{ZK}}_{\mathcal{A},\mathsf{NIP_{Trace}}}(\lambda).\end{aligned}$$

□

4.4.3 Proof of Non-frameability

Theorem 4.4: Non-frameability

Our construction is non-frameable if the SFPK scheme is existential unforgeable and the proof system used by the tracing authority is sound.

Proof. We again use the game base approach. Let us denote by S_i the event that the adversary wins the anonymity experiment in $\mathcal{H}_i$. Moreover, let n be the number of queries to the CorruptUser oracle made by the adversary and let $(m^*, \sigma^*, \mathsf{uid}^*, \pi^*_{\mathrm{Trace}}, \mathsf{info}^*_\eta)$ be the output of the adversary $\mathcal{A}$.

$\mathcal{H}_0$: The original experiment.

$\mathcal{H}_1$: Let $\sigma^* = (\mathsf{fvk}^*, \sigma^*_{\mathsf{EQ}}, \Pi^*_{\mathsf{SFPK}}, \mathsf{ctx}^*_{\mathsf{SFPK}}, \sigma^*)$. We decrypt fvk' from $\mathsf{ctx}^*_{\mathsf{SFPK}}$ using the tracing authorities secret key tsk. We abort if $\mathsf{fvk}' \neq \mathsf{upk}[\mathsf{uid}^*]$ but the $\mathsf{DGS.Judge}(\mathsf{gpk}, \mathsf{uid}^*, \mathsf{info}^*_\eta, \pi^*_{\mathrm{Trace}}, \mathsf{upk}[\mathsf{uid}^*], m^*, \sigma^*)$ outputs true.

We will show that this lowers the adversaries advantage only by a negligible fraction. In particular, this means that π^*_{Trace} is a valid proof for the statement:

$$\exists_{\mathsf{tsk}} \ \ (\mathsf{upk}[\mathsf{uid}^*]) \leftarrow \mathsf{PKE.Dec}(\mathsf{tsk}, \mathsf{ctx}_{\mathsf{SFPK}}) \ \ \wedge \ \ \mathsf{tsk} \leftrightarrow \mathsf{tpk}$$

However, since we know that $\mathsf{fvk}' \neq \mathsf{upk}[\mathsf{uid}^*]$ it follows that π^*_{Trace} is a proof that breaks the soundness property of the proof used by the tracing authority. We have shown that $|\Pr[S_1] - \Pr[S_0]| \leq \mathsf{Adv}^{\mathsf{Soundness}}_{\mathcal{A},\mathsf{NIP}_{\mathsf{Trace}}}(\lambda)$.

$\mathcal{H}_2$: We now choose a random $j \in [n]$ and abort in case $j \neq \mathsf{uid}^*$.
It is easy to see that $\Pr[S_1] = n \cdot \Pr[S_2]$.

We will now show that any adversary $\mathcal{A}$ that breaks the non-frameability of the scheme can be used to break the existential unforgeability of the SFPK scheme. To do so, we construct a reduction $\mathcal{R}$ that plays the role of the adversary in the existential unforgeability experiment. Let fvk be the verification key given to $\mathcal{R}$. The reduction sets $\mathsf{upk}[j] = \mathsf{fvk}$, where j is the identifier from $\mathcal{H}_2$. To answer the queries to the Sign oracle for $\mathsf{uid} = j$, the reduction outputs group signature $\sigma' = (\mathsf{fvk}', \sigma'_{\mathsf{EQ}}, \Pi'_{\mathsf{SFPK}}, \mathsf{ctx}'_{\mathsf{SFPK}}, \sigma')$. To do so, the reduction can choose the randomization r freely and randomize the verification key fvk by running $\mathsf{fvk}' \leftarrow \mathsf{SFPK.MoveVk}(\mathsf{fvk}, r)$. It can also randomize the SPS-EQ signature to receive σ'_{EQ} and compute the proof Π'_{SFPK}. Finally, it uses its own signing oracle OSign^2 to compute the SFPK signature σ'.

In the end, the adversary returns a group signature $\sigma^* = (\mathsf{fvk}^*, \sigma^*_{\mathsf{EQ}}, \Pi^*_{\mathsf{SFPK}}, \mathsf{ctx}^*_{\mathsf{SFPK}}, \sigma^*)$ under message m^* and for epoch info^*_{η}, for which we know (by $\mathcal{H}_2$) that fvk^* is from the same relation as the verification key fvk from the existential unforgeability experiment. Since this is a valid forgery, it follows that $(\mathsf{uid}^*, m^*, \sigma^*, \eta^*) \notin \mathbf{Q}$ and that

$$((m^*||\eta^*||\mathsf{fvk}^*||\sigma^*_{\mathsf{EQ}}||\Pi^*_{\mathsf{SFPK}}||\mathsf{ctx}^*_{\mathsf{SFPK}}), \sigma^*)$$

is a valid forgery against the SFPK scheme.

We conclude that $\Pr[S_0] = n \cdot \mathsf{Adv}^{\mathsf{Flex\text{-}Unforgeability}}_{\mathcal{A},\mathsf{SFPK}}(\lambda) + \mathsf{Adv}^{\mathsf{Soundness}}_{\mathcal{A},\mathsf{NIP}_{\mathsf{Trace}}}(\lambda)$.

□

4.4.4 Proof of Functional Tracing Soundness

Theorem 4.5: Functional Tracing Soundness

Our construction has functional tracing soundness if the underlying SFPK scheme has canonical representatives, the proof system used by the Judge is sound and the proof system used by the signers is a proof of knowledge.

Proof. Let $\mathcal{A}$ be an adversary against the tracing soundness of our scheme. We show how to construct a reduction $\mathcal{B}$ against the soundness of $\mathsf{NIP}_{\mathsf{Trace}}$.

Given the CRS $\mathsf{crs}_{\mathsf{Judge}}$, the reduction generates the remaining parameters according to Setup and forwards them to the adversary. At some point the adversary will output the group and tracing manager's key material and the initial group information $(\mathsf{info}, \mathsf{msk}, \mathsf{mpk}, \mathsf{tsk}, \mathsf{tpk})$. Let us further denote by $(m, \sigma, \{\mathsf{uid}_i, \pi_i\}_{i=1}^{2}, \mathsf{info}_\eta)$ the adversary's final output. We will assume that $\mathsf{upk}[\mathsf{uid}_1]$ and $\mathsf{upk}[\mathsf{uid}_2]$ are both defined and not equal. Assume additionally that $\mathsf{Judge}(\mathsf{gpk}, \mathsf{uid}_i, \mathsf{info}_\eta, \pi_i, \mathsf{upk}[\mathsf{uid}_i], m, \sigma) = 1$ for both $i = 1$ and $i = 2$, i.e. we have in particular $\mathsf{NIP}_{\mathsf{Trace}}.\mathsf{Verify}(\mathsf{crs}_{\mathsf{Judge}}, x_{\mathsf{Trace}}, \pi_i) = \mathsf{true}$ for both i. We now consider the following cases:

Case I: The tracing authority's secret key is not properly generated, i.e. we have $(\tau', \mathsf{crs}') \leftarrow \mathsf{NIP}_{\mathsf{Trace}}.\mathsf{ExtSetup}(1^\lambda; \omega)$ for $(\tau', \mathsf{crs}') \neq (\tau_{\mathsf{PPE}}, \mathsf{crs}_{\mathsf{PPE}})$. The reduction can check this, since the adversary provides the ω as part of the tracing authority's secret key. In this case, either of the two proofs π_i breaks the soundness of $\mathsf{NIP}_{\mathsf{Trace}}$.

Case II: The tracing authority's secret key is properly generated. In this case, the reduction uses the extraction trapdoor to obtain the witness used for the proof Π_{SFPK} contained in the signature. There are two possibilities:

1. The extraction does not produce a valid witness. We bound this case by the advantage of $\mathcal{A}$ against the extractor.
2. The extraction is successful, yielding a valid witness $(\mathsf{fvk}, r, w_1, w_2)$. Since the witness is valid, fvk is the unique canonical representative of the key that created the signature. Since the keys $\mathsf{upk}[\mathsf{uid}_1]$ and $\mathsf{upk}[\mathsf{uid}_2]$ are different, at most one of them can be equal to the extracted fvk. The reduction thus returns $(x_{\mathsf{Trace}}, \pi_i)$ such that $\mathsf{upk}[\mathsf{uid}_i] \neq \mathsf{fvk}$ again breaking the soundness of $\mathsf{NIP}_{\mathsf{Trace}}$.

□

4.4.5 Proof of Membership Privacy

Theorem 4.6: Membership Privacy

Our construction has membership privacy if the encryption scheme used by the signers is IND-CPA secure and has IND-PK key privacy and the SFPK scheme is adaptively class hiding with key corruption.

Proof. We have to show that our construction achieves both join and leave privacy. We begin with the proof of join privacy.

Theorem 4.7: Join Privacy

Our construction has private joins if the encryption scheme used by the signers is IND-CPA secure and has IND-PK key privacy and the SFPK scheme is adaptively class hiding with key corruption.

Proof. We consider a series of games. In the following let uid_b be the challenge user who is inserted into the group and let $\mathsf{gpk}[\mathsf{uid}_b] = (\mathsf{fvk}, \mathsf{ek})$ be their public key and $\vec{\mathsf{gsk}}[\mathsf{uid}_b] = (\mathsf{fsk}, \mathsf{dk})$ be their signing key. Let S_i denote the event that the adversary wins in $\mathcal{H}_i$.

$\mathcal{H}_0$ Is the original join privacy game, so $\Pr[S_0] = \mathsf{Adv}^{\mathsf{Join\text{-}Privacy}}_{\mathsf{GS},\mathcal{A}}(\lambda)$.

$\mathcal{H}_1$ We modify how the challenge group information is created. For this we generate a fresh public key encryption key pair $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$. After the challenge user uid_b is added using $\mathsf{AddUser}$, we replace their entry $(\mathsf{ctx} = \mathsf{PKE.Enc}(\mathsf{ek}_b, k), \sigma_{\mathsf{EQ}})$ in the epoch information with $(\mathsf{PKE.Enc}(\mathsf{ek}, k), \sigma_{\mathsf{EQ}})$, i.e. we replace the encryption key of the randomness to a fresh key. It is easy to see that, since the encryption scheme has key privacy we have $\Pr[S_1] \leq \Pr[S_0] + \mathsf{Adv}^{\mathsf{IND\text{-}PK}}_{\mathsf{PKE},\mathcal{A}}(\lambda)$.

$\mathcal{H}_2$ In this game we further modify the ciphertext in the challenge user's part of info^* by encrypting the value 0 instead of the randomness used to change the SFPK key signed in σ_{EQ}. Because the encryption scheme is IND-CPA secure it holds that $\Pr[S_2] \leq \Pr[S_1] + \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKE},\mathcal{A}}(\lambda)$.

$\mathcal{H}_3$ Instead of changing the representative of user uid_b's SFPK public key, we generate a fresh verification key and change its representative. The signature in info^* will now be on this fresh representative. We will also use this fresh key to sign in the queries made to Privacy. We observe that $\Pr[S_3] \leq \Pr[S_2] + \mathsf{Adv}^{\mathsf{Class\text{-}Hiding}}_{\mathsf{SFPK},\mathcal{A}}(\lambda)$. Further, we have $\Pr[S_3] = \frac{1}{2}$, since the updated epoch information and the signatures received from the challenge signing oracle are completely independent of the challenge users.

Putting it all together we thus have

$$\mathsf{Adv}^{\mathsf{Join\text{-}Privacy}}_{\mathsf{GS},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{IND\text{-}PK}}_{\mathsf{PKE},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{Class\text{-}Hiding}}_{\mathsf{SFPK},\mathcal{A}}(\lambda).$$

□

Now we prove leave privacy of our construction.

Theorem 4.8: Leave Privacy

Our construction has leave privacy if the encryption scheme used by the signers is IND-CPA secure and has IND-PK key privacy and the SFPK scheme is adaptively class hiding with key corruption.

Proof. This proof follows similar steps as the proof for join privacy. We consider a series of games, where in the first game b is fixed to 0 and in the last game, b is fixed to 1. Let S_i denote the event that $\mathcal{A}$'s final output in $\mathcal{H}_i$ is 0.

$\mathcal{H}_0$ The Leave-Privacy game, where bit b is fixed to 0.

$\mathcal{H}_1$ We change the public key used to encrypt the epoch data for user uid_0 using the public key of user uid_1. We have $|\Pr[S_0] - \Pr[S_1]| \leq \mathsf{Adv}^{\mathsf{IND\text{-}PK}}_{\mathcal{A},\mathsf{PKE}}(\lambda)$.

$\mathcal{H}_2$ We now change the randomness encrypted in this ciphertext to the randomness for user uid_1. Because of IND-CPA security of the encryption scheme we have $|\Pr[S_1] - \Pr[S_2]| \leq \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{PKE}}(\lambda)$.

$\mathcal{H}_3$ We change the SFPK verification key to the verification key of uid_1, also changing the signatures in Privacy to this signing key. The game is now the same as the Leave-Privacy game with the bit fixed to 1. Because of adaptive class hiding we have $|\Pr[S_2] - \Pr[S_3]| \leq \mathsf{Adv}^{\mathsf{Class\text{-}Hiding}}_{\mathcal{A},\mathsf{SFPK}}(\lambda)$.

□

This concludes the proof of membership privacy. □

4.5 Efficient Instantiation

The generic construction presented above can be easily instantiated in the standard model, without random oracles, using known schemes. In particular, we can use the standard model signatures on equivalence classes by Fuchsbauer and Gay [FHS15] and a compatible SFPK signature schemes from chapter 3. For the encryption scheme one can use ElGamal encryption and standard model digital signatures. Finally, both proof systems can be instantiated using the simulation-sound system by Groth [Gro06].

However, due to the simulation-sound proof system and the large public keys of the SFPK schemes, the signature size is not competitive with existing schemes. We will now show how to minimize the signature size, while still using only building blocks that are secure under standard assumptions and without random oracles. The objective is to instantiate our construction in a way that it has shorter signatures than the current state-of-the-art scheme by Libert-Peters-Yung [LPY15] presented at Crypto'15, which is only secure in a weaker model.

Optimization. To decrease the signature size we have to solve the following problems:

1. The proof system $\mathsf{NIP}_{\mathsf{Sign}}$ must allow the security reduction for the anonymity experiment to simulate the challenged proof and at the same time extract witnesses to properly simulate the Trace oracle,
2. The verification key of the SFPK signature must be short and allow for a simple proof of canonical representation,
3. If possible, simplification of the statement proven in $\mathsf{NIP}_{\mathsf{Sign}}$.

First, we replace the simulation-sound system with a simple NIWI proof system. In fact, we instantiate all building blocks such that we can use the popular Groth-Sahai proofs for pairing product equations. To do so, we introduce a trapdoor witness that can be used by the reduction to simulate the proof, while still being able to extract the witness. Of course, we have to prevent the adversary from using this trapdoor to create valid proofs. We achieve this by introducing a new element $K_2 = g_2^k$ as part of the groups public key that will be part of the statement. The trapdoor witness are then two values w_1 and w_2, such that $e(w_1, K_2) = e(w_2, g_2)$. It is easy to see that any adversary that is able to compute such a witness can be used to break the DDH assumption in $\mathbb{G}_2$.

To solve the second problem we use construction 3.4 as our SFPK instantiation. The scheme uses public keys in $\mathbb{G}_1 \times \mathbb{G}_1$ with the established projective equivalence relation, i.e. $(\mathsf{fvk}, \mathsf{fvk}') \in \mathcal{R}$ if there is a $\mu \in \mathbb{Z}_p^*$ such that $\mathsf{fvk}_1^\mu = \mathsf{fvk}_1'$ and $\mathsf{fvk}_2^\mu = \mathsf{fvk}_2'$. For such classes of public keys, we define the canonical representative as the verification key for which the first element is just g_1.

To simplify the statement proven in $\mathsf{NIP}_{\mathsf{Sign}}$, we get rid of the ciphertext c_{SFPK}, that is used by the tracing authority to identify signers. To preserve this functionality, we allow the tracing authority to generate the parameters for the proof system $\mathsf{NIP}_{\mathsf{Sign}}$, including an extraction trapdoor which allows to extract the used witness and compute the corresponding canonical representative.

When applying all the above techniques the statement proven by the signer will have the form:

$$\begin{aligned}&\exists\,(\mathsf{fvk}, r, w_1, w_2)\text{ s. t.}\\&\mathsf{SFPK.MoveVk}(\mathsf{fvk}, r) = \mathsf{fvk}' \;\;\wedge\;\; \mathsf{SFPK.Canonical?}(\mathsf{fvk})\\&\;\vee\;\; e(w_1, K_2) = e(w_2, g_2).\end{aligned}$$

Efficiency of the Instantiation. The signature itself is composed of an SFPK verification key fvk', an SFPK signature σ, an SPS-EQ signature σ'_{EQ} and proof Π_{SFPK}. To instantiate SFPK signatures we use Scheme construction 3.4, which means that fvk' is 2 elements in $\mathbb{G}_1$ and σ is 2 elements in $\mathbb{G}_1$, 1 in $\mathbb{G}_2$ and 1 in $\mathbb{Z}_p^*$. This means that the SPS-EQ signature takes 10 elements in $\mathbb{G}_1$ and 4 elements in $\mathbb{G}_2$.

Taking into account that we will use construction 3.4, the above statement can be instantiated as follows. Let $\mathsf{fvk}' = (\mathsf{fvk}'_1, \mathsf{fvk}'_2)$ and $\mathsf{fvk} = (\mathsf{fvk}_1, \mathsf{fvk}_2)$, we can then express this proof by the pairing product equations: $e(w_1, K_2) = e(w_2, g_2)$ and $e(\mathsf{fvk}'_1, g_2^{r^{-1}}) = e(g_1, g_2) \cdot e(w_1, g_2)$. It is easy to see that the witness $(r, w_1, w_2) = (0, (g_1)^{-1}, (K_1)^{-1})$ is a trapdoor witness that can be used in the security proof to create a valid proof for an arbitrary fvk'. The canonical representative fvk is only used by the tracing authority to open signatures. However, by extracting the witness $R = g_2^{r^{-1}}$ it can still do this because if $\mathsf{fvk}'_2 = g_1^{x \cdot r}$, then $e(\mathsf{fvk}'_2, R) = e(g_1^x, g_2)$ is a static value that is common for all public keys in relation with fvk'. Since the tracing authority has access to the registration table that contains public keys in canonical form of active members it can correctly open signatures.

Instantiating those equations using the fine-tuned Groth-Sahai proofs presented in [EG14] (assuming decisional Diffie-Hellman), the proof size is 10 elements in $\mathbb{G}_1$ and 8 elements in $\mathbb{G}_2$. This is constituted by: 2 group elements in $\mathbb{G}_2$ for the first equation, which is linear; 4 elements in $\mathbb{G}_1$ and $\mathbb{G}_2$ for the second equation; 6 elements in $\mathbb{G}_1$ for the three witnesses in $\mathbb{G}_1$; 2 elements in $\mathbb{G}_2$ for the witness r. Overall the group signature is composed of 28 elements in $\mathbb{G}_1$, 15 in $\mathbb{G}_2$ and 1 in $\mathbb{Z}_p^*$.

The digital signature scheme DS and the public key encryption scheme PKE are standard components, an example of a key private PKE scheme is ElGamal encryption. The proof system $\mathsf{NIP}_{\mathsf{Trace}}$ can also be instantiated using Groth-Sahai proofs for pairing product equations [EG14]. Note that this means that the tracing authority has to prove correct decryption of a ciphertext (witnesses are encoded in form of ElGamal encryptions) and that its public key was generated using a DDH tuple, which can easily be expressed as pairing product equations.

4.6 Related Work

A related property to our membership privacy was proposed for the partially dynamic setting by Kiayias and Zhou in Hidden Identity-Based Signatures [KZ07] and efficiently instantiated by Chow et al. [CZZ17]. In these works, group membership lists are avoided altogether, enabling to hide the identity of group members even from the opening authority. We stress that in the fully dynamic model some form of group membership list is necessary to implement membership revocation, separating these approaches from ours.

The generic constructions of group signatures from [BMW03] and [BSZ05] established a design paradigm, which is sometimes called the *sign-and-encrypt-and-prove* paradigm (SEP). It is used in a number of constructions and may be informally described as follows: a signature consists of an encryption under the opener's public key of both a signature of the message under the member's signing key and the member's identity, as well as a non-interactive zero-knowledge proof that the identity contained in the encryption is valid and is indeed that of the signer of the message. The identity of the group member is typically a signature issued by the group manager. Thus, relying on the unforgeability of this signature, such a group signature scheme achieves non-frameability and traceability. Beside this design paradigm and generic construction, which are also based on this paradigm, Abdalla and Warinschi proved in [AW04] that group signatures are actually equivalent to IND-CPA secure encryption schemes.

In [Bic+10], Bichsel et al. identify the SEP design paradigm as a source of inefficiency in group signatures. Then they propose a new approach based on re-randomizable signature schemes and provide an efficient construction without encryption secure in the random oracle model. In this work we follow that idea, however we do not rely on the random oracle model to prove security of our scheme. By now many group signature schemes were designed for both the static and dynamic case in the random oracle model which utilize the RSA crypto-system [Ate+00; TX03; CG05; KY05a], discrete logarithm setting [AM03; FY05], and bilinear setting [BBS04; CL04].

One of the first standard model constructions was introduced by Ateniese et al. [Ate+05a]. The scheme is highly efficient, it utilizes bilinear maps and the signature consists only of 8 group elements. However, the scheme does not provide full-anonymity in sense of the definition in the BMW model [BMW03]. In particular, the adversary is not allowed to see the private keys of honest users.

Boyen and Waters [BW06; BW07] proposed standard model schemes that use composite order bilinear groups, but in contrast to [Ate+05a] allows key exposure attacks. However, the adversary cannot see any openings of signatures. This restricted version of full-anonymity is also called CPA-*anonymity*.

The introduction of the Groth-Sahai (GS) proof system [GS08] allowed for the design of new and efficient group signature schemes in the standard model. Groth [Gro07] was the first to introduce a standard model group signature with a constant size verification key and signatures, which preserve the full-anonymity property. The security of the scheme relies on a q-type assumption. The GS proof system was also used by Libert et al. [LPY12b; LPY12a], who designed standard model group signatures with revocation capabilities.

At Crypto'15 Libert, Peters and Yung [LPY15] introduced two efficient group signature schemes that rely on simple assumptions. The first scheme is secure in the static BMW model [BMW03]. On the other hand, the second construction is less efficient, but secure in the dynamic security model from [KY06].

Bootle et al. [Boo+15] propose a generic construction of group signatures from accountable ring signatures. They instantiate it using a scheme based on a sigma protocol in the random oracle model. Later, Bootle et al. [Boo+16] show that this construction is a fully dynamic group signature scheme. The idea is to include the description of the ring as part of the epoch information. This way only users in the ring are member of the group in the current epoch. Security follows directly from the security of accountable ring signatures.

Derler and Slamanig proposed a generic construction for dynamic group signatures based on structure preserving signatures on equivalence classes (SPS-EQ) [DS16]. SPS-EQ define a relation $\mathcal{R}$ that induces a partition on the message space. By signing one representative of a partition, the signer in fact signs the whole partition. Then, without knowledge of the signing key we can transform the signature to a different representative of the partition. Their group signatures make use of signatures of knowledge (as part of the group signature) and non-interactive zero-knowledge proof systems (in the issuing procedure and to ensure opening soundness). The authors present an efficient instantiation in the random oracle model. The main disadvantage of their construction is that there currently exists no standard model instantiation.

Group signatures can also be constructed from lattice-based assumptions [Lin+18] or symmetric primitives [BEF18]. The former is the only scheme secure under lattice-based assumptions for which the signature size does not depend on the number of group members. Unfortunately, it is only secure in the partially dynamic model [BSZ05] and in the random oracle model. The latter scheme is also instantiated in the random oracle model.

5 Logarithmic Size (Linkable) Ring Signatures

In this chapter:

5.1 Introduction

5.1.1 Technical Overview of Logarithmic Ring Signatures

To describe our scheme, it is instructive to recall the standard model ring signature scheme of Bender, Katz, and Morselli [BKM06]. In the BKM scheme, a ring verification key $\mathsf{rvk} = (\mathsf{vk}, \mathsf{ek})$ consists of a verification key vk for a standard signature scheme and an encryption key ek for a public key encryption scheme. The ring signing key

Publication History

The contributions of this chapter are based on and were first published in

> Michael Backes, Nico Döttling, Lucjan Hanzlik, Kamil Kluczniak, and Jonas Schneider. "Ring Signatures: Logarithmic-Size, No Setup - from Standard Assumptions." In: *Advances in Cryptology – EUROCRYPT 2019, Part III.* ed. by Yuval Ishai and Vincent Rijmen. Vol. 11478. Lecture Notes in Computer Science. Darmstadt, Germany: Springer, Heidelberg, Germany, May 2019, pp. 281–311. DOI: 10.1007/978-3-030-17659-4_10.

is just the corresponding signing key of the digital signature scheme sk. To sign a message m given a ring signing key rsk and a ring $\mathsf{R} = (\mathsf{rvk}_1, \ldots, \mathsf{rvk}_\ell)$, one proceeds as follows. In a first step, locate verification key $\mathsf{rvk}_{i^*} = (\mathsf{vk}_{i^*}, \mathsf{ek}_{i^*})$ corresponding to the signing key sk in the ring R. Now compute a signature σ of m using the signing key sk and encrypt σ under ek_{i^*} to obtain a ciphertext ctx_{i^*}. Next, for all $i \neq i^*$ compute *filler ciphertexts* ctx_i as encryptions of 0^λ under ek_i, where $\mathsf{rvk}_i = (\mathsf{vk}_i, \mathsf{ek}_i)$. Finally, use a non-interactive[1] witness-indistinguishable proof π for the statement $(m, \mathsf{ctx}_1, \ldots, \mathsf{ctx}_\ell, \mathsf{rvk}_1, \ldots, \mathsf{rvk}_\ell)$ to show that there exists an index i^* such that ctx_{i^*} encrypts a signature σ and that σ verifies for the message m under the verification key vk_{i^*}. The ring signature is now given by $\varsigma = (\mathsf{ctx}_1, \ldots, \mathsf{ctx}_\ell, \pi)$. To verify a signature ς for a message m and ring R, use the NIWI verifier to verify that π is a proof for the statement $(m, \mathsf{ctx}_1, \ldots, \mathsf{ctx}_\ell, \mathsf{rvk}_1, \ldots, \mathsf{rvk}_\ell)$.

We also briefly review how unforgeability and anonymity of this scheme are established. To establish unforgeability, note that by the perfect soundness of the NIWI proof π one of the ctx_i must actually be an encryption of a signature on m under vk_i. The security reduction can therefore set up all the ek_i such that it knows the corresponding secret keys and can decrypt the signature. Establishing anonymity relies on witness indistinguishability of the NIWI proof system. That is, the reduction can set up the signature ς such that, in fact, two different ciphertexts ctx_{i_0} and ctx_{i_1} encrypt a valid signature (each under their corresponding verification key). We can now use witness indistinguishability to switch the witness from index i_0 to i_1. Thus, we can establish that signatures computed using sk_{i_0} are computationally indistinguishable from signatures computed using sk_{i_1}. The size of the signature is linear in the ring size ℓ. There are two major obstacles in making the size of the signatures sub-linear:

1. The signature contains all the ciphertexts $\mathsf{ctx}_1, \ldots, \mathsf{ctx}_\ell$.

[1]Bender et al. [BKM06] actually use 2-message public-coin witness-indistinguishable proofs (ZAPs) rather than NIWI proofs, which is a slightly weaker primitive than NIWI proofs.

2. The statement over $(m, \mathsf{ctx}_1, \ldots, \mathsf{ctx}_\ell, \mathsf{rvk}_1, \ldots, \mathsf{rvk}_\ell)$ is also of size linear in ℓ since it includes a disjunction over all verification keys in the ring.

Reducing the number of ciphertexts. Starting from the BKM scheme, our first idea is that if we use an appropriate public key encryption scheme PKE, then we do not need to include all the ciphertexts $\mathsf{ctx}_1, \ldots, \mathsf{ctx}_\ell$ in the signature, but only two ciphertexts ctx and ctx′. The additional property we need from PKE is that a ciphertext ctx cannot be linked to the public key ek that was used to compute ctx, *unless* one is in the possession of the corresponding decryption key dk. This property immediately holds if the public key encryption scheme PKE has pseudorandom ciphertexts. In fact, many constructions of public key encryption have pseudorandom ciphertexts, e.g. the classic ElGamal scheme based on the DDH problem [ElG84] or Regev's scheme based on the LWE problem [Reg05].

Our first modification is thus to compute ctx by encrypting the signature σ under ek_{i^*} and choosing ctx′ uniformly at random. We also compute the proof π differently. Namely, we prove that for a statement of the form $(m, \mathsf{ctx}, \mathsf{ctx}', \mathsf{rvk}_1, \ldots, \mathsf{rvk}_\ell)$ it holds that there exist indices i^* and $i^\dagger$ such that either ctx is an encryption of a signature σ^* of m with respect to the verification key vk_{i^*} under the public key ek_{i^*}, *or* ctx′ is an encryption of a signature $\sigma^\dagger$ of m with respect to the verification key $\mathsf{vk}_{i^\dagger}$ under the public key $\mathsf{ek}_{i^\dagger}$. In this modified scheme, a signature $\varsigma = (\mathsf{ctx}, \mathsf{ctx}', \pi)$ consists of the two ciphertexts $\mathsf{ctx}, \mathsf{ctx}'$ and the proof π. Verification checks that π is a proof for the statement $(m, \mathsf{ctx}, \mathsf{ctx}', \mathsf{rvk}_1, \ldots, \mathsf{rvk}_\ell)$. We will briefly argue that this scheme is still unforgeable and anonymous. First observe that if the proof π for the statement $(m, \mathsf{ctx}, \mathsf{ctx}', \mathsf{rvk}_1, \ldots, \mathsf{rvk}_\ell)$ verifies, then by the perfect soundness of the NIWI proof system either ctx or ctx′ must encrypt a signature under a public key ek_{i^*} or $\mathsf{ek}_{i^\dagger}$ respectively. Therefore, we can again construct a reduction which knows all the secret keys corresponding to the ek_i. This way, the reduction will be able to decrypt the signature σ from ctx or ctx′. To show anonymity, we transform a signature computed with sk_{i_0} into a signature computed with sk_{i_1} via a sequence of hybrids. In the first hybrid step we will make ctx′, which was uniformly random before, an encryption of a signature σ_1 of m with respect to the key vk_{i_1} under the public key ek_1. This change is possible as the ciphertexts of PKE are pseudorandom. Next, we will use witness indistinguishability of NIWI to switch the witness for the statement $(m, \mathsf{ctx}, \mathsf{ctx}', \mathsf{rvk}_1, \ldots, \mathsf{rvk}_\ell)$. The new witness shows that ctx′ encrypts a valid signature of m. This means that we do not need a witness for ctx anymore. Thus, in the next hybrid steps, we replace the ciphertext ctx by a random string, and then replace this random string by an encryption of the signature σ_1 under the public key ek_{i_1}. In the next steps, we can switch the witness we use to compute the proof π back to using the witness for ctx, and in the last hybrid we make ctx′ uniformly random again. Thus, ς is now computed using sk_{i_1}.

Compressing the membership proof. The bigger challenge, however, is reducing the size of the membership proof to linear in $\log(\ell)$. A natural approach would be to prove membership of the verification key rvk_i in the ring via a Merkle tree accumulator (as e.g. in the ROM-scheme of [Dod+04]). In this approach, one first hashes the ring R into a succinct digest ζ, and can then prove membership of rvk_i in the ring via a $\log(\ell)$-sized root-to-leaf path. To sign a message under a ring R, the signer first hashes R into a digest ζ and computes a NIWI proof π which simultaneously proves membership of their own key rvk_i in R via a succinct membership witness *and* that ctx encrypts a signature for rvk_i. To verify such a signature, the verifier recomputes the root hash ζ for the ring R and verifies the proof π. While this idea seems to resolve the above issue at first glance, it raises serious issues itself. First and foremost, we will not be able to prove unforgeability as above, as membership proofs for Merkle trees only have computational soundness, but in order to prove unforgeability as above we need perfect soundness. The problem is that an adversary might also produce a proof by finding a collision in the Merkle tree instead of forging a signature. If, in fact, we could use an NIZK proof of knowledge, then this proof strategy can be implemented with routine techniques. NIZK proofs however need a setup, and we only have NIWI proofs at our disposal. Moreover, for a Merkle tree to be binding it is necessary that the hashing key is honestly generated, as unkeyed hash functions are insecure against non-uniform adversaries, which could have a collision as part of their advice. Thus, it is also unclear where the hashing key for the Merkle tree should come from. Consequently, the Merkle tree approach seems fundamentally stuck in the standard model.

There is, however, a loophole in the above argument. Upon closer inspection, we actually do not need the Merkle tree hash function to be collision resistant. Instead, we need a guarantee that the hash value ζ binds to at least one specific value in the database, which is under the control of the signer. The key ingredient we use to make the construction work is *somewhere statistically binding* (SSB) hashing [HW15]. An SSB hash function allows to compress a database into a digest ζ such that ζ uniquely binds to a specific database entry, in our case to a verification key in a ring. More specifically, the key generator for an SSB hash function takes as an additional input an index i^* and produces a hashing key hk. When a database db is hashed into a digest ζ using the hashing key hk, the digest ζ uniquely defines db_{i^*}. In other words, any database db' with $\mathsf{db}'_{i^*} \neq \mathsf{db}_{i^*}$ hashes to a digest $\zeta' \neq \zeta$. To enable short membership proofs, we require an SSB hash function with local opening. That is, given a hashing key hk, a digest ζ of a database db, an index i and a value x, there is witness τ of size linear in $\log(|\mathsf{db}|)$ which demonstrates that $\mathsf{db}_i = x$. Besides the somewhere statistically binding property, we also require that the SSB hash function is index-hiding, i.e. the hashing key hk computationally hides the index i at which it is binding. Finally, as there is no trusted setup which could define the key for the SSB hash function, we must let the signer generate the hashing key hk itself. This introduces an additional problem, the standard notion of SSB hashing requires that the somewhere binding property holds with overwhelming probability over the coins of the key generator, but not with probability 1. However, as we let the signer generate the hashing key, the signer may in fact choose bad random coins for which the hashing key is not binding. We address this problem by using *somewhere perfectly binding* (SPB) hashing instead of SSB hashing. In fact, many constructions of SSB hashing are already SPB, e.g. the LWE-based construction of [HW15] can be made SPB via standard error-truncation techniques, and the DDH- and DCR-based constructions of [Oka+15] are immediately SPB. One additional aspect we require is that generating a hashing key hk for a database db of size ℓ can be performed by a circuit of size linear in $\log(\ell)$, but this is the case for the instantiations above. Equipped with SPB hashing, we can now construct succinct membership proofs with perfect soundness as follows. The signer generates a hashing key hk binding at position i (where rvk_i is the signer's verification key) and uses hk to compress R into a digest ζ. The membership witness shows that hk is binding at position i and that ζ opens to rvk_i at position i. Essentially, a pair (hk, ζ) of SPB hashing key hk and digest ζ form a perfectly binding commitment to rvk_i, where we can prove that (hk, ζ) opens to rvk_i at position i using a witness of size linear in $\log(\ell)$.

Relaxing the requirements on SPB hashing. It turns out that we do not need the opening witnesses for the SPB hashing scheme to be publicly computable. Indeed, we may allow the opening witness to depend on the private coins used by the key generator as we need to prove that hk is binding at position i anyway. We therefore define a slightly weakened notion called *Somewhere Perfectly Binding Hashing with private local Opening*. As observed in [Oka+15], this notion can immediately be realized from any *private information retrieval* (PIR) scheme with fully efficient client (i.e. the clients overhead is logarithmic in the database-size). Such a PIR scheme can be immediately constructed from fully homomorphic encryption [Gen09; BV11; GSW13], avoiding the Merkle tree based approach of [HW15].

Our Scheme. Armed with these techniques, we can now provide our ring signature scheme. Key generation is as described above. To sign a message m with a signing key sk_i, the signer computes a signature σ on m using sk_i and encrypts σ under ek_i obtaining a ciphertext ctx. The ciphertext ctx' is chosen uniformly at random (as in the scheme above). The signer now generates two hashing keys hk and hk' which are binding at position i and computes the hash of $\mathsf{R} = (\mathsf{rvk}_1, \ldots, \mathsf{rvk}_\ell)$ under both hk and hk', obtaining hash values ζ and ζ'. Finally, the signer computes a NIWI proof π which proves that either (hk, ζ) binds to a key rvk_i and that ctx encrypts a signature of m for rvk_i *or* (hk', ζ') bind to a key $\mathsf{rvk}_{i'}$ and that ctx' encrypts a signature of m for $\mathsf{rvk}_{i'}$. The signer then outputs the signature $\varsigma = (\mathsf{ctx}, \mathsf{ctx}', \mathsf{hk}, \mathsf{hk}', \pi)$. To verify a signature $\varsigma = (\mathsf{ctx}, \mathsf{ctx}', \mathsf{hk}, \mathsf{hk}', \pi)$ for a message m and a ring $\mathsf{R} = (\mathsf{rvk}_1, \ldots, \mathsf{rvk}_\ell)$, the verifier first computes the hashes ζ and ζ' of R using hk and hk' respectively. Now it checks if the NIWI proof π verifies for $(m, \mathsf{ctx}, \mathsf{ctx}', \mathsf{hk}, \mathsf{hk}', \zeta, \zeta')$, and if so it outputs 1. Unforgeability of this scheme is established in the same way as described above: If the proof π verifies, then by the somewhere perfectly binding property of SPB and the perfect soundness of the NIWI proof, one of the two ciphertexts $\mathsf{ctx}, \mathsf{ctx}'$ must encrypt a valid signature. The unforgeability reduction can now recover this signature by setting up the ek_i such that it knows a secret key for each of them and can therefore recover a forgery. The idea of establishing anonymity can be outlined as follows. From a high level proof perspective, SPB hashing allows us to *collapse* a ring R of ℓ verification keys into a ring of just two keys. In other words, we only care about the keys to which (hk, ζ) and (hk', ζ') bind. With this in mind, we can essentially implement the same proof strategy as before, pretending that our ring just consists of two keys. As before, we will transform a signature computed using a signing key sk_{i_0} into a signature computed using sk_{i_1} via a sequence of hybrids. In the first hybrid, we use the index-hiding property of the SPB hash function to move the binding index of hk' from i_0 to i_1. Next, we proceed similarly as above, namely compute a signature σ' using sk_{i_1} and encrypt σ' under ek_{i_1} obtaining a ciphertext ctx'. Indistinguishability of this hybrid from the previous hybrid can be argued via the pseudorandom ciphertexts property of PKE. In the next step, we switch the witness used to compute the NIWI proof π. That is, instead of proving that ctx encrypts a valid signature under ek_{i_0}, we prove that ctx' encrypts a valid signature under ek_{i_1}. Both are valid witnesses as we are proving an or-statement. Therefore, witness indistinguishability of NIWI yields that this hybrid is indistinguishable from the last one. We can now perform the same hybrid modifications to hk and ctx and finally switch the witness again. Therefore, in the last hybrid we get a signature ς computed using sk_{i_1}.

5.1.2 On Linkable Ring Signatures

Linkable Ring Signatures are an extension of ring signatures, which allow signatures by the same signer to be linked. This requirement emerged in the context of electronic voting, where a vote would be cast via a ring signature in the name of all eligible voters. Then, a linking algorithm could prevent one voter from casting more than one vote, or could indicate a change in preference over a series of votes. The presence of the linking algorithm naturally diminishes anonymity and several ideas exist how the spirit of unlinkable anonymity could be kept in linkable ring signatures.

Definitions of Linkable Anonymity. The exact definition of linkable anonymity seems to vary between different authors. However, it seems that all these definitions assume that there always remain *unspent* verification keys in an anonymity set. Take for instance the definition of linkable anonymity in [LAZ19] (Definition 10 on page 13). Their definition of linkable anonymity is the same as the definition of unlinkable anonymity, with the difference that the adversary is not given access to a signing oracle. We propose a simple definition for linkable anonymity similar in spirit to the blindness definition of blind signatures. The experiment is identical to the anonymity experiment for unlinkable ring signatures, with the following modification:

- The adversary is not allowed to corrupt the challenge keys rvk_{i_0} and rvk_{i_1}.
- In the challenge phase, the adversary submits two message-ring pairs (m_0, R_0) and (m_1, R_1) such that both R_0 and R_1 contain both rvk_{i_0} and rvk_{i_1}.
- The experiment flips a bit $b \leftarrow_{\$} \{0,1\}$, computes $\varsigma_0 \leftarrow \mathsf{Sign}(\mathsf{rsk}_{i_b}, m_0, \mathsf{R}_0)$ and $\varsigma_1 \leftarrow \mathsf{Sign}(\mathsf{rsk}_{i_{1-b}}, m_1, \mathsf{R}_1)$ and returns $(\varsigma_0, \varsigma_1)$ to the adversary.
- The adversary must now guess bit b.

Note that the signature ς_0 is computed exactly as in the experiment for unlinkable anonymity, but now we additionally provide the adversary with a signature ς_1 computed with the signing key $\mathsf{rsk}_{i_{1-b}}$. Consequently, this definition immediately implies e.g. the definition of [LAZ19], but does not impose the restriction that no signatures under $\mathsf{rvk}_{i_{1-b}}$ can be issued. Like the blindness definition for blind signatures, our definition naturally extends to larger challenge spaces, i.e. considering challenges of size 2 is complete.

A Linkable Ring Signature Scheme. We will now extend our techniques to the setting of linkable ring signatures. The underlying idea is rather basic. Every verification key rvk contains a commitment com to a random tag τ. When a signer signs a message m, they include τ into the signature ς and prove that com unveils to τ. This proof can naturally be included in the NIWI proof for the validity of the encrypted signature. Now, whenever a secret key rsk is used to sign a message m, its corresponding tag τ is spent. Thus, we can link signatures by checking whether they have the same tag.

While this idea seems to check out at first glance, we run into trouble when trying to prove linkable anonymity. In the linkable anonymity experiment the adversary gets to see the tags of both challenge signatures. This means the reduction must be able to provide witnesses that both the commitment in rvk_{i_0} and the commitment in rvk_{i_1} open to the respective tags τ_{i_0} and τ_{i_1}. The fact that we need to be able to open both commitments, however, makes it apparently impossible to use the hiding property of the commitments in order to flip the challenge bit in the security proof. Once again, the situation could be resolved easily if we had NIZK proofs at our disposal, yet we can only use witness indistinguishability.

Our way out of this conundrum is based on the following observation. To achieve linkability, we do not actually need that every verification key has a unique tag. Instead, a weaker condition is sufficient. Namely, for a ring of size ℓ it should not be possible to generate $\ell + 1$ valid signatures with pairwise distinct tags. We leverage this idea by allowing the commitments in the verification keys to be malformed *in a controlled way*. More specifically, instead of putting only one commitment to a tag τ in the verification key rvk, we put 3 commitments to τ in rvk.

As before, each signature contains two hashing keys and two hash values. Moreover, in the linkable anonymity proof we will set up things in a way such that for both challenge signatures ς_0 and ς_1, one of (hk, ζ) and (hk', ζ') will point to rvk_{i_0} and the other one to rvk_{i_1}. Assume that a signature ς contains a tag τ and that the SPB hash (hk, ζ) points to rvk_i, whereas (hk', ζ') points to $\mathsf{rvk}_{i'}$. We will make the following consistency requirement: If $i = i'$ we will require that all three commitments in rvk_i unveil to the same tag τ. However, if $i \neq i'$, then we only require that out of the six commitments in rvk_i and $\mathsf{rvk}_{i'}$ that

- at least two unveil to τ,
- at least two unveil to a tag $\tau' \neq \tau$,
- at most one commitment does not unveil correctly.

This relaxed binding condition now allows us to exchange the tags of rvk_i and $\mathsf{rvk}_{i'}$ even though we are handing out signatures which use these tags! We prove linkable anonymity via a sequence of hybrids. As above, it is instructive to think that SPB hashing collapses a ring R of ℓ keys into a ring of just two verification keys. Call these verification keys rvk_0 and rvk_1. In the linkable anonymity experiment, there are two signatures, ς_0 and ς_1 for m_0 and m_1 respectively. In the first hybrid the challenge bit of the experiment is 0, that is ς_0 is computed using the signing key rsk_0 whereas ς_1 is computed using rsk_1. In the final experiment, ς_0 will be computed using rsk_1 and ς_1 will be computed using rsk_0. The critical part of this proof is to switch the tags. Our proof strategy relies critically on the fact that the tags τ_0 and τ_1 are identically distributed. Namely, we will not switch the tags in the signatures, but switch the tags in the verification keys. More specifically, in the first hybrid rvk_{i_0} contains commitments to τ_0 and rvk_{i_1} contains commitments to tag τ_1. In the last hybrid, rvk_0 will commit to τ_1 and rvk_1 will commit to τ_0. But since the tags are identically distributed we can now simply rename them. Therefore, this hybrid is identical to the linkable anonymity experiment with challenge bit 1. In a first step we make both signatures ς_0 and ς_1 use both keys rvk_0 and rvk_1 by modifying the binding indices in hk' appropriately for both signatures. Now, our relaxed binding condition allows us to exchange the tags between rvk_0 and rvk_1 one by one. That is, the relaxed binding condition allows us to *forget* the unveil information of one of the six commitments in rvk_0 and rvk_1. Say we forget the unveil information of the first commitment in rvk_0. We can then turn this commitment into a commitment of τ_1. Next, we change the first commitment in rvk_1 into a commitment of τ_0. We continue like this alternating between rvk_0 and rvk_1, until we have completely swapped τ_0 and τ_1. Note that in each step the relaxed binding condition holds, thus we can argue via witness indistinguishability and the hiding property of the underlying commitments. Finally, using random tags τ alone does not achieve the strongest notion of non-frameability, where the adversary is allowed to *steal* tags. Thus, we use an idea due to Dolev, Dwork and Naor [DDN91] commonly used to achieve non-malleability[2]: We replace the tag τ by the verification key vk of a signature scheme DS and additionally sign (m, ς) with respect to vk. This, however, has the somewhat surprising consequence that we do not need the encrypted signatures anymore, we can rely entirely on the unforgeability of DS!

5.1.3 Contributions in This Chapter

In this chapter, we provide the first construction of ring signatures which simultaneously

- does not rely on a trusted setup or the random oracle heuristic,
- can be proven secure under *falsifiable* standard assumptions, namely the existence of non-interactive witness-indistinguishable proofs [DN00b; BOV03; GOS06; BP15] and additional standard assumptions such as the hardness of the *Decisional Diffie Hellman* problem [ElG84] or the *Learning with Errors* problem [Reg05],
- has signatures of size $\log(\ell) \cdot \mathsf{poly}(\lambda)$, where ℓ is the size of the ring of signers and λ the security parameter.

Furthermore, we extend our techniques to the domain of linkable ring signatures, i.e. we construct linkable ring signatures of size $\log(\ell) \cdot \mathsf{poly}(\lambda)$ without setup and in the standard model. To avoid any trusted setup, or random oracles, our constructions cannot rely on non-interactive zero-knowledge (NIZK) proofs and instead use non-interactive witness-indistinguishable (NIWI) proofs which do not require trusted setup. The techniques that enable us to use NIWI proofs instead of NIZK proofs may be of independent interest.

As an additional contribution, we propose a strengthened security model for linkable ring signatures and prove that our linkable ring signature scheme is secure in this model.

5.2 Chapter Preliminaries

In this section, we introduce preliminaries relevant in the constructions and arguments of this chapter. In particular, we recall the notions of non-interactive commitment schemes and somewhere perfectly binding hashing.

[2]e.g. in the construction of IND-CCA secure encryption schemes, or recently in the context of cryptocurrencies [Ruf+18].

5.2.1 Non-interactive Proof Systems

We make the following efficiency assumption about proof systems we consider in this chapter.

Definition 5.1: Efficiency of Non-interactive Proof Systems

For $\pi = \mathsf{NIP.Prove}(1^\lambda, \mathsf{x}, \mathsf{w})$ it holds that $|\pi| = |C_\mathsf{x}| \cdot \mathsf{poly}(\lambda)$, where C_x is a verification circuit for the statement x, i.e. $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}$ iff $C_\mathsf{x}(\mathsf{w}) = 1$.

5.2.2 Non-interactive Commitment Schemes

Definition 5.2: Non-interactive Commitment

$\mathsf{Commit}(1^\lambda, m) \to (\mathsf{com}, \gamma)$
: Takes as input a security parameter 1^λ and a message m. Outputs a pair com, γ of commitment and decommitment information.

$\mathsf{Decommit}(\mathsf{com}, m, \gamma) \to r \in \{\mathsf{true}, \mathsf{false}\}$
: Takes as input a commitment com, a message m and a piece of decommitment information γ. Outputs either true or false.

A commitment scheme Com is *correct*, if it holds for all security parameters $\lambda \in \mathbb{N}$ and all messages $m \in \mathcal{M}$ that given $(\mathsf{com}, \gamma) \leftarrow \mathsf{Commit}(1^\lambda, m)$, we have that $\mathsf{Verify}(\mathsf{com}, m, \gamma) = \mathsf{true}$.

We require the following properties of a non-interactive commitment scheme.

Definition 5.3: Perfect Binding

For a non-interactive commitment scheme Com consider the binding experiment between a challenger and an adversary $\mathcal{A}$.

```
$\mathsf{Binding}^{\mathcal{A}}_{\mathsf{Com}}(1^\lambda)$

$(\mathsf{com}, m_0, \gamma_0, m_1, \gamma_1) \leftarrow \mathcal{A}(1^\lambda)$
**if** $m_0 = m_1$ **then return** false
**if** $\mathsf{Com.Decommit}(\mathsf{com}, m_0, \gamma_0) = \mathsf{true}$
    **and** $\mathsf{Com.Decommit}(\mathsf{com}, m_1, \gamma_1) = \mathsf{true}$
 **then return** true
**else return** false
```

We define the advantage of $\mathcal{A}$ in the above experiment as

$$\mathsf{Adv}^{\mathsf{Binding}}_{\mathcal{A},\mathsf{Com}}(\lambda) := \Pr\left[\mathsf{Binding}^{\mathcal{A}}_{\mathsf{Com}}(1^\lambda) \Rightarrow \mathsf{true}\right].$$

We say that a commitment scheme Com is *perfectly binding* if for all $\lambda \in \mathbb{N}$ and all unbounded adversaries $\mathcal{A}$ we have

$$\mathsf{Adv}^{\mathsf{Binding}}_{\mathcal{A}}(\lambda) = 0.$$

Definition 5.4: Computational Hiding

For a non-interactive commitment scheme Com consider the hiding experiment between a challenger and a two-stage adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$.

```
$\mathsf{Hiding}^{\mathcal{A}}_{\mathsf{Com}}(1^\lambda)$
$(\mathsf{state}, m_0, m_1) \leftarrow \mathcal{A}_0(1^\lambda)$
$b \leftarrow \{0,1\}$
$(\mathsf{com}, \gamma) \leftarrow \mathsf{Com.Commit}(1^\lambda, m_b)$
$b' \leftarrow \mathcal{A}_1(\mathsf{state}, \mathsf{com})$
if $b' = b$ then return true
else return false
```

We define the advantage of $\mathcal{A}$ in the above experiment as

$$\mathsf{Adv}^{\mathsf{Hiding}}_{\mathcal{A},\mathsf{Com}}(\lambda) := \left|\Pr\left[\mathsf{Hiding}^{\mathcal{A}}_{\mathsf{Com}}(1^\lambda) \Rightarrow \mathsf{true}\right] - \frac{1}{2}\right|.$$

We say that a commitment scheme Com is *computationally hiding* if for all $\lambda \in \mathbb{N}$ and all PPT adversaries $\mathcal{A}$ we have

$$\mathsf{Adv}^{\mathsf{Hiding}}_{\mathcal{A},\mathsf{Com}}(\lambda) \leq \mathsf{negl}(\lambda).$$

Non-interactive commitment schemes can be constructed from any injective one-way function via the Goldreich-Levin hardcore bit [GL89].

5.2.3 Public Key Encryption

We introduce strengthenings of the security properties described in section 2.2.2.

Definition 5.5: Pseudorandom Public Keys

We require that public keys are computationally indistinguishable from uniform.

Definition 5.6: Pseudorandom Ciphertexts

We require that it holds for every message m that

$$(\mathsf{ek}, u) \approx_c (\mathsf{ek}, \mathsf{Enc}(\mathsf{ek}, m)),$$

where ek and u are chosen uniformly at random.

We denote the advantages of $\mathcal{A}$ in breaking pseudorandom public keys and pseudorandom ciphertexts as $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND\text{-}PK}}(\lambda)$ and $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND\text{-}ENC}}(\lambda)$, respectively. Note that the pseudorandom public keys and pseudorandom ciphertext properties together immediately imply the standard notion of IND-CPA security.

Such public key encryption schemes can be constructed e.g. from the DDH problem [ElG84] or the LWE problem [Reg05].

5.2.4 Somewhere Perfectly Binding Hashing

Somewhere statistically binding (SSB) hashing [HW15] allows a negligible fraction of hashing-keys to be non-binding. For our constructions we actually only require something slightly weaker, a primitive we call *somewhere perfectly binding hashing with private local opening*. This notion relaxes the definition of somewhere perfectly binding hashing in that we allow the Gen algorithm to output a private key shk which the Open algorithm takes as additional input. Below we give our relaxed definition which we use throughout this chapter.

Definition 5.7: Hash Family with Private Local Opening

$\mathsf{Gen}(1^\lambda, n, \mathsf{ind}) \to (\mathsf{hk}, \mathsf{shk})$
: Takes as input a security parameter 1^λ, a database size n and an index ind.[a] Outputs a hashing key hk and a private key shk.

$\mathsf{Hash}(\mathsf{hk}, \mathsf{db}) \to \zeta$
: Takes as input a hashing key hk and a database db. Outputs a digest ζ.

$\mathsf{Open}(\mathsf{hk}, \mathsf{shk}, \mathsf{db}, \mathsf{ind}) \to \tau$
: Takes as input a hashing key hk, a private key shk a database db and an

index ind. Outputs a witness τ.

$\mathsf{Verify}(\mathsf{hk}, \zeta, \mathsf{ind}, x, \tau) \to r \in \{\mathsf{true}, \mathsf{false}\}$
Takes as input a hashing key hk, a digest ζ, an index ind, a value x and a witness τ. Outputs either true or false.

[a]To simplify notation, we will usually not provide the block size of databases as an input to SPB.Gen but rather assume that the block size for the specific application context is hardwired in this function.

We say that SPB = (Gen, Hash, Open, Verify) is *correct*, if it holds for all $\lambda \in \mathbb{N}$, all $n = \mathsf{poly}(\lambda)$, all databases db of size n and all indices $\mathsf{ind} \in [n]$ that given that

$$\begin{aligned}(\mathsf{hk}, \mathsf{shk}) &\leftarrow \mathsf{SPB.Gen}(1^\lambda, n, \mathsf{ind}),\\ \zeta &\leftarrow \mathsf{SPB.Hash}(\mathsf{hk}, \mathsf{db}) \text{ and}\\ \tau &\leftarrow \mathsf{SPB.Open}(\mathsf{hk}, \mathsf{shk}, \mathsf{db}, \mathsf{ind}),\end{aligned}$$

it holds that

$$\Pr[\mathsf{SPB.Verify}(\mathsf{hk}, \zeta, \mathsf{ind}, \mathsf{db}_{\mathsf{ind}}, \tau) = \mathsf{true}] = 1.$$

Definition 5.8: Efficiency of Hashing

A hash family with local opening SPB is *efficient*, if the hashing keys hk generated by $\mathsf{Gen}(1^\lambda, n, \mathsf{ind})$ and the witnesses τ generated by $\mathsf{Open}(\mathsf{hk}, \mathsf{shk}, \mathsf{db}, \mathsf{ind})$ are of size $\log(n) \cdot \mathsf{poly}(\lambda)$. Moreover, $\mathsf{Verify}(\mathsf{hk}, \zeta, \mathsf{ind}, x, \tau)$ can be computed by a circuit of size $\log(n) \cdot \mathsf{poly}(\lambda)$.

Definition 5.9: Somewhere Perfectly Binding

A hash family with local opening SPB is *somewhere perfectly binding*, if it holds for all $\lambda \in \mathbb{N}$, all $n = \mathsf{poly}(\lambda)$, all databases db of size n, all indices $\mathsf{ind} \in [n]$, all database values x and all witnesses τ that if

$$\begin{aligned}\zeta &\leftarrow \mathsf{SPB.Hash}(\mathsf{hk}, \mathsf{db}) \text{ and}\\ &\mathsf{Verify}(\mathsf{hk}, \zeta, \mathsf{ind}, x, \tau) = \mathsf{true},\end{aligned}$$

then it holds that $x = \mathsf{db}_{\mathsf{ind}}$.

Notice that this definition provides a stronger somewhere perfectly binding guarantee in that we do not have to require that hk has been generated correctly.

Definition 5.10: Computational Index Hiding

For any hash family with private local opening SPB consider the index-hiding experiment Index-Hiding between a challenger and a two-stage adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$:

```
Index-Hiding$^{\mathcal{A}}_{\mathsf{SPB}}(1^\lambda)$

$(\mathsf{state}, n, \mathsf{ind}_0, \mathsf{ind}_1) \leftarrow \mathcal{A}_0(1^\lambda)$
$b \leftarrow \{0,1\}$
$(\mathsf{hk}, \mathsf{shk}) \leftarrow \mathsf{SPB.Gen}(1^\lambda, n, \mathsf{ind}_b)$
$b' \leftarrow \mathcal{A}_1(\mathsf{state}, \mathsf{hk})$
if $b' = b$
 then return true
else return false
```

We define the advantage of $\mathcal{A}$ in this experiment as

$$\mathsf{Adv}^{\mathsf{Index\text{-}Hiding}}_{\mathcal{A},\mathsf{SPB}}(\lambda) := \left| \Pr\left[\mathsf{Index\text{-}Hiding}^{\mathcal{A}}_{\mathsf{SPB}}(1^\lambda) \Rightarrow \mathsf{true}\right] - \frac{1}{2} \right| .$$

A hash family with local opening SPB is *computationally index hiding*, if for any PPT adversary $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathsf{Index\text{-}Hiding}}_{\mathcal{A},\mathsf{SPB}}(\lambda) \leq \mathsf{negl}(\lambda) .$$

We can immediately construct an SPB hash family SPB with private local opening from any SPB hash family SPB′ with local opening via the following construction.

Construction 5.1: Private-Opening SPB

```
SPB.Gen$(1^\lambda, n, \mathsf{ind})$

$r \leftarrow \{0,1\}^\lambda$
$\mathsf{hk} \leftarrow \mathsf{SPB'.Gen}(1^\lambda, n, \mathsf{ind}; 1^\lambda; r)$
$\mathsf{shk} := r$
return $(\mathsf{hk}, \mathsf{shk})$
```

```
SPB.Hash$(\mathsf{hk}, \mathsf{db})$

return $\mathsf{SPB'.Hash}(\mathsf{hk}, \mathsf{db})$
```

```
SPB.Open(hk, shk, db, ind)
$\tau' \leftarrow \mathsf{SPB}'.\mathsf{Open}(\mathsf{hk}, \mathsf{db}, \mathsf{ind})$
$\tau \leftarrow (\tau', \mathsf{shk})$
return $\tau$
```

```
SPB.Verify(hk, ζ, ind, x, τ)
parse $\tau = (\tau', r)$
$\hat{\mathsf{hk}} \leftarrow \mathsf{SPB}'.\mathsf{Gen}(1^\lambda, n, \mathsf{ind}; r)$
if $\hat{\mathsf{hk}} \neq \mathsf{hk}$ then
  return false
else
  return $\mathsf{SPB}'.\mathsf{Verify}(\mathsf{hk}, \zeta, \mathsf{ind}, x, \tau')$
```

Correctness and index-hiding of SPB follow directly from the corresponding properties of SPB′, the somewhere perfectly binding property follows from the fact that SPB.Verify ensures explicitly that hk is perfectly binding at index ind. Consequently, also this property follows from the corresponding property of SPB′. Moreover, we can also realize a SPB hash family with private local opening from any 2-message private information retrieval scheme with fully efficient verifier and perfect correctness. This was also observed by [Oka+15]. The construction is straightforward: A hashing key hk for index i consists of the PIR receiver message, to hash a database db run the PIR sender algorithm on hk and db. The index-hiding property follows by PIR receiver privacy, whereas the SPB property follows form perfect correctness. Finally, the receivers private coins serve as succinct private membership witness.

5.3 Logarithmic Size Ring-Signatures

In this section we will provide a construction of a ring signature scheme. Let

- PKE = (KeyGen, Enc, Dec) be a public key encryption scheme with pseudorandom keys and ciphertexts,
- DS = (KeyGen, Sign, Verify) be a signature scheme,
- SPB = (Gen, Hash, Open, Verify) be a somewhere perfectly binding hash function with private local opening and,
- NIP = (Prove, Verify) be a NIWI-proof system for the language $\mathcal{L}$ defined as follows. We define a witness-relation $\mathcal{R}$: If $x = (m, \mathsf{ctx}, \mathsf{hk}, \zeta)$ and $\mathsf{w} = (\mathsf{rvk}, \mathsf{ind}, \tau, \sigma, r_{\mathsf{ctx}})$, where $\mathsf{rvk} = (\mathsf{rvk}, \mathsf{ek})$, let

$$\begin{aligned}\mathcal{R}(x, \mathsf{w}) \Leftrightarrow\ & \mathsf{SPB.Verify}(\mathsf{hk}, \zeta, \mathsf{ind}, \mathsf{rvk}, \tau) = \mathsf{true}\\ & \text{and } \mathsf{PKE.Enc}(\mathsf{ek}, \sigma; r_{\mathsf{ctx}}) = \mathsf{ctx}\\ & \text{and } \mathsf{DS.Verify}(\mathsf{vk}, m, \sigma) = \mathsf{true}\end{aligned}$$

and let $\mathcal{L}'$ be the language accepted by $\mathcal{R}$. Now, define the language $\mathcal{L}$ by

$$\mathcal{L} = \{(m, \mathsf{ctx}_1, \mathsf{ctx}_2, \mathsf{hk}_1, \mathsf{hk}_2, \zeta_1, \zeta_2) \mid (m, \mathsf{ctx}_1, \mathsf{hk}_1, \zeta_1) \in \mathcal{L}' \text{ or } (m, \mathsf{ctx}_2, \mathsf{hk}_2, \zeta_2) \in \mathcal{L}'\}.$$

Our ring signature scheme RS = (KeyGen, Sign, Verify) is given as follows.

Construction 5.2: Logarithmic Ring Signatures

```
RS.KeyGen($1^\lambda$)
$(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{DS.KeyGen}(1^\lambda)$
$\mathsf{ek} \leftarrow [\mathsf{PKE.KeyGen}(1^\lambda)]$
$\mathsf{rvk} := (\mathsf{vk}, \mathsf{ek})$
$\mathsf{rsk} := (\mathsf{sk}, \mathsf{rvk})$
return $(\mathsf{rvk}, \mathsf{rsk})$
```

```
RS.Verify($\mathsf{R}, m, \varsigma$)
parse $\varsigma = (\mathsf{ctx}_1, \mathsf{ctx}_2, \mathsf{hk}_1, \mathsf{hk}_2, \pi)$
$\zeta_1' \leftarrow \mathsf{SPB.Hash}(\mathsf{hk}_1, \mathsf{R})$
$\zeta_2' \leftarrow \mathsf{SPB.Hash}(\mathsf{hk}_2, \mathsf{R})$
$\mathsf{x} := (m, \mathsf{ctx}_1, \mathsf{ctx}_2, \mathsf{hk}_1, \mathsf{hk}_2, \zeta_1', \zeta_2')$
return $\mathsf{NIP.Verify}(\mathsf{x}, \pi)$
```

```
RS.Sign($\mathsf{rsk}, m, \mathsf{R}$)
parse $\mathsf{rsk} = (\mathsf{sk}, \mathsf{rvk}); \mathsf{rvk} = (\mathsf{vk}, \mathsf{ek})$
$\sigma \leftarrow \mathsf{DS.Sign}(\mathsf{sk}, m)$
$\mathsf{ind} := i \in [|\mathsf{R}|]$ such that $\mathsf{rvk}_i = \mathsf{rvk}$
$(\mathsf{hk}_1, \mathsf{shk}_1) \leftarrow \mathsf{SPB.Gen}(1^\lambda, |\mathsf{R}|, \mathsf{ind})$
$(\mathsf{hk}_2, \mathsf{shk}_2) \leftarrow \mathsf{SPB.Gen}(1^\lambda, |\mathsf{R}|, \mathsf{ind})$
$\zeta_1 \leftarrow \mathsf{SPB.Hash}(\mathsf{hk}_1, \mathsf{R})$
$\zeta_2 \leftarrow \mathsf{SPB.Hash}(\mathsf{hk}_2, \mathsf{R})$
$\tau \leftarrow \mathsf{SPB.Open}(\mathsf{hk}_1, \mathsf{shk}_1, \mathsf{R}, \mathsf{ind})$
$\mathsf{ctx}_1 \leftarrow \mathsf{PKE.Enc}(\mathsf{ek}, \sigma; r_{\mathsf{ctx}})$
$\mathsf{ctx}_2 \leftarrow \{0,1\}^\lambda$
$\mathsf{x} := (m, \mathsf{ctx}_1, \mathsf{ctx}_2, \mathsf{hk}_1, \mathsf{hk}_2, \zeta_1, \zeta_2)$
$\mathsf{w} \leftarrow (\mathsf{rvk}, \mathsf{ind}, \tau, \sigma, r_{\mathsf{ctx}})$
$\pi \leftarrow \mathsf{NIP.Prove}(\mathsf{x}, \mathsf{w})$
return $\varsigma \leftarrow (\mathsf{ctx}_1, \mathsf{ctx}_2, \mathsf{hk}_1, \mathsf{hk}_2, \pi)$
```

Correctness. We will first show that our scheme is correct. Assume that $\mathsf{rvk} = (\mathsf{vk}, \mathsf{ek})$ and $\mathsf{rsk} = (\mathsf{sk}, \mathsf{rvk})$ were generated by RS.KeyGen and $\varsigma = (\mathsf{ctx}_1, \mathsf{ctx}_2, \mathsf{hk}_1, \mathsf{hk}_2, \pi)$ is the output of $\mathsf{RS.Sign}(\mathsf{rsk}, m, \mathsf{R})$, where $\mathsf{R} = (\mathsf{rvk}_1, \ldots, \mathsf{rvk}_\ell)$. We will show that it holds that $\mathsf{RS.Verify}(\mathsf{R}, m, \varsigma) = \mathsf{true}$. First note that since SPB.Hash is deterministic, it holds that $\zeta_1' = \zeta_1$ and $\zeta_2' = \zeta_2$. Also, it holds that $\mathsf{rvk} = \mathsf{rvk}_{\mathsf{ind}}$ (where ind is the index of rvk in R). Now, notice further that by the correctness of SPB it holds that $\mathsf{SPB.Verify}(\mathsf{hk}_1, \zeta_1, \mathsf{ind}, \mathsf{rvk}_{\mathsf{ind}}, \tau) = \mathsf{true}$. Moreover, by the correctness of DS it holds that $\mathsf{DS.Verify}(\mathsf{vk}, m, \sigma) = \mathsf{true}$. Consequently, $(m, \mathsf{ctx}_1, \mathsf{ctx}_2, , \mathsf{hk}_1, \mathsf{hk}_2, \zeta_1, \zeta_2) \in \mathcal{L}$ and $\mathsf{w} = (\mathsf{rvk}, \mathsf{ind}, \tau, \sigma, r_{\mathsf{ctx}})$ is a witness for membership. Thus, by the correctness of NIP it holds that

$$\mathsf{NIP.Verify}((m, \mathsf{ctx}_1, \mathsf{ctx}_2, \mathsf{hk}_1, \mathsf{hk}_2, \zeta_1, \zeta_2), \pi) = \mathsf{true}$$

and consequently $\mathsf{RS.Verify}(\mathsf{R}, m, \varsigma)$ outputs true.

Signature Size. For a signature $\varsigma = (\mathsf{ctx}_1, \mathsf{ctx}_2, \mathsf{hk}_1, \mathsf{hk}_2, \pi)$, the size of the ciphertexts $\mathsf{ctx}_1, \mathsf{ctx}_2$ is $\mathsf{poly}(\lambda)$ and independent of the ring-size ℓ. By the efficiency property of SPB the sizes of the hashing keys $\mathsf{hk}_1, \mathsf{hk}_2$ is bounded by $\log(\ell) \cdot \mathsf{poly}(\lambda)$. Also, by the efficiency property of SPB this size of the witness τ is $\log(\ell) \cdot \mathsf{poly}(\lambda)$ and SPB.Verify can be computed by a circuit of size $\log(\ell) \cdot \mathsf{poly}(\lambda)$.

Consequently, the verification circuit C_x for the language $\mathcal{L}$ and statement $x = (m, \mathsf{ctx}_1, \mathsf{ctx}_2, \mathsf{hk}_1, \mathsf{hk}_2, \zeta_1, \zeta_2)$ has size $\log(\ell) \cdot \mathsf{poly}(\lambda)$. By the proof-size property of the NIWI proof it holds that $|\pi| = |C_x| \cdot \mathsf{poly}(\lambda) = \log(\ell) \cdot \mathsf{poly}(\lambda)$. All together, the size of signatures ς is $\log(\ell) \cdot \mathsf{poly}(\lambda)$.

5.3.1 Proof of Unforgeability

We will turn to showing that RS is unforgeable.

Theorem 5.1: Unforgeability of Construction 5.2

Construction 5.2 is unforgeable, if NIP is perfectly sound, SPB is somewhere perfectly binding, PKE is perfectly correct, PKE has pseudorandom public keys and DS is unforgeable.

The main idea of the proof is that since the NIWI proof has perfect soundness, it must either hold that $(m, \mathsf{ctx}_1, \mathsf{hk}_1, \zeta_1) \in \mathcal{L}'$ or $(m, \mathsf{ctx}_2, \mathsf{hk}_2, \zeta_2) \in \mathcal{L}'$. If the first statement is true, then hk_1 corresponds to an index ind_1 and $\mathcal{A}$ must have produced a forgery for a key $\mathsf{rvk}_{\mathsf{ind}_1}$ in R. Likewise, if the second statement is true, then $\mathcal{A}$ must have produced a forgery for a key $\mathsf{rvk}_{\mathsf{ind}_2}$ in R.

Proof. Let $\mathcal{A}$ be a PPT adversary against the unforgeability experiment of RS and let further $q = \mathsf{poly}(\lambda)$ an upper bound on the number of key queries of $\mathcal{A}$. Consider the following two hybrids.

$\mathcal{H}_0$: This is the real experiment.

$\mathcal{H}_1$: The same as $\mathcal{H}_0$, except that for all $i \in [q]$ the challenger generates the public keys ek_i in rvk_i by $(\mathsf{ek}_i, \hat{\mathsf{dk}}_i) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$ instead of choosing ek_i uniformly at random. Moreover, the challenger stores all the secret keys $(\hat{\mathsf{dk}}_i)_{i \in [q]}$.

We will first argue that $\mathcal{H}_0$ and $\mathcal{H}_1$ are computationally indistinguishable given that the public keys of PKE are pseudorandom.

Claim 5.1: $\mathcal{H}_0 \approx_c \mathcal{H}_1$

There exists a reduction $\mathcal{R}_1$ such that

$$\mathsf{Adv}_{\mathcal{R}_1^{\mathcal{A}}}^{\mathsf{IND\text{-}PK}}(\lambda) \geq |\Pr[\mathcal{H}_0(\mathcal{A}) = \mathsf{true}] - \Pr[\mathcal{H}_1(\mathcal{A}) = \mathsf{true}]|.$$

The reduction $\mathcal{R}_1$ is given as follows.

Reduction $\mathcal{R}_1^{\mathcal{A}}(\mathsf{ek}^*)$

- Choose an index $i^* \leftarrow_\$ [q]$ uniformly at random.
- Simulate $\mathcal{H}_0$ with the following modifications. For all indices $i < i^*$ generate $(\mathsf{rvk}_i, \mathsf{rsk}_i)$ as in $\mathcal{H}_0$. For $i > i^*$ generate $(\mathsf{rvk}_i, \mathsf{rsk}_i)$ as in $\mathcal{H}_1$.
- Generate $(\mathsf{rvk}_{i^*}, \mathsf{rsk}_{i^*})$ as follows:
  - Compute $(\mathsf{vk}_{i^*}, \mathsf{sk}_{i^*}) \leftarrow \mathsf{DS.KeyGen}(1^\lambda; r_{\mathsf{DS}})$
  - Set $\mathsf{rvk}_{i^*} \leftarrow (\mathsf{vk}_{i^*}, \mathsf{ek}^*)$ and $\mathsf{rsk}_{i^*} \leftarrow (\mathsf{sk}_{i^*}, \mathsf{rvk}_{i^*})$
- Output whatever the simulated experiment outputs.

Let $\mathcal{PK}_0$ be the uniform distribution and $\mathcal{PK}_1$ be a distribution sampled by computing $(\mathsf{ek}^*, \hat{\mathsf{dk}}^*) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$ and outputting ek^*. First observe that when $i^* = q-1$ and ek^* was chosen from $\mathcal{PK}_0$, then $\mathcal{R}_1^{\mathcal{A}}$ perfectly simulates $\mathcal{H}_0(\mathcal{A})$. On the other hand, if $i^* = 0$ and ek^* was chosen from $\mathcal{PK}_1$, then $\mathcal{R}_1^{\mathcal{A}}$ perfectly simulates $\mathcal{H}_1(\mathcal{A})$. Moreover, observe that for $j = 1, \ldots, q-1$ it holds that $\mathcal{R}_1^{\mathcal{A}}(\mathcal{PK}_0)|_{i^*=j-1}$ and $\mathcal{R}_1^{\mathcal{A}}(\mathcal{PK}_1)|_{i^*=j}$ are identically distributed. Consequently, we get that

$$\begin{aligned}
\mathsf{Adv}_{\mathcal{R}_1^{\mathcal{A}}}^{\mathsf{IND\text{-}PK}}(\lambda) &= |\Pr[\mathcal{R}_1^{\mathcal{A}}(\mathcal{PK}_0)] - \Pr[\mathcal{R}_1^{\mathcal{A}}(\mathcal{PK}_1)]| \\
&= |\sum_{j=0}^{q-1} \Pr[i^* = j] \cdot (\Pr[\mathcal{R}_1^{\mathcal{A}}(\mathcal{PK}_0)|i^* = j] - \Pr[\mathcal{R}_1^{\mathcal{A}}(\mathcal{PK}_1)|i^* = j])| \\
&= \frac{1}{q} \cdot |(\Pr[\mathcal{R}_1^{\mathcal{A}}(\mathcal{PK}_0)|i^* = q-1] - \Pr[\mathcal{R}_1^{\mathcal{A}}(\mathcal{PK}_1)|i^* = 0] \\
&\quad + \sum_{j=1}^{q-1} (\Pr[\mathcal{R}_1^{\mathcal{A}}(\mathcal{PK}_1)|i^* = j] - \Pr[\mathcal{R}_1^{\mathcal{A}}(\mathcal{PK}_0)|i^* = j-1]))| \\
&= \frac{1}{q} \cdot |(\Pr[\mathcal{H}_0(\mathcal{A}) = \mathsf{true}] - \Pr[\mathcal{H}_1(\mathcal{A}) = \mathsf{true}])|.
\end{aligned}$$

Claim 5.2: Reduction to EUF-CMA **of** DS

There exists a reduction $\mathcal{R}_2$ such that $\mathcal{R}_2^{\mathcal{A}}$ breaks the EUF-CMA security of DS with probability $\mathsf{Adv}_{\mathcal{A}}^{\mathcal{H}_1}(\lambda)/q$.

The reduction $\mathcal{R}_2$ is given as follows.

Reduction $\mathcal{R}_2^{\mathcal{A}}(\mathsf{rvk}^*)$

- Guess an index $i^* \leftarrow_\$ [q]$. For all $i \neq i^*$ generate rvk_i and rsk_i as in $\mathcal{H}_1$. Generate $\mathsf{rvk}_{i^*} = \mathsf{rvk}^*$ as follows. Generate $(\mathsf{ek}^*, \hat{\mathsf{dk}}^*) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$ and set $\mathsf{rvk}^* \leftarrow (\mathsf{vk}^*, \mathsf{ek}^*)$, where vk^* is the verification key provided by the EUF-CMA experiment. Moreover, store $\hat{\mathsf{dk}}_{i^*} = \hat{\mathsf{dk}}^*$.
- If $\mathcal{A}$ asks to corrupt rvk^*, abort.
- If $\mathcal{A}$ sends signature query $(m, \mathsf{rvk}^*, \mathsf{R})$, send m to the signing oracle of the EUF-CMA game to obtain a signature σ. Compute the signature ς by
  - Let ind^* be the index of rvk^* in R.
  - Computing $(\mathsf{hk}_1, \mathsf{shk}_1) \leftarrow \mathsf{SPB.Gen}(1^\lambda, |\mathsf{R}|, \mathsf{ind}^*)$
  - Computing $(\mathsf{hk}_2, \mathsf{shk}_2) \leftarrow \mathsf{SPB.Gen}(1^\lambda, |\mathsf{R}|, \mathsf{ind}^*)$
  - Computing $\zeta_1 \leftarrow \mathsf{SPB.Hash}(\mathsf{hk}_1, R)$
  - Computing $\zeta_2 \leftarrow \mathsf{SPB.Hash}(\mathsf{hk}_2, R)$
  - Computing $\tau \leftarrow \mathsf{SPB.Open}(\mathsf{hk}_1, \mathsf{shk}_1, R, \mathsf{ind}^*)$
  - Computing $\mathsf{ctx}_1 \leftarrow \mathsf{PKE.Enc}(\mathsf{ek}^*, \sigma; r_{\mathsf{ctx}})$
  - Computing $\mathsf{ctx}_2 \leftarrow_\$ \{0,1\}^\lambda$
  - Computing

    $$\pi \leftarrow \mathsf{NIP.Prove}((m, \mathsf{ctx}_1, \mathsf{ctx}_2, , \mathsf{hk}_1, \mathsf{hk}_2, \zeta_1, \zeta_2), (\mathsf{rvk}^*, \mathsf{ind}^*, \tau, \sigma, r_{\mathsf{ctx}}))$$

  - Output $\varsigma \leftarrow (\mathsf{ctx}_1, \mathsf{ctx}_2, \mathsf{hk}_1, \mathsf{hk}_2, \pi)$
- Once $\mathcal{A}$ outputs a forgery ς^* for (m^*, R^*), check if it is valid, that is in the query phase $\mathcal{A}$ has not requested a signature of m^* for any key in R^*, none of the keys in R^* has been corrupted and it holds that $\mathsf{RS.Verify}(\mathsf{R}, m^*, \varsigma^*) = \mathsf{true}$. If the forgery is valid proceed.
- Parse ς^* as $\varsigma^* = (\mathsf{ctx}_1^*, \mathsf{ctx}_2^*, \mathsf{hk}_1^*, \mathsf{hk}_2^*, \pi^*)$.
- Let $|\mathsf{R}^*| = \ell$ and let $i_1, \ldots, i_\ell$ be the indices of the keys in R^*, i.e. $\mathsf{R} = (\mathsf{rvk}_{i_1}, \ldots, \mathsf{rvk}_{i_\ell})$.
- For $j = 1, \ldots, \ell$:
  - Compute $\check{\sigma}_1 \leftarrow \mathsf{PKE.Dec}(\hat{\mathsf{dk}}_{i_j}, \mathsf{ctx}_1^*)$ and $\check{\sigma}_2 \leftarrow \mathsf{PKE.Dec}(\hat{\mathsf{dk}}_{i_j}, \mathsf{ctx}_2^*)$.
  - If $\mathsf{DS.Verify}(\mathsf{vk}^*, m^*, \check{\sigma}_1) = \mathsf{true}$ stop and output $\check{\sigma}_1$.
  - If $\mathsf{DS.Verify}(\mathsf{vk}^*, m^*, \check{\sigma}_2) = \mathsf{true}$ stop and output $\check{\sigma}_2$.

First note that the key-pair $(\mathsf{ek}_{i^*}, \hat{\mathsf{dk}}_{i^*})$ is correct for all messages. Notice further that, unless $\mathcal{A}$ asks to corrupt rvk^*, $\mathcal{H}_1$ and the simulation of $\mathcal{R}_2$ are identically distributed from the view of $\mathcal{A}$. Observe that with probability at least $1/q$ the adversary $\mathcal{A}$ does not trigger an abort. Thus, conditioned that no abort happened, from the view of $\mathcal{A}$ the index i^* is distributed uniformly random. Assume now that $\mathcal{A}$ outputs a valid forgery ς^* for (m^*, R^*) with $\mathsf{R}^* = (\mathsf{rvk}_{i_1}, \dots, \mathsf{rvk}_{i_\ell})$. By the perfect soundness of NIP, it holds that either $(m^*, \mathsf{ctx}_1^*, \mathsf{hk}_1^*, \zeta_1^*) \in \mathcal{L}'$ or $(m, \mathsf{ctx}_2^*, \mathsf{hk}_2^*, \zeta_2^*) \in \mathcal{L}'$. Assume w.l.o.g. that $(m, \mathsf{ctx}_1^*, \mathsf{hk}_1^*, \zeta_1^*) \in \mathcal{L}'$. That is, there exist $(\mathsf{rvk}^\dagger, \mathsf{ind}^\dagger, \tau^\dagger, \sigma^\dagger, r_{\mathsf{ctx}})$ with $\mathsf{rvk}^\dagger = (\mathsf{vk}^\dagger, \mathsf{ek}^\dagger)$ such that

$$\begin{aligned} &\mathsf{SPB.Verify}(\mathsf{hk}_1^*, \zeta_1^*, \mathsf{ind}^\dagger, \check{\mathsf{rvk}}, \check{\tau}) = \mathsf{true} \\ \text{and } &\mathsf{PKE.Enc}(\mathsf{ek}^\dagger, \sigma^\dagger; r_{\mathsf{ctx}}) = \mathsf{ctx}_1^* \\ \text{and } &\mathsf{DS.Verify}(\mathsf{vk}^\dagger, m^*, \sigma^\dagger) = \mathsf{true} \end{aligned}$$

As $\mathsf{SPB.Verify}(\mathsf{hk}_1^*, \zeta_1^*, \mathsf{ind}^\dagger, \check{\mathsf{rvk}}, \check{\tau}) = \mathsf{true}$ and $\zeta_1^* = \mathsf{SPB.Hash}(\mathsf{hk}_1^*, \mathsf{R})$ it holds by the somewhere perfectly binding property of SPB that $\mathsf{rvk}^\dagger = \mathsf{rvk}_{i_{\mathsf{ind}^\dagger}}$, i.e. $\mathsf{vk}^\dagger = \mathsf{vk}^\dagger_{i_{\mathsf{ind}^\dagger}}$ and $\mathsf{ek}^\dagger = \mathsf{ek}_{i_{\mathsf{ind}^\dagger}}$. Moreover, by the above it also holds that $\mathsf{ctx}_1^* = \mathsf{PKE.Enc}(\mathsf{ek}_{i_{\mathsf{ind}^\dagger}}, \sigma^\dagger; r_{\mathsf{ctx}})$ and $\mathsf{DS.Verify}(\mathsf{vk}_{i_{\mathsf{ind}^\dagger}}, m^*, \sigma^\dagger) = \mathsf{true}$.

Now observe that, as i^* is uniformly random from the view of $\mathcal{A}$, it holds that $i_{\mathsf{ind}^\dagger} = i^*$ with probability at least $1/q$. Assume therefore that $i_{\mathsf{ind}^\dagger} = i^*$. As $(\mathsf{ek}_{i^*}, \hat{\mathsf{dk}}_{i^*})$ are correct for all messages, it holds that $\check{\sigma}_1 = \mathsf{PKE.Dec}(\hat{\mathsf{dk}}_{i^*}, \mathsf{ctx}_1^*) = \sigma^\dagger$. Therefore, it holds that $\mathsf{DS.Verify}(\mathsf{vk}_{i_{\mathsf{ind}^\dagger}}, m^*, \check{\sigma}_1) = \mathsf{true}$ for the signature $\check{\sigma}_1$ decrypted by $\mathcal{R}_2^{\mathcal{A}}$, i.e. $\check{\sigma}_1$ is a valid signature of m^* under vk^*. We conclude that $\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{R}_2^{\mathcal{A}}}(\lambda) \geq \frac{1}{q}|\mathsf{Adv}^{\mathcal{H}_1}_{\mathcal{A}}(\lambda) - \nu|$.

All together, as $\mathsf{Adv}^{\mathcal{H}_1}_{\mathcal{A}}(\lambda) \geq |\mathsf{Adv}^{\mathcal{H}_0}_{\mathcal{A}}(\lambda) - q \cdot \mathsf{Adv}^{\mathsf{IND\text{-}PK}}_{\mathcal{R}_1^{\mathcal{A}}}(\lambda)|$ and $\mathsf{Adv}^{\mathcal{H}_0}_{\mathcal{A}}(\lambda) = \mathsf{Adv}^{\mathsf{RUF\text{-}IC}}_{\mathcal{A}}(\lambda)$, we can conclude that

$$\mathsf{Adv}^{\mathsf{RUF\text{-}IC}}_{\mathcal{A}}(\lambda) \leq q \cdot \mathsf{Adv}^{\mathsf{IND\text{-}PK}}_{\mathcal{R}_1^{\mathcal{A}}}(\lambda) + q \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{R}_2^{\mathcal{A}}}(\lambda) + \nu.$$

This concludes the proof. □

On Tightness. Using a public key encryption scheme with tight multi-user security, we can improve the bound on the advantage above to

$$\mathsf{Adv}^{\mathsf{RUF\text{-}IC}}_{\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{IND\text{-}PK}}_{\mathcal{R}_1^{\mathcal{A}}}(\lambda) + q \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{R}_2^{\mathcal{A}}}(\lambda) + \nu.$$

However, getting rid of the q factor for $q \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{R}_2^{\mathcal{A}}}(\lambda)$ seems beyond the scope of current techniques.

5.3.2 Proof of Anonymity

We will now turn to establishing anonymity of RS.

Theorem 5.2: Anonymity of Construction 5.2

Construction 5.2 is anonymous, if SPB is index-hiding, PKE has pseudorandom ciphertexts and NIP is computationally witness-indistinguishable.

Our strategy is to first move the index of hk_2 from i_0 to i_1 and argue indistinguishability via the index-hiding property of SPB. Next we switch ctx_2 to an encryption of a signature σ' of m for the verification key rvk_{i_1}. This modification will not be detected due to the pseudorandom ciphertexts property of the PKE. Now, we can switch the NIWI witness to a witness for $(m, \mathsf{ctx}_2, \mathsf{hk}_2, \zeta_2) \in \mathcal{L}'$. Next, we perform the first two changes above for hk_1 and ctx_1, switch the witness back to the witness for $(m, \mathsf{ctx}_1, \mathsf{hk}_1, \zeta_1) \in \mathcal{L}'$, and finally replace ctx_2 with a random string. The signature in the last experiment is now a real signature of m under rvk_{i_1}.

Proof. Let $\mathcal{A}$ be a PPT-adversary against the anonymity of RS. Assume that $\mathcal{A}$ makes at most $q = \mathsf{poly}(\lambda)$ key queries. Let in the following ind_0 be the index of rvk_{i_0} in R and ind_1 be the index of rvk_{i_1} ind R, where $(i_0, i_1, m^*, \mathsf{R})$ is the challenge query of $\mathcal{A}$. Consider the following hybrids:

$\mathcal{H}_0$: This is the real experiment with challenge-bit $b^* = 0$.

$\mathcal{H}_1$: Same as $\mathcal{H}_0$, except that in ς^* we compute hk_2^* by $(\mathsf{hk}_2^*, \mathsf{shk}_2^*) \leftarrow \mathsf{SPB.Gen}(1^\lambda, |\mathsf{R}|, \mathsf{ind}_1)$ instead of computing $(\mathsf{hk}_2^*, \mathsf{shk}_2^*) \leftarrow \mathsf{SPB.Gen}(1^\lambda, |\mathsf{R}|, \mathsf{ind}_0)$. Moreover, also compute $\tau' \leftarrow \tau \leftarrow \mathsf{SPB.Open}(\mathsf{hk}_2, \mathsf{shk}_2, \mathsf{R}, \mathsf{ind}_1)$.

$\mathcal{H}_2$: Same as $\mathcal{H}_1$, except that we compute ctx_2^* by

- $\sigma' \leftarrow \mathsf{DS.Sign}(\mathsf{sk}_{i_1}, m^*)$
- $\mathsf{ctx}_2^* \leftarrow \mathsf{PKE.Enc}(\mathsf{ek}_{i_1}, \sigma'; r_{\mathsf{ctx}_2})$

instead of $\mathsf{ctx}_2^* \leftarrow_\$ \{0,1\}^\lambda$.

$\mathcal{H}_3$: The same as $\mathcal{H}_2$, except that we use the witness $\mathsf{w}' \leftarrow (\mathsf{rvk}_{i_1}, \mathsf{ind}_1, \tau', \sigma', r_{\mathsf{ctx}_2})$ instead of $\mathsf{w} \leftarrow (\mathsf{rvk}_{i_0}, \mathsf{ind}_0, \tau, \sigma, r_{\mathsf{ctx}_1})$ to compute π, i.e. we compute $\pi \leftarrow \mathsf{NIP.Prove}(x, \mathsf{w}')$.

$\mathcal{H}_4$: The same as $\mathcal{H}_3$, except that we compute ctx_1^* by $\mathsf{ctx}_1^* \leftarrow_\$ \{0,1\}^\lambda$.

$\mathcal{H}_5$: The same as $\mathcal{H}_4$, except that we compute hk_1^* by $(\mathsf{hk}_1^*, \mathsf{shk}_1^*) \leftarrow \mathsf{SPB.Gen}(1^\lambda, |\mathsf{R}|, \mathsf{ind}_1)$ instead of $(\mathsf{hk}_1^*, \mathsf{shk}_1^*) \leftarrow \mathsf{SPB.Gen}(1^\lambda, |\mathsf{R}|, \mathsf{ind}_0)$. Moreover, also compute τ by $\tau \leftarrow \mathsf{SPB.Open}(\mathsf{hk}_1, \mathsf{shk}_1, \mathsf{R}, \mathsf{ind}_1)$.

$\mathcal{H}_6$: The same as $\mathcal{H}_5$, except that we compute ctx_1^* by

- $\sigma \leftarrow \mathsf{DS.Sign}(\mathsf{sk}_{i_1}, m^*)$
- $\mathsf{ctx}_1^* \leftarrow \mathsf{PKE.Enc}(\mathsf{ek}_{i_1}, \sigma; r_{\mathsf{ctx}_1})$

instead of $\mathsf{ctx}_1^* \leftarrow_{\$} \{0,1\}^\lambda$.

$\mathcal{H}_7$: The same as $\mathcal{H}_6$, except that we use the witness $\mathsf{w}'' \leftarrow (\mathsf{rvk}_{i_1}, \mathsf{ind}_1, \tau, \sigma, r_{\mathsf{ctx}_1})$ instead of $\mathsf{w}' \leftarrow (\mathsf{rvk}_{i_1}, \mathsf{ind}_1, \tau', \sigma', r_{\mathsf{ctx}_2})$ to compute π, i.e. we compute $\pi \leftarrow \mathsf{NIP.Prove}(x, \mathsf{w}'')$.

$\mathcal{H}_8$: The same as $\mathcal{H}_7$ except that we compute ctx_2^* by $\mathsf{ctx}_2^* \leftarrow_{\$} \{0,1\}^\lambda$. This is identical to the real experiment with $b^* = 1$.

We will show indistinguishability of the hybrids via a sequence of claims.

Claim 5.3: $\mathcal{H}_0 \approx_c \mathcal{H}_1$

$\mathcal{H}_0$ and $\mathcal{H}_1$ are computationally indistinguishable, given that SPB is index-hiding. More specifically, there exists a reduction $\mathcal{R}_1$ such that

$$\mathsf{Adv}_{\mathcal{R}_1^{\mathcal{A}}}^{\mathsf{Index\text{-}Hiding}}(\lambda) = |\Pr[\mathcal{H}_1(\mathcal{A}) = \mathsf{true}] - \Pr[\mathcal{H}_0(\mathcal{A}) = \mathsf{true}]|.$$

We will provide an informal description of $\mathcal{R}_1$. $\mathcal{R}_1$ simulates $\mathcal{H}_0$ faithfully, until $\mathcal{A}$ announces the challenge query $(i_0, i_1, m^*, \mathsf{R})$. $\mathcal{R}_1$ now provides $(i_0, i_1, |\mathsf{R}|)$ to the index-hiding experiment and receives a hashing key hk^*. $\mathcal{R}_1$ continues the simulation of $\mathcal{H}_0$ faithfully, except that in the challenge ciphertext it sets $\mathsf{hk}_2 \leftarrow \mathsf{hk}^*$. In the end, $\mathcal{R}_1$ outputs whatever the simulated $\mathcal{H}_0$ outputs.

Clearly, if the challenge bit of the index-hiding experiment is 0 then $\mathcal{R}_1^{\mathcal{A}}$ simulates $\mathcal{H}_0$ perfectly. On the other hand, if the challenge bit of the index-hiding experiment is 1 then $\mathcal{R}_1^{\mathcal{A}}$ simulates $\mathcal{H}_1$ perfectly. The claim follows.

Claim 5.4: $\mathcal{H}_1 \approx_c \mathcal{H}_2$

We claim that $\mathcal{H}_1$ and $\mathcal{H}_2$ are computationally indistinguishable, given that the ciphertexts of PKE are pseudorandom. More specifically, there exists a reduction $\mathcal{R}_2$ against the pseudorandomness of ciphertexts of PKE such that

$$q \cdot \mathsf{Adv}_{\mathcal{R}_2^{\mathcal{A}}}^{\mathsf{IND\text{-}ENC}}(\lambda) \geq |\Pr[\mathcal{H}_2(\mathcal{A}) = \mathsf{true}] - \Pr[\mathcal{H}_1(\mathcal{A}) = \mathsf{true}]|.$$

The reduction $\mathcal{R}_2$ receives as input a public key ek^*. $\mathcal{R}_2$ simulates $\mathcal{H}_1$ faithfully, except for the following. Before the simulation starts, $\mathcal{R}_2$ guesses an index i_1^* and sets $\mathsf{rvk}_{i_1^*} \leftarrow (\mathsf{vk}_{i_1^*}, \mathsf{ek}^*)$, where $\mathsf{vk}_{i_1^*}$ is generated as in $\mathcal{H}_1$ and ek^* is the input of $\mathcal{R}_2$. $\mathcal{R}_2$ continues the simulation of $\mathcal{H}_1$ until $\mathcal{A}$ announces $(i_0, i_1, m^*, \mathsf{R})$. If it holds

$i_1 \neq i_1^*$, $\mathcal{R}_2$ outputs $\perp$. Otherwise, $\mathcal{R}_2$ computes $\sigma' \leftarrow \mathsf{DS.Sign}(\mathsf{sk}_{i_1}, m^*)$ and sends σ' to the experiment and receives a ciphertext ctx^*. In the computation of the challenge signature it sets $\mathsf{ctx}_2^* \leftarrow \mathsf{ctx}^*$. $\mathcal{R}_2$ now continues the simulation and outputs whatever the simulated $\mathcal{H}_1$ outputs.

Clearly, if the challenge bit of the pseudorandom ciphertexts experiment is 0, then from the view of $\mathcal{A}$ the simulation of $\mathcal{R}_2$ is identically distributed to $\mathcal{H}_0$. On the other hand, if the challenge bit of the pseudorandom ciphertexts experiment is 1, then the view of $\mathcal{A}$ is identically distributed as in $\mathcal{H}_2$. Finally, the guess of i_1^* of $\mathcal{R}_2$ is correct with probability at least $\frac{1}{q}$, thus with probability $\frac{1}{q}$ no abort happens and therefore

$$\mathsf{Adv}_{\mathcal{R}_2^{\mathcal{A}}}^{\mathsf{IND\text{-}ENC}}(\lambda) \geq \frac{1}{q} \cdot |\Pr[\mathcal{H}_2(\mathcal{A}) = \mathsf{true}] - \Pr[\mathcal{H}_1(\mathcal{A}) = \mathsf{true}]|.$$

Claim 5.5: $\mathcal{H}_2 \approx_c \mathcal{H}_3$

We claim that $\mathcal{H}_2$ and $\mathcal{H}_3$ are computationally indistinguishable, given that NIP is computationally witness-indistinguishable. More specifically, there exists a reduction $\mathcal{R}_3$ against the witness indistinguishability of NIP such that

$$\mathsf{Adv}_{\mathcal{R}_3}^{\mathsf{WI}}(\lambda) = |\Pr[\mathcal{H}_3(\mathcal{A}) = \mathsf{true}] - \Pr[\mathcal{H}_2(\mathcal{A}) = \mathsf{true}]|.$$

The reduction $\mathcal{R}_3$ simulates $\mathcal{H}_2$ faithfully, until the challenge signature is computed. Instead of computing the proof π itself, $\mathcal{R}_3$ sends the statement $x = (m, \mathsf{ctx}_1, \mathsf{ctx}_2, \mathsf{hk}_1, \mathsf{hk}_2, \zeta_1, \zeta_2)$ and the witnesses $\mathsf{w}_0 \leftarrow (\mathsf{rvk}_{i_0}, \mathsf{ind}_0, \tau, \sigma, r_{\mathsf{ctx}_1})$ and $\mathsf{w}_1 \leftarrow (\mathsf{rvk}_{i_1}, \mathsf{ind}_1, \tau', \sigma', r_{\mathsf{ctx}_2})$ to the witness indistinguishability experiment. The experiment returns a proof π^*, and $\mathcal{R}_3$ uses the proof π^* in the challenge signature. $\mathcal{R}_3$ continues the simulation of $\mathcal{H}_2$ faithfully and outputs whatever the simulated $\mathcal{H}_2$ outputs.

Clearly, if the challenge-bit of the witness indistinguishability experiment is 0, then $\mathcal{R}_3^{\mathcal{A}}$ simulates $\mathcal{H}_2(\mathcal{A})$ perfectly. On the other hand, if the challenge bit is 1 then $\mathcal{R}_3^{\mathcal{A}}$ simulates $\mathcal{H}_3(\mathcal{A})$ perfectly. The claim follows.

Claim 5.6: $\mathcal{H}_3 \approx_c \mathcal{H}_4$

We claim that $\mathcal{H}_3$ and $\mathcal{H}_4$ are computationally indistinguishable, given that the ciphertexts of PKE are pseudorandom. More specifically, there exists a reduction $\mathcal{R}_4$ against the pseudorandomness of ciphertexts of PKE such that

$$q \cdot \mathsf{Adv}_{\mathcal{R}_4^{\mathcal{A}}}^{\mathsf{IND\text{-}PK}}(\lambda) \geq |\Pr[\mathcal{H}_4(\mathcal{A}) = \mathsf{true}] - \Pr[\mathcal{H}_3(\mathcal{A}) = \mathsf{true}]|.$$

The proof of claim follows analogously to the proof of claim 5.4, except that we perform the change on ctx_1.

Claim 5.7: $\mathcal{H}_4 \approx_c \mathcal{H}_5$

$\mathcal{H}_4$ and $\mathcal{H}_5$ are computationally indistinguishable, given that SPB is index-hiding. More specifically, there exists a reduction $\mathcal{R}_5$ such that

$$\mathsf{Adv}^{\mathsf{Index\text{-}Hiding}}_{\mathcal{R}_5^{\mathcal{A}}}(\lambda) = |\Pr[\mathcal{H}_4(\mathcal{A}) = \mathsf{true}] - \Pr[\mathcal{H}_5(\mathcal{A}) = \mathsf{true}]|.$$

The proof follows analogously to the proof of claim 5.3.

Claim 5.8: $\mathcal{H}_5 \approx_c \mathcal{H}_6$

We claim that $\mathcal{H}_6$ and $\mathcal{H}_5$ are computationally indistinguishable, given that the ciphertexts of PKE are pseudorandom. More specifically, there exists a reduction $\mathcal{R}_6$ against the pseudorandomness of ciphertexts of PKE such that

$$q \cdot \mathsf{Adv}^{\mathsf{IND\text{-}PK}}_{\mathcal{R}_6^{\mathcal{A}}}(\lambda) \geq |\Pr[\mathcal{H}_6(\mathcal{A}) = \mathsf{true}] - \Pr[\mathcal{H}_5(\mathcal{A}) = \mathsf{true}]|.$$

The proof follows analogously to the proof of claim 5.4.

Claim 5.9: $\mathcal{H}_6 \approx_c \mathcal{H}_7$

We claim that $\mathcal{H}_7$ and $\mathcal{H}_6$ are computationally indistinguishable, given that NIP is computationally witness-indistinguishable. More specifically, there exists a reduction $\mathcal{R}_7$ against the witness indistinguishability of NIP such that

$$\mathsf{Adv}^{\mathsf{WI}}_{\mathcal{R}_7}(\lambda) = |\Pr[\mathcal{H}_7(\mathcal{A}) = \mathsf{true}] - \Pr[\mathcal{H}_6(\mathcal{A}) = \mathsf{true}]|.$$

The proof follows analogously to the proof of claim 5.5, except that we perform the change on ctx_1.

Claim 5.10: $\mathcal{H}_7 \approx_c \mathcal{H}_8$

We claim that $\mathcal{H}_8$ and $\mathcal{H}_7$ are computationally indistinguishable, given that the ciphertexts of PKE are pseudorandom. More specifically, there exists a reduction $\mathcal{R}_8$ against the pseudorandomness of ciphertexts of PKE such that

$$q \cdot \mathsf{Adv}^{\mathsf{IND\text{-}ENC}}_{\mathcal{R}_8^{\mathcal{A}}}(\lambda) \geq |\Pr[\mathcal{H}_8(\mathcal{A}) = \mathsf{true}] - \Pr[\mathcal{H}_7(\mathcal{A}) = \mathsf{true}]|.$$

The proof follows analogously to the proof of claim 5.4.

□

5.4 Linkable Ring Signatures, Revisited

In this section we introduce our new model for linkable ring signatures and their security.

Linkable ring signatures provide a public linking algorithm that allows anyone to check whether two signatures were created by the same signer. Beyond the e-voting application sketched in the chapter introduction, this is useful in the setting of cryptocurrencies. Here, on a high level, funds are assigned to public signature verification keys, or *addresses* in a public ledger. Transactions of funds work by giving a signature under the signing key that belongs to the source of the funds and including the target address in the signed message. The transaction can then be checked for validity, i.e. it can be checked on the public ledger that sufficient funds are available at the source.

A privacy focused transaction scheme could allow the sender of a transaction to sign in the name of a ring of potential signers, in order to hide the source of the transaction.[3] To prevent double spending of funds, linkable ring signatures can be employed to link any transactions under the same key.

A similar approach is taken by the *Monero* cryptocurrency.

Definition 5.11: Linkable Ring Signatures

A ring signature scheme is called *linkable* if, in addition to the interface presented in definition 2.17 there is an algorithm Link as follows:

$\mathsf{Link}(m_1, m_2, \varsigma_1, \varsigma_2) \to r \in \{\mathsf{true}, \mathsf{false}\}$
Takes as input two messages m_1, m_2 and two signatures ς_1, ς_2. Outputs either true or false.

The correctness property remains unchanged in the presence of a Link algorithm.

Linkability

We begin our definition of the security properties of linkable ring signatures with the core property called linkability. Informally, we may think of it as the requirement that any two or more uses of a signing key can be publicly linked. We model this property by letting an adversary output q verification keys and signatures, where none of the signatures link with each other. In order to break linkability the adversary has to output one additional signature which does not link with any of the former signatures. Note that producing q signatures which do not link is easy. The adversary only has to use

[3] A different mechanism is then required to ensure that transaction senders are actually in possession of the claimed amount they want to move.

the q different signing keys. But producing the one additional signature without an additional signing key, is required to be infeasible.

Definition 5.12: Linkability

For a linkable ring signature scheme LRS consider the linkability experiment Linkability between a challenger and a multi-stage adversary $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_{q+1})$ for any $q = \mathsf{poly}(\lambda)$.

$\mathsf{Linkability}_{\mathcal{A}}^{\mathsf{LRS}}(1^\lambda, q)$

```
$\mathcal{VK} := \emptyset$; $\mathsf{state}_0 := 1^\lambda$
for $i = 1 \ldots q$ do
    $\mathsf{state}_i, (\mathsf{rvk}_i, \varsigma_i, m_i, \mathsf{R}_i) \leftarrow \mathcal{A}_i(\mathsf{state}_{i-1})$
    $\mathcal{VK} := \mathcal{VK} \cup \{\mathsf{rvk}_i\}$
$(\varsigma^*, m^*, \mathsf{R}^*) \leftarrow \mathcal{A}_{q+1}(\mathsf{state}_q)$
if $\mathsf{R}^* \subseteq \mathcal{VK}$
    and $\mathsf{LRS.Verify}(m^*, \varsigma^*, \mathsf{R}^*) = \mathsf{true}$
    and $\forall i \in [q]. (\mathsf{R}_i \subseteq \mathcal{VK}$
                    and $\mathsf{LRS.Verify}(m_i, \varsigma_i, \mathsf{R}_i) = \mathsf{true}$
                    and $\mathsf{LRS.Link}(m_i, m^*, \varsigma_i, \varsigma^*) = \mathsf{false})$
    and $\forall i, j \in [q],$ s.t. $i \neq j$. $\mathsf{LRS.Link}(m_i, m_j, \varsigma_i, \varsigma_j) = \mathsf{false}$
 then return true
else return false
```

We define the advantage of $\mathcal{A}$ in the above experiment as

$$\mathsf{Adv}_{\mathcal{A},\mathsf{LRS},q}^{\mathsf{Linkability}}(\lambda) = \Pr\left[\mathsf{Linkability}_{\mathcal{A}}^{\mathsf{LRS}}(1^\lambda, q) \Rightarrow \mathsf{true}\right].$$

A linkable signature scheme LRS satisfies *linkability* if for all PPT adversaries $\mathcal{A}$, and all $q = \mathsf{poly}(\lambda)$ we have

$$\mathsf{Adv}_{\mathcal{A},\mathsf{LRS},q}^{\mathsf{Linkability}}(\lambda) \leq \mathsf{negl}(\lambda).$$

Anonymity

We now turn to anonymity. Since, in linkable ring signatures, there is a public link function, it is easy to tell whether multiple signatures were produced by the same signer or not. However, it should still be infeasible to tell which exact user from a ring produced the signature. We argue that, in contrast to the state-of-the-art definitions, in our definition anonymity is not lost at the moment an adversary obtains the first signature of a user. In reality, even when an adversary obtains multiple signature from the same member, identity of the signer should still be unknown, i.e. it should be infeasible to associate the signatures with a verification key. We model this by letting the adversary choose two users, which need to be always in the same rings, and imposing a permutation on the secret keys. If an adversary was able to associate a signature of one of these users with its verification key, then the adversary would also be able to guess the permutation.

Definition 5.13: Linkable Anonymity

For linkable ring signature scheme LRS consider the linkable anonymity experiment Linkable-Anonymity between a challenger and a two-stage adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ for any $q = \mathsf{poly}(\lambda)$.

$\mathsf{Linkable\text{-}Anonymity}_{\mathcal{A}}^{\mathsf{LRS}}(1^\lambda, q)$

```
$\mathcal{VK} := \emptyset$
for $i = 1 \ \dots \ q$ do
   $(\mathsf{rsk}_i, \mathsf{rvk}_i) \leftarrow \mathsf{LRS.KeyGen}(1^\lambda; r_i)$
   $\mathcal{VK} := \mathcal{VK} \cup \{\mathsf{rvk}_i\}$
$b \leftarrow \{0,1\}$
$(\mathsf{state}, \mathcal{U}, i_0, i_1) \leftarrow \mathcal{A}_0(\mathcal{VK})$
if $\mathcal{U} \not\subset \mathcal{VK}$
   or $\mathsf{rvk}_{i_0}, \mathsf{rvk}_{i_1} \notin \mathcal{VK} \setminus \mathcal{U}$
 then abort
$R_{\mathcal{U}} := \{r_i \mid \mathsf{rvk}_i \in \mathcal{U}\}$
$b' \leftarrow \mathcal{A}_1(R_{\mathcal{U}})$
if $b = b'$ then return true
else return false
```

$\mathcal{A}_1$ may query OSign at any point during its runtime.

$\mathsf{OSign}(m, \mathsf{rvk}_s, \mathsf{R})$

```
if $\mathsf{rvk}_s \notin \mathcal{VK} \setminus \mathcal{U}$
   or $\mathsf{rvk}_s \notin \mathsf{R}$
 then return $\perp$
let $S = \{\mathsf{rvk}_{i_0}, \mathsf{rvk}_{i_1}\}$
if not $(S \cap \mathsf{R} = \emptyset$
          or $S \cap \mathsf{R} = S)$
 then abort Linkable-Anonymity
if $\mathsf{rvk}_s \notin S$ then
   return $\mathsf{LRS.Sign}(\mathsf{rsk}_s, m, \mathsf{R})$
if $\mathsf{rvk}_s = \mathsf{rvk}_{i_0}$ then
   return $\mathsf{LRS.Sign}(\mathsf{rsk}_{i_b}, m, \mathsf{R})$
if $\mathsf{rvk}_s = \mathsf{rvk}_{i_1}$ then
   return $\mathsf{LRS.Sign}(\mathsf{rsk}_{i_{(1-b)}}, m, \mathsf{R})$
```

We define the advantage of $\mathcal{A}$ in the above experiment as

$$\mathsf{Adv}_{\mathcal{A},\mathsf{LRS},q}^{\mathsf{Linkable\text{-}Anonymity}}(\lambda) = \left| \Pr\left[\mathsf{Linkable\text{-}Anonymity}_{\mathcal{A}}^{\mathsf{LRS}}(1^\lambda, q) \Rightarrow \mathsf{true}\right] - \frac{1}{2} \right| .$$

A linkable ring signature scheme LRS is *linkably anonymous* if for all PPT adversaries $\mathcal{A}$, and all $q = \mathsf{poly}(\lambda)$ we have

$$\mathsf{Adv}^{\mathsf{Linkable\text{-}Anonymity}}_{\mathcal{A},\mathsf{LRS},q}(\lambda) \leq \mathsf{negl}(\lambda)\,.$$

Non-Frameability

Finally, we require that a linkable ring signature is non-frameable. This property guarantees that it is infeasible for an adversary to forge a signature which would link with an honest users' signature, even when the adversary saw a number of their signatures in the past.

Definition 5.14: Non-frameability

For a linkable ring signature scheme LRS consider the non-frameability experiment Non-Frameability between a challenger and two-stage adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ for any $q = \mathsf{poly}(\lambda)$.

```
Non-Frameability$^{\mathsf{LRS}}_{\mathcal{A}}(1^\lambda, q)$

$\mathcal{VK} := \emptyset$; $\mathbf{C} := \emptyset$; $\mathbf{Q} := \emptyset$
for $i = 1 \ldots q$ do
    $(\mathsf{rsk}_i, \mathsf{rvk}_i) \leftarrow \mathsf{LRS.KeyGen}(1^\lambda; r_i)$
    $\mathcal{VK} := \mathcal{VK} \cup \{\mathsf{rvk}_i\}$
$(\mathsf{R}^*, m^*, \varsigma^*) \leftarrow \mathcal{A}_0(\mathcal{VK})$
$(\mathsf{R}^\dagger, m^\dagger, \varsigma^\dagger) \leftarrow \mathcal{A}_1(r_1, \ldots r_q)$
if $\mathsf{LRS.Verify}(m^*, \varsigma^*, \mathsf{R}^*) = \mathsf{true}$
    and $\mathsf{LRS.Verify}(m^\dagger, \varsigma^\dagger, \mathsf{R}^\dagger) = \mathsf{true}$
    and $\mathsf{R}^* \subseteq \mathcal{VK}$ and $\mathsf{R}^* \cap \mathbf{C} = \emptyset$
    and $\varsigma^* \notin \mathbf{Q}$
    and $\mathsf{LRS.Link}(m^*, m^\dagger, \varsigma^*, \varsigma^\dagger)$
 then return true
else return false
- - - - - - - - - - - - - - - - - - - -
$\mathcal{A}_0$ may query OSign and OCorrupt at any point during its runtime.
```

```
OSign$(\mathsf{rvk}_i, m, \mathsf{R})$

if $\mathsf{rvk}_i \notin \mathsf{R}$
 then return $\perp$
$\varsigma \leftarrow \mathsf{LRS.Sign}(\mathsf{rsk}_i, m, \mathsf{R})$
$\mathbf{Q} := \mathbf{Q} \cup \{\varsigma\}$
return $\varsigma$
```

```
OCorrupt$(\mathsf{rvk}_i)$

$\mathbf{C} := \mathbf{C} \cup \{\mathsf{rvk}_i\}$
return $r_i$
```

We define the advantage of $\mathcal{A}$ in the above experiment as

$$\mathsf{Adv}^{\mathsf{Non\text{-}Frameability}}_{\mathcal{A},\mathsf{LRS},q}(\lambda) = \Pr\left[\mathsf{Non\text{-}Frameability}^{\mathsf{LRS}}_{\mathcal{A}}(1^\lambda, q) \Rightarrow \mathsf{true}\right].$$

A linkable signature scheme LRS is *non-frameable* if for all PPT adversaries $\mathcal{A}$, and all $q = \mathsf{poly}(\lambda)$ we have

$$\mathsf{Adv}^{\mathsf{Non\text{-}Frameability}}_{\mathcal{A},\mathsf{LRS},q}(\lambda) \leq \mathsf{negl}(\lambda).$$

Remark

Beside the properties defined above, we also require the standard unforgeability property from ring signatures to hold for linkable ring signatures.

5.5 Construction of Linkable Ring Signatures

We will now provide a construction of linkable ring signatures from the following primitives. Let

- $\mathsf{Com} = (\mathsf{Commit}, \mathsf{Decommit})$ be a non-interactive commitment scheme.
- $\mathsf{DS} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme.
- $\mathsf{SPB} = (\mathsf{Gen}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify})$ be a somewhere perfectly binding hash function with private local opening.
- $\mathsf{NIP} = (\mathsf{Prove}, \mathsf{Verify})$ be a NIWI-proof system for the language $\mathcal{L}$ and with witness-relation $\mathcal{R}$ which we will both define shortly.

Before we describe the NIWI-proof system NIP in detail, we will define an algorithm CheckValidity. The algorithm takes as input two commitment triples $\mathsf{rvk} = (\mathsf{com}_j)_{j\in[3]}$ and $\mathsf{rvk}' = (\mathsf{com}'_j)_{j\in[3]}$, two inputs vk and vk$'$ as well as two decommitment triples $\Gamma = (\gamma_j)_{j\in[3]}$, $\Gamma' = (\gamma'_j)_{j\in[3]}$. The algorithm checks that of the commitments $\mathsf{com}_1, \mathsf{com}_2, \mathsf{com}_3, \mathsf{com}'_1, \mathsf{com}'_2, \mathsf{com}'_3$ at least two open to vk and at least two open to vk$'$ and at least 5 open to either vk or vk$'$. The last condition can be rephrased as at most one of the 6 commitments does not verify and all the others open to either vk or vk$'$. As the name suggests, the algorithm verifies if the triples rvk and rvk$'$ jointly commit to the values vk and vk$'$, but we allow some leeway which of the 6 commitments actually commit to which value.

Definition 5.15: Validity Check

For the purposes of this algorithm, we bend our rules of notation by identifying the outputs of Com.Decommit with natural numbers as follows:

$$\mathsf{true} \mapsto 1, \mathsf{false} \mapsto 0.$$

```
CheckValidity(rvk, rvk', vk, vk', Γ, Γ')

parse $\mathsf{rvk} = (\mathsf{com}_j)_{j\in[3]}$ and $\mathsf{rvk}' = (\mathsf{com}'_j)_{j\in[3]}$
parse $\Gamma = (\gamma_j)_{j\in[3]}$ and $\Gamma' = (\gamma'_j)_{j\in[3]}$
$s \leftarrow \sum_{j=1}^{3}(\mathsf{Com.Decommit}(\mathsf{com}_j, \mathsf{vk}, \gamma_j) + \mathsf{Com.Decommit}(\mathsf{com}'_j, \mathsf{vk}, \gamma'_j))$
$s' \leftarrow \sum_{j=1}^{3}(\mathsf{Com.Decommit}(\mathsf{com}_j, \mathsf{vk}', \gamma_j) + \mathsf{Com.Decommit}(\mathsf{com}'_j, \mathsf{vk}', \gamma'_j))$
if $s \geq 2$ and $s' \geq 2$ and $s + s' \geq 5$
 then return true
else return false
```

We remark that the expression

$$\mathsf{CheckValidity}(\mathsf{rvk}, \mathsf{rvk}', \mathsf{vk}, \mathsf{vk}', \Gamma, \Gamma') = \mathsf{true}$$

can be unrolled into a short (constant size) sequence of conjunctions and disjunctions over expressions of the form

$$\begin{aligned}\mathsf{Com.Decommit}(\mathsf{com}_j, \mathsf{vk}, \gamma_j) &= \mathsf{true}\\ \mathsf{Com.Decommit}(\mathsf{com}'_j, \mathsf{vk}, \gamma'_j) &= \mathsf{true},\\ \mathsf{Com.Decommit}(\mathsf{com}_j, \mathsf{vk}', \gamma_j) &= \mathsf{true},\\ \text{and } \mathsf{Com.Decommit}(\mathsf{com}'_j, \mathsf{vk}', \gamma'_j) &= \mathsf{true}\end{aligned}$$

for $j \in \{1, 2, 3\}$.[4]

Continuing to our description of NIP, for

$$\begin{aligned}\mathsf{x} &= (\mathsf{vk}, (\mathsf{hk}^{(i)}, \zeta^{(i)})_{i\in[3]})\\ \mathsf{w} &= ((\mathsf{ind}^{(i)}, \mathsf{rvk}^{(i)}, \tau^{(i)}, \mathbf{\Gamma}^{(i)})_{i\in[3]}, \mathsf{vk}')\end{aligned}$$

[4]The expression can be unrolled into a disjunction of $6 \cdot \left(\binom{5}{2} + \binom{5}{3}\right) = 480$ clauses, where each clause is a conjunction of 5 Com.Decommit statements

where

$$\begin{aligned} \mathsf{rvk}^{(i)} &= (\mathsf{com}_1^{(i)}, \mathsf{com}_2^{(i)}, \mathsf{com}_3^{(i)}) \text{ and} \\ \Gamma^{(i)} &= (\gamma_1^{(i)}, \gamma_2^{(i)}, \gamma_3^{(i)}) \end{aligned}$$

for $i \in \{1, 2, 3\}$, let

$$\begin{aligned} \mathcal{R}(\mathsf{x}, \mathsf{w}) \Leftrightarrow\ & \mathsf{SPB.Verify}(\mathsf{hk}^{(1)}, \zeta^{(1)}, \mathsf{ind}^{(1)}, \mathsf{rvk}^{(1)}, \tau^{(1)}) = \mathsf{true} \\ & \text{and } \forall j \in [3] : \mathsf{Com.Decommit}(\mathsf{com}_j^{(1)}, \mathsf{vk}, \gamma_j^{(1)}) = \mathsf{true} \\ & \text{or} \\ & \mathsf{ind}^{(2)} \neq \mathsf{ind}^{(3)} \\ & \text{and } \forall i \in \{2, 3\} : \mathsf{SPB.Verify}(\mathsf{hk}^{(i)}, \zeta^{(i)}, \mathsf{ind}^{(i)}, \mathsf{rvk}^{(i)}, \tau^{(i)}) = \mathsf{true} \\ & \text{and } \mathsf{CheckValidity}(\mathsf{rvk}^{(2)}, \mathsf{rvk}^{(3)}, \mathsf{vk}, \mathsf{vk}', \Gamma^{(2)}, \Gamma^{(3)}) = \mathsf{true}. \end{aligned}$$

And let $\mathcal{L}$ be the language accepted by $\mathcal{R}$.

Construction 5.3: Linkable Ring Signatures

```
LRS.KeyGen($1^\lambda$)
$(\mathsf{vk}_{\mathsf{DS}}, \mathsf{sk}_{\mathsf{DS}}) \leftarrow \mathsf{DS.KeyGen}(1^\lambda)$
for $i = 1, 2, 3$ do
    $(\mathsf{com}_i, \gamma_i) \leftarrow \mathsf{Com.Commit}(\mathsf{vk}_{\mathsf{DS}})$
$\mathsf{rvk} \leftarrow (\mathsf{com}_i)_{i \in [3]}$
$\Gamma \leftarrow (\gamma_i)_{i \in [3]}$
$\mathsf{rsk} \leftarrow (\mathsf{sk}_{\mathsf{DS}}, \mathsf{rvk}, \mathsf{vk}_{\mathsf{DS}}, \Gamma)$
return $(\mathsf{rvk}, \mathsf{rsk})$
```

```
LRS.Link($m_1, m_2, \varsigma_1, \varsigma_2$)
parse
    $\varsigma_1 = (\mathsf{vk}_{\mathsf{DS},1}, (\mathsf{hk}_1^{(i)})_{i \in [3]}, \pi_1, \sigma_1)$
    $\varsigma_2 = (\mathsf{vk}_{\mathsf{DS},2}, (\mathsf{hk}_2^{(i)})_{i \in [3]}, \pi_2, \sigma_2)$
if $\mathsf{vk}_{\mathsf{DS},1} = \mathsf{vk}_{\mathsf{DS},2}$
 then return true
else return false
```

```
LRS.Sign(rsk, $m$, R)

parse
    rsk = $(\mathsf{sk}_{\mathsf{DS}}, \mathsf{rvk}, \mathsf{vk}_{\mathsf{DS}}, \Gamma)$
    rvk = $(\mathsf{com}_j)_{j\in[3]}$
    R = $(\mathsf{rvk}_1, \ldots, \mathsf{rvk}_\ell)$
let ind $\in [\ell]$ s. t. $\mathsf{rvk}_{\mathsf{ind}} = \mathsf{rvk}$
for $i = 1, 2, 3$ do
    $(\mathsf{hk}^{(i)}, \mathsf{shk}^{(i)}) \leftarrow \mathsf{SPB.Gen}(1^\lambda, \ell, \mathsf{ind})$
    $\zeta^{(i)} \leftarrow \mathsf{SPB.Hash}(\mathsf{hk}^{(i)}, \mathsf{R})$
$\tau^{(1)} \leftarrow \mathsf{SPB.Open}(\mathsf{hk}^{(1)}, \mathsf{shk}^{(1)}, \mathsf{R}, \mathsf{ind})$
$\mathsf{x} \leftarrow (\mathsf{sk}_{\mathsf{DS}}, (\mathsf{hk}^{(i)}, \zeta^{(i)})_{i\in[3]})$
$\mathsf{w} \leftarrow ((\mathsf{ind}, \mathsf{rvk}, \tau^{(1)}, \Gamma), \emptyset, \emptyset, \emptyset)$
$\pi \leftarrow \mathsf{NIP.Prove}(\mathsf{x}, \mathsf{w})$
$M := m \| (\mathsf{hk}^{(i)}, \zeta^{(i)})_{i\in[3]} \| \pi$
$\sigma_{\mathsf{DS}} \leftarrow \mathsf{DS.Sign}(\mathsf{sk}_{\mathsf{DS}}, M)$
$\varsigma := (\mathsf{sk}_{\mathsf{DS}}, (\mathsf{hk}^{(i)})_{i\in[3]}, \pi, \sigma_{\mathsf{DS}})$
return $\varsigma$
```

```
LRS.Verify($m, \varsigma$, R)

parse $\varsigma = (\mathsf{sk}_{\mathsf{DS}}, (\mathsf{hk}^{(i)})_{i\in[3]}, \pi, \sigma_{\mathsf{DS}})$
for $i \in [3]$ do
    $\tilde{\zeta}^{(i)} \leftarrow \mathsf{SPB.Hash}(\mathsf{hk}^{(i)}, \mathsf{R})$
$\mathsf{x} \leftarrow (\mathsf{sk}_{\mathsf{DS}}, (\mathsf{hk}^{(i)}, \tilde{\zeta}^{(i)})_{i\in[3]})$
$M := m \| (\mathsf{hk}^{(i)}, \tilde{\zeta}^{(i)})_{i\in[3]} \| \pi$
if $\mathsf{NIP.Verify}(x, \pi) = \mathsf{false}$
 then return false
if $\mathsf{DS.Verify}(\mathsf{sk}_{\mathsf{DS}}, M, \sigma) = \mathsf{false}$
 then return false
return true
```

Correctness. Again, we will first show correctness of our scheme. Assume that rvk $= (\mathsf{com}_1, \mathsf{com}_2, \mathsf{com}_3)$ and rsk $= (\mathsf{sk}, \mathsf{rvk}, \mathsf{vk}, \Gamma)$ with $\Gamma = (\gamma_1, \gamma_2, \gamma_3)$ were generated by LRS.KeyGen and

$$\varsigma := (\mathsf{vk}, \mathsf{hk}^{(1)}, \mathsf{hk}^{(2)}, \mathsf{hk}^{(3)}, \pi, \sigma) \leftarrow \mathsf{LRS.Sign}(\mathsf{rsk}, m, \mathsf{R}),$$

where $\mathsf{R} = (\mathsf{rvk}_1, \ldots, \mathsf{rvk}_\ell)$. We will show that it holds that $\mathsf{LRS.Verify}(m, \varsigma, \mathsf{R}) = \mathsf{true}$.

Because SPB.Hash is deterministic, it holds for the hashes $\tilde{\zeta}^{(1)}, \tilde{\zeta}^{(2)}, \tilde{\zeta}^{(3)}$ computed by $\mathsf{LRS.Verify}(\mathsf{R}, m, \varsigma)$ that $\tilde{\zeta}^{(i)} = \zeta^{(i)}$ (for $i = 1, 2, 3$), where the $\zeta^{(i)}$ are the hashes computed by $\mathsf{LRS.Sign}(\mathsf{rsk}, m, \mathsf{R})$. Also, it holds that $\mathsf{rvk} = \mathsf{rvk}^{(1)}$. Now, notice further that by the correctness of SPB it holds that $\mathsf{SPB.Verify}(\mathsf{hk}^{(1)}, \zeta^{(1)}, \mathsf{ind}, \mathsf{rvk}, \tau^{(1)}) = 1$. By the correctness of the commitment scheme, it holds that $\mathsf{Com.Decommit}(\mathsf{com}_j, \mathsf{vk}, \gamma_j) =$ true for $j = 1, 2, 3$. Thus, $\mathsf{w} = ((\mathsf{ind}, \mathsf{rvk}, \tau^{(1)}, \Gamma)), \emptyset, \emptyset, \emptyset)$ is a valid witness for the statement $x = (\mathsf{vk}, (\mathsf{hk}^{(i)}, \zeta^{(i)})_{i\in[3]})$. Consequently, by the correctness of NIP it holds that $\mathsf{NIP.Verify}(x, \pi) = \mathsf{true}$. Finally, by the correctness of DS we get that $\mathsf{DS.Verify}(\mathsf{vk}, (m, (\mathsf{hk}^{(i)}, \zeta^{(i)})_{i\in[3]}, \pi), \sigma) = \mathsf{true}$ and $\mathsf{LRS.Verify}(\mathsf{R}, m, \varsigma)$ outputs true.

Signature Size. For a signature $\varsigma = (\mathsf{vk}, (\mathsf{hk}^{(i)})_{i\in[3]}, \pi, \sigma)$, the size of the signature component σ is $\mathsf{poly}(\lambda)$ and independent of the ring-size ℓ. By the efficiency property of SPB the sizes of the hashing keys $\mathsf{hk}^{(1)}, \mathsf{hk}^{(2)}, \mathsf{hk}^{(3)}$ is bounded by $\log(\ell) \cdot \mathsf{poly}(\lambda)$. Furthermore, for a statement $x = (\mathsf{vk}, (\mathsf{hk}^{(i)}, \tilde{\zeta}^{(i)})_{i\in[3]})$, the size of the verification circuit C_x is dominated SPB.Verify, which by the efficiency property of SPB can be computed by a circuit of size $\log(\ell) \cdot \mathsf{poly}(\lambda)$. All other algorithms can be computed by circuits of size $\mathsf{poly}(\lambda)$ and independent of ℓ. Consequently, it holds that $|C_x| = \log(\ell) \cdot \mathsf{poly}(\lambda)$. By the efficiency property of the NIWI proof, it holds that $|\pi| = |C_x| \cdot \mathsf{poly}(\lambda) = \log(\ell) \cdot \mathsf{poly}(\lambda)$. All together, we can conclude that $|\varsigma| = \log(\ell) \cdot \mathsf{poly}(\lambda)$.

5.5.1 Proof of Unforgeability

We will turn to showing that LRS is unforgeable. Basically, LRS inherits its unforgeability directly from DS.

Theorem 5.3: Unforgeability of Construction 5.3

Construction 5.3 is unforgeable, given that NIP has perfect soundness, SPB is somewhere perfectly binding, Com is perfectly binding and DS is unforgeable.

The main idea of the proof is as follows. As all keys in the ring R chosen by the adversary are well-formed, and further as Com is perfectly binding and NIP is perfectly sound, the verification key vk used by the adversary $\mathcal{A}$ to produce a forgery must be one of the keys committed to in one of the verification keys rvk_i of R. Thus, we can leverage $\mathcal{A}$ to forge a signature under vk.

Proof. Let $\mathcal{A}$ be a PPT-adversary against the unforgeability experiment of LRS and let further $q = \mathsf{poly}(\lambda)$ be an upper bound on the number of key queries made by $\mathcal{A}$. Let Linkable-Unforgeability be the unforgeability experiment of LRS and EUF-CMA be the unforgeability experiment of DS. We will construct a reduction $\mathcal{R}$ such that it holds that $\mathsf{Adv}_{\mathcal{R}^{\mathcal{A}}}^{\mathsf{EUF\text{-}CMA}}(\lambda) \geq \frac{1}{q} \cdot \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Linkable\text{-}Unforgeability}}(\lambda)$. The reduction $\mathcal{R}$ is given as follows.

Reduction $\mathcal{R}^{\mathcal{A}}(\mathsf{vk}^*)$

- Guess an index $k^* \leftarrow_\$ [q]$. On the k-th key query of $\mathcal{A}$, if $k \neq k^*$, answer the query as in Linkable-Unforgeability. For the k^*-th key query, proceed as follows.
  - For $j = 1, 2, 3$ compute $(\mathsf{com}_j^*, \gamma_j^*) \leftarrow \mathsf{Com.Commit}(1^\lambda, \mathsf{vk}^*)$
  - Set $\Gamma^* \leftarrow (\gamma_j^*)_{j\in[3]}$
  - Output $\mathsf{rvk}^* \leftarrow (\mathsf{com}_j^*)_{j\in[3]}$ and $\mathsf{rsk}^* \leftarrow (\emptyset, \mathsf{rvk}^*, \Gamma^*)$
- If $\mathcal{A}$ asks to corrupt rvk^* abort.

- If $\mathcal{A}$ sends a signature query of the form $(m, \mathsf{rvk}^*, \mathsf{R})$, proceed as follows.
  - Parse $\mathsf{R} = (\mathsf{rvk}_1, \ldots, \mathsf{rvk}_\ell)$
  - Retrieve $\mathsf{rsk}^* = (\emptyset, \mathsf{rvk}^*, \mathsf{vk}^*, \Gamma^*)$
  - Parse $\mathsf{rvk}^* = (\mathsf{com}_j^*)_{j \in [3]}$
  - Find an index $\mathsf{ind}^* \in [\ell]$ such that $\mathsf{rvk}_{\mathsf{ind}^*} = \mathsf{rvk}^*$
  - For $i = 1, 2, 3$ compute $(\mathsf{hk}^{(i)}, \mathsf{shk}^{(i)}) \leftarrow \mathsf{SPB.Gen}(1^\lambda, |\mathsf{R}|, \mathsf{ind}^*)$ and $\zeta^{(i)} \leftarrow \mathsf{SPB.Hash}(\mathsf{hk}^{(i)}, \mathsf{R})$.
  - Compute $\tau^{(1)} \leftarrow \mathsf{SPB.Open}(\mathsf{hk}^{(1)}, \mathsf{shk}^{(1)}, \mathsf{R}, \mathsf{ind}^*)$
  - Set $x \leftarrow (\mathsf{vk}^*, (\mathsf{hk}^{(i)}, \zeta^{(i)})_{i \in [3]})$
  - Set $\mathsf{w} \leftarrow ((\mathsf{ind}^*, \mathsf{rvk}^*, \tau^{(1)}, \Gamma), \emptyset, \emptyset, \emptyset)$.
  - Compute $\pi \leftarrow \mathsf{NIP.Prove}(x, \mathsf{w})$.
  - Send $(m, (\mathsf{hk}^{(i)}, \zeta^{(i)})_{i \in [3]}, \pi)$ to the signing oracle and receive a signature σ.
  - Output $\varsigma \leftarrow (\mathsf{vk}^*, (\mathsf{hk}^{(i)})_{i \in [3]}, \pi, \sigma)$
- Once $\mathcal{A}$ outputs a forgery ς^* for (m^*, R^*), check if it is valid, that is in the query phase $\mathcal{A}$ has not requested a signature of m^* for any key in R^*, none of the keys in R^* has been corrupted, and it holds that $\mathsf{LRS.Verify}(\mathsf{R}, m^*, \varsigma^*) = \mathsf{true}$. If the forgery is valid proceed, otherwise abort.
- Parse ς^* as $\varsigma^* = (\tilde{\mathsf{vk}}, (\mathsf{hk}^{(i)})_{i \in [3]}, \pi^*, \sigma^*)$.
- If $\tilde{\mathsf{vk}} = \mathsf{vk}^*$, output message $(m^*, (\mathsf{hk}^{(i)}, \zeta^{(i)})_{i \in [3]}, \pi^*)$ and signature σ^*, otherwise abort.

First notice that unless $\mathcal{A}$ asks to corrupt rvk^*, the $\mathsf{Linkable\text{-}Unforgeability}$ experiment and the simulation of $\mathcal{R}$ are identically distributed from the view of $\mathcal{A}$. Therefore, from the view of $\mathcal{A}$ the index of the key rvk^* is distributed uniformly at random. Consequently, with probability at least $1/q$ the adversary $\mathcal{A}$ does not trigger an abort by a corruption query to rvk^*.

Assume now that $\mathcal{A}$ outputs a valid forgery $\varsigma^* = (\tilde{\mathsf{vk}}, (\mathsf{hk}^{(i)})_{i \in [3]}, \pi, \sigma^*)$ for (m^*, R^*) with $\mathsf{R}^* = (\mathsf{rvk}_1, \ldots, \mathsf{rvk}_\ell)$. Let $\zeta^{(i)} = \mathsf{SPB.Hash}(\mathsf{hk}^{(i)}, \mathsf{R}^*)$ for $i = 1, 2, 3$. As $\mathsf{LRS.Verify}(\mathsf{R}, m^*, \varsigma^*) = \mathsf{true}$, it holds that $\mathsf{NIP.Verify}((\tilde{\mathsf{vk}}, (\mathsf{hk}^{(i)}, \zeta^{(i)})_{i \in [3]}), \pi) = \mathsf{true}$ and $\mathsf{DS.Verify}(\tilde{\mathsf{vk}}, (m^*, (\mathsf{hk}^{(i)}, \zeta^{(i)})_{i \in [3]}, \pi), \sigma^*) = \mathsf{true}$. Consequently, it holds by the perfect soundness of NIP that $(\tilde{\mathsf{vk}}, (\mathsf{hk}^{(i)}, \zeta^{(i)})_{i \in [3]}) \in \mathcal{L}$. Thus, there exists a witness $\mathsf{w} = ((\mathsf{ind}^{(i)}, \mathsf{rvk}^{(i)}, \tau^{(i)}, \Gamma^{(i)})_{i \in [3]}, \mathsf{vk}')$ where $\mathsf{rvk}^{(i)} = (\mathsf{com}_1^{(i)}, \mathsf{com}_2^{(i)}, \mathsf{com}_3^{(i)})$ and

$\Gamma^{(i)} = (\gamma_1^{(i)}, \gamma_2^{(i)}, \gamma_3^{(i)})$ for $i = 1, \ldots, 3$ such that it holds that

$$\begin{aligned}
&\mathsf{SPB.Verify}(\mathsf{hk}^{(1)}, \zeta^{(1)}, \mathsf{ind}^{(1)}, \mathsf{rvk}^{(1)}, \tau^{(1)}) = \mathsf{true}\\
&\text{and } \forall j \in [3] : \mathsf{Com.Decommit}(\mathsf{com}_j^{(1)}, \tilde{\mathsf{vk}}, \gamma_j^{(1)}) = \mathsf{true}\\
&\text{or}\\
&\mathsf{ind}^{(2)} \neq \mathsf{ind}^{(3)}\\
&\text{and } \forall i \in \{2,3\} : \mathsf{SPB.Verify}(\mathsf{hk}^{(i)}, \zeta^{(i)}, \mathsf{ind}^{(i)}, \mathsf{rvk}^{(i)}, \tau^{(i)}) = \mathsf{true}\\
&\text{and } \mathsf{CheckValidity}(\mathsf{rvk}^{(2)}, \mathsf{rvk}^{(3)}, \tilde{\mathsf{vk}}, \mathsf{vk}', \Gamma^{(2)}, \Gamma^{(3)}) = \mathsf{true}.
\end{aligned}$$

Now recall that all the keys in R^* are honestly generated. That is, the expression

$$\mathsf{CheckValidity}(\mathsf{rvk}^{(2)}, \mathsf{rvk}^{(3)}, \tilde{\mathsf{vk}}, \mathsf{vk}', \Gamma^{(2)}, \Gamma^{(3)}) = \mathsf{true}$$

implies that either

$$\forall j \in [3] : \mathsf{Com.Decommit}(\mathsf{com}_j^{(2)}, \tilde{\mathsf{vk}}, \gamma_j^{(2)}) = \mathsf{true}$$

or

$$\forall j \in [3] : \mathsf{Com.Decommit}(\mathsf{com}_j^{(3)}, \tilde{\mathsf{vk}}, \gamma_j^{(3)}) = \mathsf{true}.$$

Thus, it must hold for an $i^* \in [3]$ both $\mathsf{SPB.Verify}(\mathsf{hk}^{(i^*)}, \zeta^{(i^*)}, \mathsf{ind}^{(i^*)}, \mathsf{rvk}^{(i^*)}, \tau^{(i^*)}) = \mathsf{true}$ and $\forall j \in [3] : \mathsf{Com.Decommit}(\mathsf{com}_j^{(i^*)}, \tilde{\mathsf{vk}}, \gamma_j^{(i^*)}) = \mathsf{true}$.

Now, by the somewhere perfectly binding property of SPB it holds that $\mathsf{rvk}^{(i^*)}$ is in the ring R^* and by the perfectly binding property of Com it holds that $\tilde{\mathsf{vk}}$ is the key that $\mathsf{rvk}^{(i^*)} = (\mathsf{com}_1^{(i^*)}, \mathsf{com}_2^{(i^*)}, \mathsf{com}_3^{(i^*)})$ commits to.

Finally, observe that the index k^* of the key rvk^* is uniformly random in the view of $\mathcal{A}$, thus it holds with probability at least $1/q$ that $\mathsf{rvk}^{(i^*)} = \mathsf{rvk}^*$ and therefore $\tilde{\mathsf{vk}} = \mathsf{vk}^*$. If this event happens, then $\mathcal{R}^{\mathcal{A}}$ outputs a valid forgery σ^* for vk^*.

We conclude that $\mathsf{Adv}_{\mathcal{R}^{\mathcal{A}}}^{\mathsf{EUF\text{-}CMA}}(\lambda) \geq \frac{1}{q} \cdot \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Linkable\text{-}Unforgeability}}(\lambda)$, which concludes the proof. □

5.5.2 Proof of Linkable Anonymity

We will now turn to establishing linkable anonymity of LRS.

Theorem 5.4: Linkable Anonymity of Construction 5.3

Construction 5.3 is linkably anonymous, given that SPB is index-hiding, Com is computationally hiding and NIP is computationally witness-indistinguishable.

Proof. Let $\mathcal{A}$ be a PPT-adversary against the linkable anonymity of LRS. Assume that $\mathcal{A}$ makes at most $q = \mathsf{poly}(\lambda)$ key queries. Consider the following hybrids:

$\mathcal{H}_0$: This experiment is identical to the real experiment Linkable-Anonymity, except for the following modification. Before interacting with $\mathcal{A}$, the experiment guesses two key indices $i_0^*, i_1^* \in [q]$. Later, when $\mathcal{A}$ announces two indices i_0, i_1 in the challenge phase, the experiment aborts if $(i_0, i_1) \neq (i_0^*, i_1^*)$.

It follows immediately that $\mathsf{Adv}_{\mathcal{A}}^{\mathcal{H}_0}(\lambda) \geq \frac{1}{q^2}\mathsf{Adv}_{\mathcal{A}}^{\mathcal{H}_0'}(\lambda)$. We will now show via a sequence of hybrids that show that $\mathsf{Adv}_{\mathcal{A}}^{\mathcal{H}_0}(\lambda)$ is negligible.

For convenience, let in the following hybrids rvk_0 be the verification key at key-index i_0^* and rvk_1 be the verification key at key-index i_1^*. We will briefly review how $\mathsf{rvk}_0, \mathsf{rvk}_1$ and ς_0, ς_1 are computed in $\mathcal{H}_0$. rvk_β for $\beta = 0, 1$ is computed by

- $(\mathsf{vk}_\beta, \mathsf{sk}_\beta) \leftarrow \mathsf{DS.KeyGen}(1^\lambda)$
- For $i = 1, 2, 3$ compute $(\mathsf{com}_{\beta,j}, \gamma_{\beta,j}) \leftarrow \mathsf{Com.Commit}(1^\lambda, \mathsf{vk}_\beta)$
- $\Gamma_\beta \leftarrow (\gamma_{\beta,j})_{j\in[3]}$
- $\mathsf{rvk}_\beta \leftarrow (\mathsf{com}_{\beta,j})_{j\in[3]}$ and $\mathsf{rsk}_\beta \leftarrow (\mathsf{sk}_\beta, \mathsf{rvk}_\beta, \mathsf{vk}_\beta, \Gamma_\beta)$.

Moreover, let ς_0 be challenge signature of m_0 under R_0 and ς_1 be the challenge signature of m_1 under R_1. To facilitate notation let $\mathsf{ind}_{c,d}$ be the index of rvk_d in ring R_c^* for $c, d \in \{0, 1\}$ (these 4 indices will be used frequently in the proof). We will also briefly review how ς_0 and ς_1 are computed.

The signature ς_β for $\beta = 0, 1$ is computed by

- For $i \in [3]$ compute $(\mathsf{hk}_\beta^{(i)}, \mathsf{shk}_\beta^{(i)}) \leftarrow \mathsf{SPB.Gen}(1^\lambda, |\mathsf{R}_\beta|, \mathsf{ind}_{\beta,\beta})$ and $\zeta_\beta^{(i)} \leftarrow \mathsf{SPB.Hash}(\mathsf{hk}_\beta^{(i)}, \mathsf{R}_\beta)$
- $\tau_\beta^{(1)} \leftarrow \mathsf{SPB.Open}(\mathsf{hk}_\beta^{(1)}, \mathsf{shk}_\beta^{(1)}, \mathsf{R}, \mathsf{ind}_{\beta,\beta})$
- $x_\beta \leftarrow (\mathsf{vk}_\beta, (\mathsf{hk}_\beta^{(i)}, \zeta_\beta^{(i)})_{i\in[3]})$
- $\mathsf{w}_\beta \leftarrow ((\mathsf{ind}_{\beta,\beta}, \mathsf{rvk}_\beta, \tau_\beta^{(1)}, \Gamma_\beta), \emptyset, \emptyset, \emptyset)$.
- $\pi_\beta \leftarrow \mathsf{NIP.Prove}(x_\beta, \mathsf{w}_\beta)$
- $\sigma_\beta \leftarrow \mathsf{DS.Sign}(\mathsf{sk}_\beta, (m_\beta, (\mathsf{hk}_\beta^{(i)}, \zeta_\beta^{(i)})_{i\in[3]}, \pi_\beta))$
- $\varsigma_\beta \leftarrow (\mathsf{vk}_\beta, (\mathsf{hk}_\beta^{(i)})_{i\in[3]}, \pi_\beta, \sigma_\beta)$.

This concludes the review of the structure of $\mathsf{rvk}_0, \mathsf{rvk}_1$ and ς_0, ς_1. Now consider the following hybrids.

$\mathcal{H}_0^0$: This is the experiment $\mathcal{H}_0$ with challenge bit $b = 0$.

$\mathcal{H}_1$: This experiment is identical to $\mathcal{H}_0^0$, except for the following conceptual change. We will compute rvk_0 and rvk_1 simultaneously before the experiment begins.

- $(\mathsf{vk}_0, \mathsf{sk}_0) \leftarrow \mathsf{DS.KeyGen}(1^\lambda)$
- $(\mathsf{vk}_1, \mathsf{sk}_1) \leftarrow \mathsf{DS.KeyGen}(1^\lambda)$
- For $i = 1, 2, 3$ compute $(\mathsf{com}_{0,j}, \gamma_{0,j}) \leftarrow \mathsf{Com.Commit}(1^\lambda, \mathsf{vk}_0)$
- For $i = 1, 2, 3$ compute $(\mathsf{com}_{1,j}, \gamma_{1,j}) \leftarrow \mathsf{Com.Commit}(1^\lambda, \mathsf{vk}_1)$
- $\Gamma_0 \leftarrow (\gamma_{0,j})_{j\in[3]}$
- $\Gamma_0 \leftarrow (\gamma_{0,j})_{j\in[3]}$
- $\mathsf{rvk}_0 \leftarrow (\mathsf{com}_{0,j})_{j\in[3]}$ and $\mathsf{rsk}_0 \leftarrow (\mathsf{sk}_0, \mathsf{rvk}_0, \mathsf{vk}_0, \Gamma_0)$.
- $\mathsf{rvk}_0 \leftarrow (\mathsf{com}_{0,j})_{j\in[3]}$ and $\mathsf{rsk}_0 \leftarrow (\mathsf{sk}_0, \mathsf{rvk}_0, \mathsf{vk}_0, \Gamma_0)$.

That is, in the computation of rvk_0 and rvk_1 we first choose vk_0 and vk_1 before computing anything else. Looking ahead, this will be important later on when we change the roles of vk_0 and vk_1. This change is only conceptual and therefore $\mathcal{H}_0^0$ and $\mathcal{H}_1$ are identically distributed from the view of $\mathcal{A}$.

$\mathcal{H}_2$: Identical to $\mathcal{H}_1$, except that we compute $\mathsf{hk}_0^{(2)}, \mathsf{hk}_0^{(3)}, \mathsf{hk}_1^{(2)}, \mathsf{hk}_1^{(3)}$ as follows:

$$(\mathsf{hk}_0^{(2)}, \mathsf{shk}_0^{(2)}) \leftarrow \mathsf{SPB.Gen}(1^\lambda, |\mathsf{R}_0|, \mathsf{ind}_{0,0})$$
$$(\mathsf{hk}_0^{(3)}, \mathsf{shk}_0^{(3)}) \leftarrow \mathsf{SPB.Gen}(1^\lambda, |\mathsf{R}_0|, \mathsf{ind}_{0,1})$$
$$(\mathsf{hk}_1^{(2)}, \mathsf{shk}_1^{(2)}) \leftarrow \mathsf{SPB.Gen}(1^\lambda, |\mathsf{R}_1|, \mathsf{ind}_{1,0})$$
$$(\mathsf{hk}_1^{(3)}, \mathsf{shk}_1^{(3)}) \leftarrow \mathsf{SPB.Gen}(1^\lambda, |\mathsf{R}_1|, \mathsf{ind}_{1,1}).$$

That is, we move the binding index of $\mathsf{hk}_0^{(3)}$ from $\mathsf{ind}_{0,0}$ to $\mathsf{ind}_{0,1}$ and of $\mathsf{hk}_1^{(2)}$ from $\mathsf{ind}_{1,1}$ to $\mathsf{ind}_{1,0}$. $\mathsf{hk}_0^{(2)}$ and $\mathsf{hk}_1^{(3)}$ are computed as before and only stated here for convenience. The secret keys $\mathsf{shk}_0^{(2)}, \mathsf{shk}_0^{(3)}, \mathsf{shk}_1^{(2)}, \mathsf{shk}_1^{(3)}$ are not needed to compute information which is used in the witnesses w_0 and w_1 in the experiments $\mathcal{H}_1$ and $\mathcal{H}_2$. Therefore, we can argue that $\mathcal{H}_1$ and $\mathcal{H}_2$ are computationally indistinguishable via the index-hiding property of SPB.

$\mathcal{H}_3$: In this hybrid we also compute

$$\tau_0^{(2)} \leftarrow \mathsf{SPB.Open}(\mathsf{hk}_0^{(2)}, \mathsf{shk}_0^{(2)}, \mathsf{rvk}_0, \mathsf{ind}_{0,0})$$
$$\tau_0^{(3)} \leftarrow \mathsf{SPB.Open}(\mathsf{hk}_0^{(3)}, \mathsf{shk}_0^{(3)}, \mathsf{rvk}_0, \mathsf{ind}_{0,1})$$

in the computation of ς_0 and

$$\tau_1^{(2)} \leftarrow \mathsf{SPB.Open}(\mathsf{hk}_1^{(2)}, \mathsf{shk}_1^{(2)}, \mathsf{rvk}_0, \mathsf{ind}_{1,0})$$
$$\tau_1^{(3)} \leftarrow \mathsf{SPB.Open}(\mathsf{hk}_1^{(3)}, \mathsf{shk}_1^{(3)}, \mathsf{rvk}_0, \mathsf{ind}_{1,1})$$

in the computation of ς_1. The values $\tau_0^{(2)}, \tau_0^{(3)}, \tau_1^{(2)}, \tau_1^{(3)}$ are not used further in $\mathcal{H}_3$. Thus, this modification is only conceptual and we get that $\mathcal{H}_2$ and $\mathcal{H}_3$ are identically distributed from the view of $\mathcal{A}$.

$\mathcal{H}_4$: In this hybrid we switch the witnesses w_0 and w_1 used for the computation of ς_0 and ς_1. Specifically, instead of computing

$$\mathsf{w}_0 \leftarrow ((\mathsf{ind}_{0,0}, \mathsf{rvk}_0, \tau_0^{(1)}, \Gamma_0), \emptyset, \emptyset, \emptyset)$$
$$\mathsf{w}_1 \leftarrow ((\mathsf{ind}_{1,1}, \mathsf{rvk}_1, \tau_1^{(1)}, \Gamma_1), \emptyset, \emptyset, \emptyset)$$

we compute

$$\mathsf{w}_0 \leftarrow (\emptyset, (\mathsf{ind}_{0,0}, \mathsf{rvk}_0, \tau_0^{(2)}, \Gamma_0), (\mathsf{ind}_{0,1}, \mathsf{rvk}_1, \tau_0^{(3)}, \Gamma_1), \mathsf{vk}_1)$$
$$\mathsf{w}_1 \leftarrow (\emptyset, (\mathsf{ind}_{1,0}, \mathsf{rvk}_0, \tau_1^{(2)}, \Gamma_0), (\mathsf{ind}_{1,1}, \mathsf{rvk}_1, \tau_1^{(3)}, \Gamma_1), \mathsf{vk}_0).$$

First notice that w_0 is also a witness for the statement $x_0 = (\mathsf{vk}_0, (\mathsf{hk}_0^{(i)}, \zeta_0^{(i)})_{i\in[3]})$ as $\mathsf{CheckValidity}(\mathsf{rvk}_0, \mathsf{rvk}_1, \mathsf{vk}_0, \mathsf{vk}_1, \Gamma_0, \Gamma_1) = \mathsf{true}$ holds. Recall that this predicate holds if at least two out of the six commitments correctly open to vk_0, at most one commitment does not open correctly, and the remaining commitments open to vk_1. Likewise, the witness w_1 is a witness for the statement $x_1 = (\mathsf{vk}_1, (\mathsf{hk}_1^{(i)}, \zeta_1^{(i)})_{i\in[3]})$ as $\mathsf{CheckValidity}(\mathsf{rvk}_0, \mathsf{rvk}_1, \mathsf{vk}_1, \mathsf{vk}_0, \Gamma_0, \Gamma_1) = \mathsf{true}$. We can argue via the computational witness indistinguishability of NIP that $\mathcal{H}_3$ and $\mathcal{H}_4$ are computationally indistinguishable form the view of $\mathcal{A}$.

In the following hybrids $\mathcal{H}_5, \dots, \mathcal{H}_{17}$, we will simultaneously modify $\mathsf{rvk}_0 = (\mathsf{com}_{0,1}, \mathsf{com}_{0,2}, \mathsf{com}_{0,3})$ and $\mathsf{rvk}_1 = (\mathsf{com}_{1,1}, \mathsf{com}_{1,2}, \mathsf{com}_{1,3})$ such that rvk_0 commits to vk_1 and rvk_1 commits to vk_0. In other words, we will switch the roles of rvk_0 and rvk_1. The configuration in each hybrid is given in table table 5.1. Recall that the joint verification algorithm $\mathsf{CheckValidity}$ tolerates one incorrect/missing unveil. In order to use the hiding property of Com, in each hybrid we will *forget* the unveil γ for (at most) one commitment. The commitment for which the unveil is missing is given in a gray box. As a brief explanation, in $\mathcal{H}_4$ all unveils γ are present. In $\mathcal{H}_5$ we forget the unveil $\gamma_{0,1}$ of the commitment $\mathsf{com}_{0,1}$, i.e. we set $\gamma_{0,1} \leftarrow \emptyset$. Indistinguishability between $\mathcal{H}_4$ and $\mathcal{H}_5$ follows by the witness indistinguishability of NIP. In $\mathcal{H}_6$, we change $\mathsf{com}_{0,1}$ into a commitment of vk_1 and can argue indistinguishability of $\mathcal{H}_5$ and $\mathcal{H}_6$ via the hiding property of Com. In $\mathcal{H}_7$ we erase the unveil information $\gamma_{1,1}$ of $\mathsf{com}_{1,1}$ instead of $\gamma_{0,0}$ indistinguishability of $\mathcal{H}_6$ and $\mathcal{H}_7$ follows again by the witness indistinguishability of NIP. The remaining steps follow analogously, observing that in each row of table table 5.1 it holds *both* $\mathsf{CheckValidity}(\mathsf{rvk}_0, \mathsf{rvk}_1, \mathsf{vk}_0, \mathsf{vk}_1, \Gamma_0, \Gamma_1) = \mathsf{true}$ and $\mathsf{CheckValidity}(\mathsf{rvk}_0, \mathsf{rvk}_1, \mathsf{vk}_1, \mathsf{vk}_0, \Gamma_0, \Gamma_1) = \mathsf{true}$.

Hybrid	$\mathsf{com}_{0,1}$	$\mathsf{com}_{0,2}$	$\mathsf{com}_{0,3}$	$\mathsf{com}_{1,1}$	$\mathsf{com}_{1,2}$	$\mathsf{com}_{1,3}$
$\mathcal{H}_4$	vk_0	vk_0	vk_0	vk_1	vk_1	vk_1
$\mathcal{H}_5$	vk_0	vk_0	vk_0	vk_1	vk_1	vk_1
$\mathcal{H}_6$	vk_1	vk_0	vk_0	vk_1	vk_1	vk_1
$\mathcal{H}_7$	vk_1	vk_0	vk_0	vk_1	vk_1	vk_1
$\mathcal{H}_8$	vk_1	vk_0	vk_0	vk_0	vk_1	vk_1
$\mathcal{H}_9$	vk_1	vk_0	vk_0	vk_0	vk_1	vk_1
$\mathcal{H}_{11}$	vk_1	vk_1	vk_0	vk_0	vk_1	vk_1
$\mathcal{H}_{11}$	vk_1	vk_1	vk_0	vk_0	vk_1	vk_1
$\mathcal{H}_{12}$	vk_1	vk_1	vk_0	vk_0	vk_0	vk_1
$\mathcal{H}_{13}$	vk_1	vk_1	vk_0	vk_0	vk_0	vk_1
$\mathcal{H}_{14}$	vk_1	vk_1	vk_1	vk_0	vk_0	vk_1
$\mathcal{H}_{15}$	vk_1	vk_1	vk_1	vk_0	vk_0	vk_1
$\mathcal{H}_{16}$	vk_1	vk_1	vk_1	vk_0	vk_0	vk_0
$\mathcal{H}_{17}$	vk_1	vk_1	vk_1	vk_0	vk_0	vk_0

Table 5.1: The hybrids

$\mathcal{H}_{18}$: Identical to $\mathcal{H}_{17}$, except that we compute $\mathsf{hk}_0^{(1)}$ and $\mathsf{hk}_1^{(1)}$ differently. Instead of computing $\mathsf{hk}_0^{(1)}$ and $\mathsf{hk}_1^{(1)}$ by

$$(\mathsf{hk}_0^{(1)}, \mathsf{shk}_0^{(1)}) \leftarrow \mathsf{SPB.Gen}(1^\lambda, |\mathsf{R}_0|, \mathsf{ind}_{0,0})$$
$$(\mathsf{hk}_1^{(1)}, \mathsf{shk}_1^{(1)}) \leftarrow \mathsf{SPB.Gen}(1^\lambda, |\mathsf{R}_1|, \mathsf{ind}_{1,1})$$

we compute

$$(\mathsf{hk}_0^{(1)}, \mathsf{shk}_0^{(1)}) \leftarrow \mathsf{SPB.Gen}(1^\lambda, |\mathsf{R}_0|, \mathsf{ind}_{0,1})$$
$$(\mathsf{hk}_1^{(1)}, \mathsf{shk}_1^{(1)}) \leftarrow \mathsf{SPB.Gen}(1^\lambda, |\mathsf{R}_0|, \mathsf{ind}_{1,0}).$$

That is, we move the binding index of $\mathsf{hk}_0^{(1)}$ from $\mathsf{ind}_{0,0}$ to $\mathsf{ind}_{0,1}$ and of $\mathsf{hk}_1^{(1)}$ from $\mathsf{ind}_{1,1}$ to $\mathsf{ind}_{1,0}$. The secret keys $\mathsf{shk}_0^{(1)}, \mathsf{shk}_1^{(1)}$ are not needed to compute information which is used in the witnesses w_0 and w_1 in $\mathcal{H}_{17}$ and $\mathcal{H}_{18}$. Therefore, we can argue that $\mathcal{H}_{17}$ and $\mathcal{H}_{18}$ are computationally indistinguishable via the index-hiding property of SPB.

In the remaining hybrids, we essentially mirror the modification in hybrids $\mathcal{H}_1, \ldots, \mathcal{H}_4$.

$\mathcal{H}_{19}$ (mirroring $\mathcal{H}_4$): The same as $\mathcal{H}_{18}$, except that we switch the witnesses w_0 and w_1 used for the computation of ς_0 and ς_1. Specifically, instead of computing

$$\mathsf{w}_0 \leftarrow (\emptyset, (\mathsf{ind}_{0,0}, \mathsf{rvk}_0, \tau_0^{(2)}, \Gamma_0), (\mathsf{ind}_{0,1}, \mathsf{rvk}_1, \tau_0^{(3)}, \Gamma_1), \mathsf{vk}_1)$$
$$\mathsf{w}_1 \leftarrow (\emptyset, (\mathsf{ind}_{1,0}, \mathsf{rvk}_0, \tau_1^{(2)}, \Gamma_0), (\mathsf{ind}_{1,1}, \mathsf{rvk}_1, \tau_1^{(3)}, \Gamma_1), \mathsf{vk}_0)$$

we compute

$$\mathsf{w}_0 \leftarrow ((\mathsf{ind}_{0,1}, \mathsf{rvk}_1, \tau_0^{(1)}, \Gamma_1), \emptyset, \emptyset, \emptyset)$$
$$\mathsf{w}_1 \leftarrow ((\mathsf{ind}_{1,0}, \mathsf{rvk}_0, \tau_1^{(1)}, \Gamma_0), \emptyset, \emptyset, \emptyset).$$

First notice that w_0 is also a witness for the statement $x_0 = (\mathsf{vk}_0, (\mathsf{hk}_0^{(i)}, \zeta_0^{(i)})_{i\in[3]})$ as it holds $\forall j \in [3] : \mathsf{Com.Decommit}(\mathsf{com}_{1,j}^{(1)}, \mathsf{vk}_0, \gamma_{1,j}^{(1)}) = \mathsf{true}$. Likewise, the witness w_1 is a witness for the statement $x_1 = (\mathsf{vk}_1, (\mathsf{hk}_1^{(i)}, \zeta_1^{(i)})_{i\in[3]})$ as it holds $\forall j \in [3] : \mathsf{Com.Decommit}(\mathsf{com}_{0,j}^{(1)}, \mathsf{vk}_1, \gamma_{0,j}^{(1)}) = \mathsf{true}$. We can argue via the computational witness indistinguishability of NIP that $\mathcal{H}_{18}$ and $\mathcal{H}_{19}$ are computationally indistinguishable form the view of $\mathcal{A}$.

$\mathcal{H}_{20}$ (mirroring $\mathcal{H}_3$): In this hybrid we drop the computation of $\tau_0^{(2)}, \tau_0^{(3)}$ in the computation of ς_0 and we also drop the computation of $\tau_1^{(2)}, \tau_1^{(3)}$ in the computation of ς_1. The values $\tau_0^{(2)}, \tau_0^{(3)}, \tau_1^{(2)}, \tau_1^{(3)}$ are not used further in $\mathcal{H}_{19}$. Thus, this modification is only conceptual and we get that $\mathcal{H}_{19}$ and $\mathcal{H}_{20}$ are identically distributed from the view of $\mathcal{A}$.

$\mathcal{H}_{21}$ (mirroring $\mathcal{H}_2$): Identical to $\mathcal{H}_{20}$, except that we compute $\mathsf{hk}_0^{(2)}$, $\mathsf{hk}_0^{(3)}$, $\mathsf{hk}_1^{(2)}$, $\mathsf{hk}_1^{(3)}$ as follows:

$$(\mathsf{hk}_0^{(2)}, \mathsf{shk}_0^{(2)}) \leftarrow \mathsf{SPB.Gen}(1^\lambda, |\mathsf{R}_0|, \mathsf{ind}_{0,1})$$
$$(\mathsf{hk}_0^{(3)}, \mathsf{shk}_0^{(3)}) \leftarrow \mathsf{SPB.Gen}(1^\lambda, |\mathsf{R}_0|, \mathsf{ind}_{0,1})$$
$$(\mathsf{hk}_1^{(2)}, \mathsf{shk}_1^{(2)}) \leftarrow \mathsf{SPB.Gen}(1^\lambda, |\mathsf{R}_0|, \mathsf{ind}_{1,0})$$
$$(\mathsf{hk}_1^{(3)}, \mathsf{shk}_1^{(3)}) \leftarrow \mathsf{SPB.Gen}(1^\lambda, |\mathsf{R}_0|, \mathsf{ind}_{1,0}).$$

That is, we move the binding index of $\mathsf{hk}_0^{(2)}$ from $\mathsf{ind}_{0,0}$ to $\mathsf{ind}_{0,1}$ and of $\mathsf{hk}_1^{(3)}$ from $\mathsf{ind}_{1,1}$ to $\mathsf{ind}_{1,0}$. $\mathsf{hk}_0^{(3)}$ and $\mathsf{hk}_1^{(2)}$ are computed as before and only stated here for convenience. The secret keys $\mathsf{shk}_0^{(2)}, \mathsf{shk}_0^{(3)}, \mathsf{shk}_1^{(2)}, \mathsf{shk}_1^{(3)}$ are not needed to compute information which is used in the witnesses w_0 and w_1 in the experiments $\mathcal{H}_{20}$ and $\mathcal{H}_{21}$. Therefore, we can argue that $\mathcal{H}_{20}$ and $\mathcal{H}_{21}$ are computationally indistinguishable via the index-hiding property of SPB.

$\mathcal{H}_{22}$ (mirroring $\mathcal{H}_1$): Identical to $\mathcal{H}_{21}$ except that we compute rvk_0 and rvk_1 as follows. We compute rvk_β for $\beta = 0, 1$ is computed by

- $(\mathsf{vk}_{1-\beta}, \mathsf{sk}_{1-\beta}) \leftarrow \mathsf{DS.KeyGen}(1^\lambda)$
- For $i = 1, 2, 3$ compute $(\mathsf{com}_{\beta,j}, \gamma_{\beta,j}) \leftarrow \mathsf{Com.Commit}(1^\lambda, \mathsf{vk}_{1-\beta})$
- $\Gamma_\beta \leftarrow (\gamma_{\beta,j})_{j\in[3]}$

- $\mathsf{rvk}_\beta \leftarrow (\mathsf{com}_{\beta,j})_{j\in[3]}$ and $\mathsf{rsk}_\beta \leftarrow (\mathsf{sk}_{1-\beta}, \mathsf{rvk}_\beta, \mathsf{vk}_{1-\beta}, \Gamma_\beta)$.

In $\mathcal{H}_{21}$ we have computed rvk_0 and rvk_1 simultaneously. However, as the computation of rvk_0 and rvk_1 has no shared information, we can compute them independently of one another. Thus, this modification is only conceptual and we get that from the view of $\mathcal{A}$ $\mathcal{H}_{21}$ and $\mathcal{H}_{22}$ are identically distributed.

We now observe that from the view of $\mathcal{A}$, $\mathcal{H}_{22}$ is identically distributed to $\mathcal{H}_0^1$, i.e. $\mathcal{H}_0$ with challenge bit $b = 1$. Consequently, we get that

$$\mathsf{Adv}_{\mathcal{A}}^{\mathcal{H}_0}(\lambda) = |\Pr[\mathcal{H}_0^0(\mathcal{A}) = \mathsf{true} \; - \Pr[\mathcal{H}_0^1(\mathcal{A}) = \mathsf{true}]| < \mathsf{negl}(\lambda)\,,$$

which concludes the proof.

□

5.5.3 Proof of Linkability

Before we provide the proof of linkability of the scheme LRS we need the following lemma about the algorithm CheckValidity (cf. definition 5.15).

We will first introduce some terminology that will facilitate notation.

Definition 5.16: Yielding Elements and Partner Keys

- We say that a key $\mathsf{rvk} = (\mathsf{com}_1, \mathsf{com}_2, \mathsf{com}_3)$ *yields an element* x, if there exist $\gamma_1, \gamma_2, \gamma_3$ such that $\mathsf{Com.Decommit}(\mathsf{com}_j, x, \gamma_j) = \mathsf{true}$ for $i = 1, 2, 3$.
- We say that a pair of keys $(\mathsf{rvk}, \mathsf{rvk}')$ yield x, if there exist y and Γ, Γ' such that $\mathsf{CheckValidity}(\mathsf{rvk}, \mathsf{rvk}', x, y, \Gamma, \Gamma') = \mathsf{true}$.
- Furthermore, we say that a ring $\mathsf{R} = (\mathsf{rvk}_1, \ldots, \mathsf{rvk}_n)$ yields a set $\mathcal{S}$, if for every $x \in \mathcal{S}$ there exists a $\mathsf{rvk}_i \in \mathsf{R}$ such that rvk_i yields x or there exist distinct $\mathsf{rvk}_{i_1}, \mathsf{rvk}_{i_2}$ such that $(\mathsf{rvk}_{i_1}, \mathsf{rvk}_{i_2})$ yield x.
- We say that $\mathsf{rvk}^* \in \mathsf{R}$ has a partner in R, if there exists a $\widetilde{\mathsf{rvk}} \in \mathsf{R}$ such that $(\mathsf{rvk}^*, \widetilde{\mathsf{rvk}})$ yield an element in $\mathcal{S}$.

In abuse of notation, we also say that rvk is of the form $(\mathsf{com}(x), \mathsf{com}(y), \mathsf{com}(z))$, if one of the commitments in rvk commits to x, one to y and one to z, where we allow x, y, z to be the same.

Lemma 5.5: Yield Size

Assume that a ring $\mathsf{R} = (\mathsf{rvk}_1, \ldots, \mathsf{rvk}_n)$ yields a set $\mathcal{S}$. Then it holds that $|\mathcal{S}| \leq |\mathsf{R}|$.

In other words, a ring of size n yields at most n distinct elements.

Proof. We will prove the lemma by induction over the size of R. For the base case of $|\mathcal{V}| = 2$, it follows immediately from the definition of $\mathsf{CheckValidity}$ that R yields at most 2 elements. Assume that the assertion holds for rings of size at most $n-1$. We will now show that it also holds for rings of size n. Let therefore R be any ring of size n. We will distinguish three cases.

Case 1: In this case R contains a rvk^* of the form $(\mathsf{com}(x), \mathsf{com}(x), \mathsf{com}(x))$.

Let $\mathcal{S}'$ be the set yielded by $\mathsf{R}\backslash\{\mathsf{rvk}^*\}$. Assume that $\mathsf{R}\backslash\{\mathsf{rvk}^*\}$ and rvk^* together yield k elements outside of $\mathcal{S}'$. Call the set of these elements $\mathcal{T}$. We will show that there exists a sub-ring $\mathsf{R}' \subseteq \mathsf{R}$ of size $n-k$ which yields $\mathcal{S}'$. As $|\mathsf{R}'| \leq n-1$ we will conclude by the induction hypothesis that $|\mathcal{S}'| \leq |\mathsf{R}'| \leq n-k$, which in turn will imply that $|\mathcal{S}| \leq |\mathcal{S}'| + |\mathcal{T}| \leq n-k+k = n$.

For all $t \in \mathcal{T}$ with $t \neq x$ there must exist a $\mathsf{rvk}_t^* \in \mathsf{R}\backslash\{\mathsf{rvk}^*\}$ such that $(\mathsf{rvk}^*, \mathsf{rvk}_t^*)$ yield t. But this means that rvk_t^* is of the form $(\mathsf{com}(t), \mathsf{com}(t), *)$, and this implies that rvk_t^* does not have a partner in $\mathsf{R}\backslash\{\mathsf{rvk}^*\}$, as otherwise it would hold $t \in \mathcal{S}'$. Moreover, for $t' \neq t$ it we immediately get by the above that $\mathsf{rvk}_{t'}^* \neq \mathsf{rvk}_t^*$. Thus, we can remove at least $k-1$ elements from $\mathsf{R}\backslash\{\mathsf{rvk}^*\}$ (i.e. all rvk_t^* corresponding to the elements in $\mathcal{T}\backslash\{x\}$) and obtain a ring R' of size $n-k$ while guaranteeing that the resulting ring still yields $\mathcal{S}'$. Using the induction hypothesis we conclude that $|\mathcal{S}'| \leq |\mathsf{R}'| \leq n-k$ and therefore $\mathcal{S} \leq n$.

Case 2: In this case R does not contain any rvk of the form $(\mathsf{com}(x), \mathsf{com}(x), \mathsf{com}(x))$, but it does contain a rvk^* of the form $(\mathsf{com}(x), \mathsf{com}(x), \mathsf{com}(y))$. Let $\mathcal{S}'$ be the set yielded by $\mathsf{R}\backslash\{\mathsf{rvk}^*\}$. Again assume that $\mathsf{R}\backslash\{\mathsf{rvk}^*\}$ and rvk^* yield k elements outside of $\mathcal{S}'$ and call this set $\mathcal{T}$. As in case 1 we will show that there exists a sub-ring $\mathsf{R}' \subseteq \mathsf{R}$ of size $n-k$ which yields $\mathcal{S}'$. As $|\mathsf{R}'| \leq n-1$ we will conclude by the induction hypothesis that $|\mathcal{S}'| \leq |\mathsf{R}'| \leq n-k$, which in turn will imply that $|\mathcal{S}| \leq |\mathcal{S}'| + |\mathcal{T}| \leq n-k+k = n$.

For every $t \in \mathcal{T}$ different from x and y there exists a $\mathsf{rvk}_t^* \in \mathsf{R}\backslash\{\mathsf{rvk}^*\}$ such that rvk_t^* and rvk^* yield t. Such a rvk_t^* must be of the form $\mathsf{rvk}_t^* = (\mathsf{com}(t), \mathsf{com}(t), \mathsf{com}(x))$. Consequently, rvk_t^* cannot have a partner in $\mathsf{R}\backslash\{\mathsf{rvk}^*\}$, as this would imply that $t \in \mathcal{S}'$. For two distinct $t' \neq t \in \mathcal{T}$, it must hold that $\mathsf{rvk}_t^* \neq \mathsf{rvk}_{t'}^*$ by the above.

If $y \notin \mathcal{T}$ we are done, and will be able to argue as in case 1. If $y \in \mathcal{T}$, then there exists a $\mathsf{rvk}_y^* \in \mathsf{R}\backslash\{\mathsf{rvk}^*\}$ such that rvk_y^* and rvk^* yield y. But this means that rvk_y^* must be of the form $(\mathsf{com}(x), \mathsf{com}(y), *)$. If rvk_y^* has no partner in $\mathsf{R}\backslash\{\mathsf{rvk}^*\}$ we are done and can argue as before. If rvk_y^* has a partner in $\mathsf{R}\backslash\{\mathsf{rvk}_y^*\}$ then it holds that $x \in \mathcal{S}'$, as $y \notin \mathcal{S}'$ (as it is in $\mathcal{T}$). Consequently, in any of these cases we can remove $k-1$ elements from $\mathsf{R}\backslash\{\mathsf{rvk}^*\}$ and obtain a ring R' of size $n-k$ which yields $\mathcal{S}'$. Using the induction hypothesis we conclude that $|\mathcal{S}'| \leq |\mathsf{R}'| \leq n-k$ and therefore $\mathcal{S} \leq n$.

Case 3: R only contains rvk of the form $(\mathsf{com}(x), \mathsf{com}(y), \mathsf{com}(z))$ for distinct x, y, z. None of these rvk can be partnered, and it immediately follows that $\mathcal{S} = \emptyset$.

This concludes the proof. □

We will now show that our scheme is perfectly linkable.

Theorem 5.6: Perfect Linkability of Construction 5.3

Construction 5.3 is perfectly linkable, if SPB is somewhere perfectly binding, Com is perfectly binding and NIP has perfect soundness.

Proof. Assume that the linking adversary $\mathcal{A}$ outputs a ring $\mathsf{R} = (\mathsf{rvk}_1, \ldots, \mathsf{rvk}_\ell)$ of ℓ keys. Then it holds by Lemma theorem 5.5 that R yields at most ℓ distinct keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell$. Assume that $\mathcal{A}$ outputs $\ell+1$ valid message-signature pairs (m_1, ς_1), $\ldots$, $(m_{\ell+1}, \varsigma_{\ell+1})$, where $\varsigma_i = (\tilde{\mathsf{vk}}_i, *, *, *)$. Then by the perfect soundness of NIP, the somewhere perfectly binding property of SPB and the perfect binding property of Com it holds that each $\tilde{\mathsf{vk}}_i$ must be yield-able from one or two keys in R. But this means that $\tilde{\mathsf{vk}}_i$ is in the set $\{\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell\}$. Since this set contains only ℓ elements, at least two of the ς_i must link.

□

5.5.4 Proof of Non-Frameability

We will now show that LRS is non-frameable.

Theorem 5.7: Non-frameability of Construction 5.3

Construction 5.3 has non-frameability, if DS is unforgeable, Com is perfectly binding and NIP is perfectly sound.

The idea of the proof is simple. The only way the adversary can win the experiment is by producing a signature DS* which links to one of the honest keys. In order to achieve this however, the adversary must forge a signature under the key to which this key commits to, which contradicts the unforgeability of the underlying signature scheme.

Proof. Let $\mathcal{A}$ be a PPT adversary against the non-frameability of LRS. Assume that $\mathcal{A}$ makes at most $q = \mathsf{poly}(\lambda)$ key queries. Consider the following hybrids.

$\mathcal{H}_0$: This is the real experiment.

$\mathcal{H}_1$: Identical to $\mathcal{H}_0$, except for the following modification. Let $\mathsf{rvk}_1, \ldots, \mathsf{rvk}_q$ be the keys generated by the experiment, where rvk_i commits to a key vk_i of DS. If in phase 1 the adversary $\mathcal{A}$ outputs a valid message-signature pair $(m^*, \varsigma^* = (\tilde{\mathsf{vk}}, (\mathsf{hk}_h^*)_{j\in[3]}, \pi^*, \sigma^*))$ such that $\tilde{\mathsf{vk}} = \mathsf{vk}_i$ for an uncorrupted rvk_i and $(m, \mathsf{rvk}_i, *)$ has not been queried to the signing oracle, then the experiment aborts and outputs 0.

We will first show that $|\Pr[\mathcal{H}_0(\mathcal{A}) = \mathsf{true}] - \Pr[\mathcal{H}_1(\mathcal{A}) = \mathsf{true}]| < \mathsf{negl}(\lambda)$ given that DS is existentially unforgeable. Let $F(\mathcal{A})$ be the event that $\mathcal{A}$ outputs a signature $\varsigma^* = (\mathsf{vk}^*, (\mathsf{hk}_h)_{j\in[3]}, \pi)$ such that $\mathsf{vk}^* = \mathsf{vk}_i$ for an uncorrupted rvk_i. Clearly, conditioned on $\neg F(\mathcal{A})$ both experiments are identically distributed. Assume therefore towards contradiction that $\Pr[F(\mathcal{A})] \geq \epsilon$ for a non-negligible ϵ. We will sketch a reduction $\mathcal{R}$ such that $\mathcal{R}^{\mathcal{A}}$ breaks the unforgeability of DS with advantage ϵ. $\mathcal{R}$ first guesses an index $i^* \in [q]$ and uses its own input vk^* to generate the key rvk_{i^*}. If $\mathcal{A}$ requests a signature of a message m under rvk_{i^*} and ring $\mathcal{R}$, $\mathcal{R}$ computes the signature in the same way as in $\mathcal{H}_0$, but uses its own signing oracle with input $(m, (\mathsf{hk}^{(i)}, \tilde{\zeta}^{(i)})_{i\in[3]}, \pi)$ to obtain the signature σ. Once $\mathcal{A}$ outputs (m^*, ς^*) with $(\mathsf{vk}^*, (\mathsf{hk}_h^*)_{j\in[3]}, \pi^*, \sigma^*))$, $\mathcal{R}$ checks if $\tilde{\mathsf{vk}} = \mathsf{vk}^*$, and if so outputs $((m^*, (\mathsf{hk}_h^*)_{j\in[3]}, \pi^*), \sigma^*)$.

Clearly, the index i^* is uniformly random in the view of $\mathcal{A}$ and therefore the event $\tilde{\mathsf{vk}} = \mathsf{vk}^*$ happens with probability at least $\frac{1}{q}$. Consequently, it holds that

$$\mathsf{Adv}_{\mathcal{R},\mathcal{A}}^{\mathsf{EUF\text{-}CMA}}(\lambda) \geq \frac{1}{q} \cdot \Pr[F(\mathcal{A})] \geq \frac{\epsilon}{q},$$

which contradicts the unforgeability of DS.

Finally, notice that in $\mathcal{H}_1$ the advantage of $\mathcal{A}$ is 0: Due to the perfect binding property of Com and the perfect soundness of NIP, any signature $\varsigma^\dagger$ that $\mathcal{A}$ generates must use one of the $\mathsf{vk}_1, \ldots, \mathsf{vk}_q$.

If the key $\tilde{\mathsf{vk}}$ used in ς^* is one of $\mathsf{vk}_1, \ldots, \mathsf{vk}_q$ then $\mathcal{A}$ will immediately lose after phase 1. If $\tilde{\mathsf{vk}}$ is different from all keys in $\mathsf{vk}_1, \ldots, \mathsf{vk}_q$, then $\varsigma^\dagger$ does not link to ς^*, and consequently $\mathcal{A}$ loses in phase 2. Consequently, the advantage of $\mathcal{A}$ in $\mathcal{H}_1$ is 0. This concludes the proof. □

5.6 Related Work

After the initial work of Rivest, Shamir and Tauman-Kalai on ring signatures [RST01], a number of works provided constructions in the random oracle model under various computational hardness assumptions [AOS02; Bon+03; HS03]. The scheme of Dodis et al. [Dod+04] was the first to achieve sub-linear size signatures in the ROM. Libert, Peters, Qian [LPQ18] constructed a scheme with logarithmic size ring signatures from DDH in the ROM. Schemes in the CRS model include [SW07; Boy07; SS10; CGS07; Gha13; Gon17] achieving varying degrees of compactness but focusing mainly on practical efficiency. Standard model ring signatures were simultaneously proposed by Chow et al. [Cho+06] and by Bender, Katz, and Morselli [BKM06]. Malavolta and Schröder [MS17] build setup free and constant size ring signatures assuming hardness of a variant of the knowledge of exponent assumption. Linkable Ring signatures were introduced by Liu et al. [LWW04] as linkable spontaneous anonymous group signatures. They propose a notion of linkability which requires that signatures created by the same signer using the same ring must be publicly linkable. In their security model, a scheme achieves a weaker, non-adaptive model of anonymity called signer-ambiguity, if given one signature under signing key sk and ring R as well as a subset of the signing keys corresponding to the keys in the ring which does not include sk, the probability of determining the actual signer as sk is at most negligibly better than guessing one of the remaining keys in the ring uniformly at random. This model is extended by Boyen and Haies [BH18], introducing signing epochs which allow for forward secure notions of anonymity and unforgeability. Recently, several works described linkable ring signature schemes in post-quantum setting, e.g. [Tor+18] based on the hardness of the Ring-SIS problem or [BLO18] based on the Module-SIS and Module-LWE problems. Finally, the idea of replacing NIZK proofs with NIWI proofs in standard model constructions has gained momentum recently, e.g. in the construction of verifiable random functions (VRFs) [Bit17; Goy+17].

6 Outlook

In this chapter we suggest possibilities for future work on the basis of this thesis.

6.1 Recapitulating our Results

First, we offer a pointed summary of our contributions to the study of group signatures and ring signatures in the following comparisons to previous schemes. We denote by n the size of the group in question for group signatures and by ℓ the size of the ring in ring signatures.

Group Signature Schemes. In the following table, we assume a 256-bit (respectively 512-bit) representation of $\mathbb{Z}_q, \mathbb{G}_1$ (respectively $\mathbb{G}_2$) for Type 3 pairings and a 3072-bit factoring and DL modulus with 256-bit key.

Scheme	Signature size* $[bits]$	Membership	Assumptions
[BW07][‡]	6 656	static	q-type
[BBS04]	2 304	static	q-type
[LPY15]	8 448	static	standard
Construction 3.5	6 400	static	standard
[Bic+10]	1 280	dynamic[†]	interactive
[Gro07]	13 056	dynamic	q-type
[LPY15]	14 848	dynamic	standard
[Boo+16]	$\mathcal{O}(\log n)$	fully dynamic	standard
Construction 4.1	$\mathcal{O}(1)$	membership private	standard
+ Construction 3.4	12 056	membership private	standard

[†] The scheme defines additionally a join$\leftrightarrow$issue procedure

[‡] Adapted from type 1 to type 3 pairings as in [LPY15]

Ring Signature Schemes.

	Scheme	Signature size	Assumptions
Trusted Setup	[SW07]	$\mathcal{O}(\ell)$	standard
	[Boy07]	$\mathcal{O}(\ell)$	q-type
	[CGS07]	$\mathcal{O}(\sqrt{\ell})$	q-type
	[MS17]	$\mathcal{O}(1)$	q-type + GGM
No Trusted Setup	[Cho+06]	$\mathcal{O}(\ell)$	q-type
	[BKM06]	$\mathcal{O}(\ell)$	standard
	[MS17]	$\mathcal{O}(\ell)$	q-type + knowledge
	Construction 3.6	$\mathcal{O}(\ell)$	standard
	Construction 3.6 + [CGS07]	$\mathcal{O}(\sqrt{\ell})$	standard
	Construction 5.2	$\mathcal{O}(\log \ell)$	standard

6.2 Future Work

6.2.1 Further Applications of SFPK

Group signature schemes are sometimes described as a kind of precursor primitive to more complicated primitives that have a group-signature-like functionality at their core. In many ways the fully dynamic group signatures of chapter 4 are so much richer in functionality compared to static group signatures that they could be seen this way. Another example of a concept that is closely related to group signatures are anonymous credentials and their delegatable variants as well as direct anonymous attestation. Our work here poses the question if signatures with flexible public keys are equally useful in these settings as they are for static and fully dynamic group signatures. The related work of Crites and Lysyanskaya [CL19] on mercurial signatures shows promise in this direction.

6.2.2 Instantiation of SPB Hashing

Our constructions of linkable and unlinkable ring signature in chapter 5 make crucial use of the concept of somewhere perfectly binding hashing in order to achieve signature size logarithmic in the size of the ring. This makes our constructions asymptotically optimal in size, since we can at most consider rings of polynomial size in the security parameter.

In terms of concrete efficiency, however, at the moment we cannot hope to be close to the practical size of the asymptotically less efficient constructions proposed in chapter 3. The main reason for this is that, compared with constructions based on signatures with flexible public keys, our generic constructions in chapter 5 cannot be fully instantiated in the bilinear pairing setting, and thus cannot make full use of the highly efficient non-interactive proof systems that are known in this setting. Of the components of our construction, only the somewhere perfectly binding hash family has no compatible instantiation at the point of writing. The features of this construction would require it to fit neatly into the pairing product equation framework delineated in section 3.2.3, requiring that verification of a hash can be described in these.

Bibliography

[ALP13] Nuttapong Attrapadung, Benoît Libert, and Thomas Peters. "Efficient Completely Context-Hiding Quotable and Linearly Homomorphic Signatures." In: *PKC 2013: 16th International Conference on Theory and Practice of Public Key Cryptography*. Ed. by Kaoru Kurosawa and Goichiro Hanaoka. Vol. 7778. Lecture Notes in Computer Science. Nara, Japan: Springer, Heidelberg, Germany, Feb. 2013, pp. 386–404. DOI: 10.1007/978-3-642-36362-7_24 (cit. on p. 81).

[AM03] Giuseppe Ateniese and Breno de Medeiros. "Efficient Group Signatures without Trapdoors." In: *Advances in Cryptology – ASIACRYPT 2003*. Ed. by Chi-Sung Laih. Vol. 2894. Lecture Notes in Computer Science. Taipei, Taiwan: Springer, Heidelberg, Germany, Nov. 2003, pp. 246–268. DOI: 10.1007/978-3-540-40061-5_15 (cit. on p. 118).

[AOS02] Masayuki Abe, Miyako Ohkubo, and Koutarou Suzuki. "1-out-of-n Signatures from a Variety of Keys." In: *Advances in Cryptology – ASIACRYPT 2002*. Ed. by Yuliang Zheng. Vol. 2501. Lecture Notes in Computer Science. Queenstown, New Zealand: Springer, Heidelberg, Germany, Dec. 2002, pp. 415–432. DOI: 10.1007/3-540-36178-2_26 (cit. on p. 164).

[Ate+00] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. "A Practical and Provably Secure Coalition-Resistant Group Signature Scheme." In: *Advances in Cryptology – CRYPTO 2000*. Ed. by Mihir Bellare. Vol. 1880. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2000, pp. 255–270. DOI: 10.1007/3-540-44598-6_16 (cit. on p. 118).

[Ate+05a] Giuseppe Ateniese, Jan Camenisch, Susan Hohenberger, and Breno de Medeiros. *Practical Group Signatures without Random Oracles*. Cryptology ePrint Archive, Report 2005/385. 2005 (cit. on p. 118).

[Ate+05b] Giuseppe Ateniese, Daniel H. Chou, Breno de Medeiros, and Gene Tsudik. "Sanitizable Signatures." In: *ESORICS 2005: 10th European Symposium on Research in Computer Security*. Ed. by Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann. Vol. 3679. Lecture Notes in Computer Science. Milan, Italy: Springer, Heidelberg, Germany, Sept. 2005, pp. 159–177. DOI: 10.1007/11555827_10 (cit. on p. 29).

[AW04] Michel Abdalla and Bogdan Warinschi. "On the Minimal Assumptions of Group Signature Schemes." In: *ICICS 04: 6th International Conference on Information and Communication Security*. Ed. by Javier López, Sihan Qing, and Eiji Okamoto. Vol. 3269. Lecture Notes in Computer Science. Malaga, Spain: Springer, Heidelberg, Germany, Oct. 2004, pp. 1–13 (cit. on p. 118).

[Bac+18] Michael Backes, Lucjan Hanzlik, Kamil Kluczniak, and Jonas Schneider. "Signatures with Flexible Public Key: Introducing Equivalence Classes for Public Keys." In: *Advances in Cryptology – ASIACRYPT 2018, Part II*. Ed. by Thomas Peyrin and Steven Galbraith. Vol. 11273. Lecture Notes in Computer Science. Brisbane, Queensland, Australia: Springer, Heidelberg, Germany, Dec. 2018, pp. 405–434. DOI: 10.1007/978-3-030-03329-3_14 (cit. on pp. 4, 8, 30, 82).

[Bac+19] Michael Backes, Nico Döttling, Lucjan Hanzlik, Kamil Kluczniak, and Jonas Schneider. "Ring Signatures: Logarithmic-Size, No Setup - from Standard Assumptions." In: *Advances in Cryptology – EUROCRYPT 2019, Part III*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11478. Lecture Notes in Computer Science. Darmstadt, Germany: Springer, Heidelberg, Germany, May 2019, pp. 281–311. DOI: 10.1007/978-3-030-17659-4_10 (cit. on pp. 9, 122).

[Bal+17] Foteini Baldimtsi, Jan Camenisch, Maria Dubovitskaya, Anna Lysyanskaya, Leonid Reyzin, Kai Samelin, and Sophia Yakoubov. "Accumulators with Applications to Anonymity-Preserving Revocation." In: *EuroS&P 2017*. IEEE, 2017, pp. 301–315. ISBN: 978-1-5090-5762-7. DOI: 10.1109/EuroSP.2017.13. URL: https://doi.org/10.1109/EuroSP.2017.13 (cit. on p. 85).

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. "Short Group Signatures." In: *Advances in Cryptology – CRYPTO 2004*. Ed. by Matthew Franklin. Vol. 3152. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2004, pp. 41–55. DOI: 10.1007/978-3-540-28628-8_3 (cit. on pp. 118, 167).

[BEF18] Dan Boneh, Saba Eskandarian, and Ben Fisch. *Post-Quantum EPID Group Signatures from Symmetric Primitives*. Cryptology ePrint Archive, Report 2018/261. https://eprint.iacr.org/2018/261. 2018 (cit. on p. 119).

[Bel+01] Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. "Key-Privacy in Public-Key Encryption." In: *Advances in Cryptology – ASIACRYPT 2001*. Ed. by Colin Boyd. Vol. 2248. Lecture Notes in Computer Science. Gold Coast, Australia: Springer, Heidelberg, Germany, Dec. 2001, pp. 566–582. DOI: 10.1007/3-540-45682-1_33 (cit. on pp. 17, 18).

[BF01] Dan Boneh and Matthew K. Franklin. “Identity-Based Encryption from the Weil Pairing.” In: *Advances in Cryptology – CRYPTO 2001*. Ed. by Joe Kilian. Vol. 2139. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2001, pp. 213–229. DOI: 10.1007/3-540-44647-8_13 (cit. on pp. 33, 35).

[BF03] Dan Boneh and Matthew K. Franklin. “Identity-Based Encryption from the Weil Pairing.” In: *SIAM J. Comput.* 32.3 (2003), pp. 586–615. DOI: 10.1137/S0097539701398521. URL: https://doi.org/10.1137/S0097539701398521 (cit. on p. 35).

[BH18] Xavier Boyen and Thomas Haines. “Forward-Secure Linkable Ring Signatures.” In: *ACISP 18: 23rd Australasian Conference on Information Security and Privacy*. Ed. by Willy Susilo and Guomin Yang. Vol. 10946. Lecture Notes in Computer Science. Wollongong, NSW, Australia: Springer, Heidelberg, Germany, July 2018, pp. 245–264. DOI: 10.1007/978-3-319-93638-3_15 (cit. on p. 165).

[BHS19] Michael Backes, Lucjan Hanzlik, and Jonas Schneider-Bensch. “Membership Privacy for Fully Dynamic Group Signatures.” In: *ACM CCS 2019: 26th Conference on Computer and Communications Security*. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. ACM Press, Nov. 2019, pp. 2181–2198. DOI: 10.1145/3319535.3354257 (cit. on pp. 9, 30, 83).

[Bic+10] Patrik Bichsel, Jan Camenisch, Gregory Neven, Nigel P. Smart, and Bogdan Warinschi. “Get Shorty via Group Signatures without Encryption.” In: *SCN 10: 7th International Conference on Security in Communication Networks*. Ed. by Juan A. Garay and Roberto De Prisco. Vol. 6280. Lecture Notes in Computer Science. Amalfi, Italy: Springer, Heidelberg, Germany, Sept. 2010, pp. 381–398. DOI: 10.1007/978-3-642-15317-4_24 (cit. on pp. 118, 167).

[Bit17] Nir Bitansky. “Verifiable Random Functions from Non-interactive Witness-Indistinguishable Proofs.” In: *TCC 2017: 15th Theory of Cryptography Conference, Part II*. Ed. by Yael Kalai and Leonid Reyzin. Vol. 10678. Lecture Notes in Computer Science. Baltimore, MD, USA: Springer, Heidelberg, Germany, Nov. 2017, pp. 567–594. DOI: 10.1007/978-3-319-70503-3_19 (cit. on p. 165).

[BKM06] Adam Bender, Jonathan Katz, and Ruggero Morselli. “Ring Signatures: Stronger Definitions, and Constructions Without Random Oracles.” In: *TCC 2006: 3rd Theory of Cryptography Conference*. Ed. by Shai Halevi and Tal Rabin. Vol. 3876. Lecture Notes in Computer Science. New York, NY, USA: Springer, Heidelberg, Germany, Mar. 2006, pp. 60–79. DOI: 10.1007/11681878_4 (cit. on pp. 26, 32, 121, 122, 165, 168).

[Bla+11] Olivier Blazy, Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. "Signatures on Randomizable Ciphertexts." In: *PKC 2011: 14th International Conference on Theory and Practice of Public Key Cryptography*. Ed. by Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi. Vol. 6571. Lecture Notes in Computer Science. Taormina, Italy: Springer, Heidelberg, Germany, Mar. 2011, pp. 403–422. DOI: 10.1007/978-3-642-19379-8_25 (cit. on p. 81).

[BLO18] Carsten Baum, Huang Lin, and Sabine Oechsner. "Towards Practical Lattice-Based One-Time Linkable Ring Signatures." In: *ICICS 18: 20th International Conference on Information and Communication Security*. Ed. by David Naccache, Shouhuai Xu, Sihan Qing, Pierangela Samarati, Gregory Blanc, Rongxing Lu, Zonghua Zhang, and Ahmed Meddahi. Vol. 11149. Lecture Notes in Computer Science. Lille, France: Springer, Heidelberg, Germany, Oct. 2018, pp. 303–322. DOI: 10.1007/978-3-030-01950-1_18 (cit. on p. 165).

[BMW03] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. "Foundations of Group Signatures: Formal Definitions, Simplified Requirements, and a Construction Based on General Assumptions." In: *Advances in Cryptology – EUROCRYPT 2003*. Ed. by Eli Biham. Vol. 2656. Lecture Notes in Computer Science. Warsaw, Poland: Springer, Heidelberg, Germany, May 2003, pp. 614–629. DOI: 10.1007/3-540-39200-9_38 (cit. on pp. 5, 23–26, 66, 84, 118, 119).

[BN06] Paulo S. L. M. Barreto and Michael Naehrig. "Pairing-Friendly Elliptic Curves of Prime Order." In: *SAC 2005: 12th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Bart Preneel and Stafford Tavares. Vol. 3897. Lecture Notes in Computer Science. Kingston, Ontario, Canada: Springer, Heidelberg, Germany, Aug. 2006, pp. 319–331. DOI: 10.1007/11693383_22 (cit. on p. 33).

[Bon+03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. "Aggregate and Verifiably Encrypted Signatures from Bilinear Maps." In: *Advances in Cryptology – EUROCRYPT 2003*. Ed. by Eli Biham. Vol. 2656. Lecture Notes in Computer Science. Warsaw, Poland: Springer, Heidelberg, Germany, May 2003, pp. 416–432. DOI: 10.1007/3-540-39200-9_26 (cit. on p. 164).

[Bon+09] Dan Boneh, David Freeman, Jonathan Katz, and Brent Waters. "Signing a Linear Subspace: Signature Schemes for Network Coding." In: *PKC 2009: 12th International Conference on Theory and Practice of Public Key Cryptography*. Ed. by Stanislaw Jarecki and Gene Tsudik. Vol. 5443. Lecture Notes in Computer Science. Irvine, CA, USA: Springer, Heidelberg, Germany, Mar. 2009, pp. 68–87. DOI: 10.1007/978-3-642-00468-1_5 (cit. on p. 81).

[Boo+15] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Essam Ghadafi, Jens Groth, and Christophe Petit. "Short Accountable Ring Signatures Based on DDH." In: *ESORICS 2015: 20th European Symposium on Research in Computer Security, Part I.* Ed. by Günther Pernul, Peter Y. A. Ryan, and Edgar R. Weippl. Vol. 9326. Lecture Notes in Computer Science. Vienna, Austria: Springer, Heidelberg, Germany, Sept. 2015, pp. 243–265. DOI: 10.1007/978-3-319-24174-6_13 (cit. on pp. 85, 119).

[Boo+16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Essam Ghadafi, and Jens Groth. "Foundations of Fully Dynamic Group Signatures." In: *ACNS 16: 14th International Conference on Applied Cryptography and Network Security.* Ed. by Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider. Vol. 9696. Lecture Notes in Computer Science. Guildford, UK: Springer, Heidelberg, Germany, June 2016, pp. 117–136. DOI: 10.1007/978-3-319-39555-5_7 (cit. on pp. 6, 23, 84–88, 95, 96, 98, 119, 167, 187).

[BOV03] Boaz Barak, Shien Jin Ong, and Salil P. Vadhan. "Derandomization in Cryptography." In: *Advances in Cryptology – CRYPTO 2003.* Ed. by Dan Boneh. Vol. 2729. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2003, pp. 299–315. DOI: 10.1007/978-3-540-45146-4_18 (cit. on pp. 21, 129).

[Boy07] Xavier Boyen. "Mesh Signatures." In: *Advances in Cryptology – EUROCRYPT 2007.* Ed. by Moni Naor. Vol. 4515. Lecture Notes in Computer Science. Barcelona, Spain: Springer, Heidelberg, Germany, May 2007, pp. 210–227. DOI: 10.1007/978-3-540-72540-4_12 (cit. on pp. 165, 168).

[BP15] Nir Bitansky and Omer Paneth. "ZAPs and Non-Interactive Witness Indistinguishability from Indistinguishability Obfuscation." In: *TCC 2015: 12th Theory of Cryptography Conference, Part II.* Ed. by Yevgeniy Dodis and Jesper Buus Nielsen. Vol. 9015. Lecture Notes in Computer Science. Warsaw, Poland: Springer, Heidelberg, Germany, Mar. 2015, pp. 401–427. DOI: 10.1007/978-3-662-46497-7_16 (cit. on pp. 21, 129).

[BSZ05] Mihir Bellare, Haixia Shi, and Chong Zhang. "Foundations of Group Signatures: The Case of Dynamic Groups." In: *Topics in Cryptology – CT-RSA 2005.* Ed. by Alfred Menezes. Vol. 3376. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, Heidelberg, Germany, Feb. 2005, pp. 136–153. DOI: 10.1007/978-3-540-30574-3_11 (cit. on pp. 84, 98, 118, 119).

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. "Efficient Fully Homomorphic Encryption from (Standard) LWE." In: *52nd Annual Symposium on Foundations of Computer Science.* Ed. by Rafail Ostrovsky. Palm Springs, CA, USA: IEEE Computer Society Press, Oct. 2011, pp. 97–106. DOI: 10.1109/FOCS.2011.12 (cit. on p. 125).

[BW06] Xavier Boyen and Brent Waters. "Compact Group Signatures Without Random Oracles." In: *Advances in Cryptology – EUROCRYPT 2006*. Ed. by Serge Vaudenay. Vol. 4004. Lecture Notes in Computer Science. St. Petersburg, Russia: Springer, Heidelberg, Germany, May 2006, pp. 427–444. DOI: 10.1007/11761679_26 (cit. on p. 118).

[BW07] Xavier Boyen and Brent Waters. "Full-Domain Subgroup Hiding and Constant-Size Group Signatures." In: *PKC 2007: 10th International Conference on Theory and Practice of Public Key Cryptography*. Ed. by Tatsuaki Okamoto and Xiaoyun Wang. Vol. 4450. Lecture Notes in Computer Science. Beijing, China: Springer, Heidelberg, Germany, Apr. 2007, pp. 1–15. DOI: 10.1007/978-3-540-71677-8_1 (cit. on pp. 118, 167).

[CG05] Jan Camenisch and Jens Groth. "Group Signatures: Better Efficiency and New Theoretical Aspects." In: *SCN 04: 4th International Conference on Security in Communication Networks*. Ed. by Carlo Blundo and Stelvio Cimato. Vol. 3352. Lecture Notes in Computer Science. Amalfi, Italy: Springer, Heidelberg, Germany, Sept. 2005, pp. 120–133. DOI: 10.1007/978-3-540-30598-9_9 (cit. on p. 118).

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. "The Random Oracle Methodology, Revisited (Preliminary Version)." In: *30th Annual ACM Symposium on Theory of Computing*. Dallas, TX, USA: ACM Press, May 1998, pp. 209–218. DOI: 10.1145/276698.276741 (cit. on p. 19).

[CGS07] Nishanth Chandran, Jens Groth, and Amit Sahai. "Ring Signatures of Sublinear Size Without Random Oracles." In: *ICALP 2007: 34th International Colloquium on Automata, Languages and Programming*. Ed. by Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki. Vol. 4596. Lecture Notes in Computer Science. Wroclaw, Poland: Springer, Heidelberg, Germany, July 2007, pp. 423–434. DOI: 10.1007/978-3-540-73420-8_38 (cit. on pp. 73, 80, 81, 165, 168).

[Cha+14] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. "Malleable Signatures: New Definitions and Delegatable Anonymous Credentials." In: *CSF 2014: IEEE 27st Computer Security Foundations Symposium*. Ed. by Anupam Datta and Cedric Fournet. Vienna, Austria: IEEE Computer Society Press, July 2014, pp. 199–213. DOI: 10.1109/CSF.2014.22 (cit. on p. 81).

[Cho+06] Sherman S. M. Chow, Victor K.-W. Wei, Joseph K. Liu, and Tsz Hon Yuen. "Ring signatures without random oracles." In: *ASIACCS 06: 1st ACM Symposium on Information, Computer and Communications Security*. Ed. by Ferng-Ching Lin, Der-Tsai Lee, Bao-Shuh Lin, Shiuhpyng Shieh, and Sushil

Jajodia. Taipei, Taiwan: ACM Press, Mar. 2006, pp. 297–302 (cit. on pp. 165, 168).

[CL04] Jan Camenisch and Anna Lysyanskaya. "Signature Schemes and Anonymous Credentials from Bilinear Maps." In: *Advances in Cryptology – CRYPTO 2004*. Ed. by Matthew Franklin. Vol. 3152. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2004, pp. 56–72. DOI: 10.1007/978-3-540-28628-8_4 (cit. on p. 118).

[CL19] Elizabeth C. Crites and Anna Lysyanskaya. "Delegatable Anonymous Credentials from Mercurial Signatures." In: *Topics in Cryptology – CT-RSA 2019*. Ed. by Mitsuru Matsui. Vol. 11405. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, Heidelberg, Germany, Mar. 2019, pp. 535–555. DOI: 10.1007/978-3-030-12612-4_27 (cit. on pp. 82, 168).

[CM11] Sanjit Chatterjee and Alfred Menezes. "On cryptographic protocols employing asymmetric pairings - The role of Ψ revisited." In: *Discrete Applied Mathematics* 159.13 (2011), pp. 1311–1322 (cit. on p. 38).

[Cv91] David Chaum and Eugène van Heyst. "Group Signatures." In: *Advances in Cryptology – EUROCRYPT'91*. Ed. by Donald W. Davies. Vol. 547. Lecture Notes in Computer Science. Brighton, UK: Springer, Heidelberg, Germany, Apr. 1991, pp. 257–265. DOI: 10.1007/3-540-46416-6_22 (cit. on pp. 2, 31, 97).

[CZZ17] Sherman S. M. Chow, Haibin Zhang, and Tao Zhang. "Real Hidden Identity-Based Signatures." In: *Financial Cryptography and Data Security - 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers*. Ed. by Aggelos Kiayias. Vol. 10322. Lecture Notes in Computer Science. Springer, 2017, pp. 21–38. ISBN: 978-3-319-70971-0. DOI: 10.1007/978-3-319-70972-7_2. URL: https://doi.org/10.1007/978-3-319-70972-7%5C_2 (cit. on p. 118).

[DDN91] Danny Dolev, Cynthia Dwork, and Moni Naor. "Non-Malleable Cryptography (Extended Abstract)." In: *23rd Annual ACM Symposium on Theory of Computing*. New Orleans, LA, USA: ACM Press, May 1991, pp. 542–552. DOI: 10.1145/103418.103474 (cit. on p. 129).

[DN00a] Ivan Damgård and Jesper Buus Nielsen. "Improved Non-committing Encryption Schemes Based on a General Complexity Assumption." In: *Advances in Cryptology – CRYPTO 2000*. Ed. by Mihir Bellare. Vol. 1880. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2000, pp. 432–450. DOI: 10.1007/3-540-44598-6_27 (cit. on p. 34).

[DN00b] Cynthia Dwork and Moni Naor. "Zaps and Their Applications." In: *41st Annual Symposium on Foundations of Computer Science*. Redondo Beach, CA, USA: IEEE Computer Society Press, Nov. 2000, pp. 283–293. DOI: 10.1109/SFCS.2000.892117 (cit. on pp. 21, 129).

[Dod+04] Yevgeniy Dodis, Aggelos Kiayias, Antonio Nicolosi, and Victor Shoup. "Anonymous Identification in Ad Hoc Groups." In: *Advances in Cryptology – EUROCRYPT 2004*. Ed. by Christian Cachin and Jan Camenisch. Vol. 3027. Lecture Notes in Computer Science. Interlaken, Switzerland: Springer, Heidelberg, Germany, May 2004, pp. 609–626. DOI: 10.1007/978-3-540-24676-3_36 (cit. on pp. 124, 164).

[DS16] David Derler and Daniel Slamanig. *Fully-Anonymous Short Dynamic Group Signatures Without Encryption*. Cryptology ePrint Archive, Report 2016/154. http://eprint.iacr.org/2016/154. 2016 (cit. on p. 119).

[EG14] Alex Escala and Jens Groth. "Fine-Tuning Groth-Sahai Proofs." In: *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*. Ed. by Hugo Krawczyk. Vol. 8383. Lecture Notes in Computer Science. Buenos Aires, Argentina: Springer, Heidelberg, Germany, Mar. 2014, pp. 630–649. DOI: 10.1007/978-3-642-54631-0_36 (cit. on p. 117).

[ElG84] Taher ElGamal. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms." In: *Advances in Cryptology – CRYPTO'84*. Ed. by G. R. Blakley and David Chaum. Vol. 196. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1984, pp. 10–18 (cit. on pp. 123, 129, 132).

[FG18] Georg Fuchsbauer and Romain Gay. "Weakly Secure Equivalence-Class Signatures from Standard Assumptions." In: *PKC 2018: 21st International Conference on Theory and Practice of Public Key Cryptography, Part II*. Ed. by Michel Abdalla and Ricardo Dahab. Vol. 10770. Lecture Notes in Computer Science. Rio de Janeiro, Brazil: Springer, Heidelberg, Germany, Mar. 2018, pp. 153–183. DOI: 10.1007/978-3-319-76581-5_6 (cit. on pp. 43, 79).

[FHS14] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. *Structure-Preserving Signatures on Equivalence Classes and Constant-Size Anonymous Credentials*. Cryptology ePrint Archive, Report 2014/944. http://eprint.iacr.org/2014/944. 2014 (cit. on pp. 40, 80).

[FHS15] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. "Practical Round-Optimal Blind Signatures in the Standard Model." In: *Advances in Cryptology – CRYPTO 2015, Part II*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9216. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2015, pp. 233–253. DOI: 10.1007/978-3-662-48000-7_12 (cit. on pp. 40, 43, 115).

[Fle+16] Nils Fleischhacker, Johannes Krupp, Giulio Malavolta, Jonas Schneider, Dominique Schröder, and Mark Simkin. "Efficient Unlinkable Sanitizable Signatures from Signatures with Re-randomizable Keys." In: *PKC 2016: 19th International Conference on Theory and Practice of Public Key Cryptography, Part I*. Ed. by Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang. Vol. 9614. Lecture Notes in Computer Science. Taipei, Taiwan: Springer, Heidelberg, Germany, Mar. 2016, pp. 301–330. DOI: 10.1007/978-3-662-49384-7_12 (cit. on pp. 81, 82).

[FY05] Jun Furukawa and Shoko Yonezawa. "Group Signatures with Separate and Distributed Authorities." In: *SCN 04: 4th International Conference on Security in Communication Networks*. Ed. by Carlo Blundo and Stelvio Cimato. Vol. 3352. Lecture Notes in Computer Science. Amalfi, Italy: Springer, Heidelberg, Germany, Sept. 2005, pp. 77–90. DOI: 10.1007/978-3-540-30598-9_6 (cit. on p. 118).

[Gen09] Craig Gentry. "Fully homomorphic encryption using ideal lattices." In: *41st Annual ACM Symposium on Theory of Computing*. Ed. by Michael Mitzenmacher. Bethesda, MD, USA: ACM Press, May 2009, pp. 169–178. DOI: 10.1145/1536414.1536440 (cit. on p. 125).

[Gha13] Essam Ghadafi. "Sub-linear Blind Ring Signatures without Random Oracles." In: *14th IMA International Conference on Cryptography and Coding*. Ed. by Martijn Stam. Vol. 8308. Lecture Notes in Computer Science. Oxford, UK: Springer, Heidelberg, Germany, Dec. 2013, pp. 304–323. DOI: 10.1007/978-3-642-45239-0_18 (cit. on p. 165).

[GL89] Oded Goldreich and Leonid A. Levin. "A Hard-Core Predicate for all One-Way Functions." In: *21st Annual ACM Symposium on Theory of Computing*. Seattle, WA, USA: ACM Press, May 1989, pp. 25–32. DOI: 10.1145/73007.73010 (cit. on p. 131).

[Gon17] Alonso González. *A Ring Signature of size $O(\sqrt[3]{n})$ without Random Oracles*. Cryptology ePrint Archive, Report 2017/905. http://eprint.iacr.org/2017/905. 2017 (cit. on p. 165).

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. "Non-interactive Zaps and New Techniques for NIZK." In: *Advances in Cryptology – CRYPTO 2006*. Ed. by Cynthia Dwork. Vol. 4117. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2006, pp. 97–111. DOI: 10.1007/11818175_6 (cit. on pp. 21, 39, 73, 129).

[Goy+17] Rishab Goyal, Susan Hohenberger, Venkata Koppula, and Brent Waters. "A Generic Approach to Constructing and Proving Verifiable Random Functions." In: *TCC 2017: 15th Theory of Cryptography Conference, Part II*.

Ed. by Yael Kalai and Leonid Reyzin. Vol. 10678. Lecture Notes in Computer Science. Baltimore, MD, USA: Springer, Heidelberg, Germany, Nov. 2017, pp. 537–566. DOI: 10.1007/978-3-319-70503-3_18 (cit. on p. 165).

[Gro06] Jens Groth. "Simulation-Sound NIZK Proofs for a Practical Language and Constant Size Group Signatures." In: *ASIACRYPT 2006*. Ed. by Xuejia Lai and Kefei Chen. Vol. 4284. Lecture Notes in Computer Science. Springer, 2006, pp. 444–459. ISBN: 3-540-49475-8. DOI: 10.1007/11935230_29. URL: https://doi.org/10.1007/11935230%5C_29 (cit. on p. 115).

[Gro07] Jens Groth. "Fully Anonymous Group Signatures Without Random Oracles." In: *Advances in Cryptology – ASIACRYPT 2007*. Ed. by Kaoru Kurosawa. Vol. 4833. Lecture Notes in Computer Science. Kuching, Malaysia: Springer, Heidelberg, Germany, Dec. 2007, pp. 164–180. DOI: 10.1007/978-3-540-76900-2_10 (cit. on pp. 118, 167).

[GS08] Jens Groth and Amit Sahai. "Efficient Non-interactive Proof Systems for Bilinear Groups." In: *Advances in Cryptology – EUROCRYPT 2008*. Ed. by Nigel P. Smart. Vol. 4965. Lecture Notes in Computer Science. Istanbul, Turkey: Springer, Heidelberg, Germany, Apr. 2008, pp. 415–432. DOI: 10.1007/978-3-540-78967-3_24 (cit. on pp. 38, 86, 118).

[GSW10] Essam Ghadafi, Nigel P. Smart, and Bogdan Warinschi. "Groth-Sahai Proofs Revisited." In: *PKC 2010: 13th International Conference on Theory and Practice of Public Key Cryptography*. Ed. by Phong Q. Nguyen and David Pointcheval. Vol. 6056. Lecture Notes in Computer Science. Paris, France: Springer, Heidelberg, Germany, May 2010, pp. 177–192. DOI: 10.1007/978-3-642-13013-7_11 (cit. on pp. 36, 39).

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. "Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based." In: *Advances in Cryptology – CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2013, pp. 75–92. DOI: 10.1007/978-3-642-40041-4_5 (cit. on p. 125).

[HK08] Dennis Hofheinz and Eike Kiltz. "Programmable Hash Functions and Their Applications." In: *Advances in Cryptology – CRYPTO 2008*. Ed. by David Wagner. Vol. 5157. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2008, pp. 21–38. DOI: 10.1007/978-3-540-85174-5_2 (cit. on pp. 36, 38).

[HS03] Javier Herranz and Germán Sáez. "Forking Lemmas for Ring Signature Schemes." In: *Progress in Cryptology - INDOCRYPT 2003: 4th International Conference in Cryptology in India*. Ed. by Thomas Johansson and Subhamoy

Maitra. Vol. 2904. Lecture Notes in Computer Science. New Delhi, India: Springer, Heidelberg, Germany, Dec. 2003, pp. 266–279 (cit. on p. 164).

[HS14] Christian Hanser and Daniel Slamanig. “Structure-Preserving Signatures on Equivalence Classes and Their Application to Anonymous Credentials.” In: *Advances in Cryptology – ASIACRYPT 2014, Part I.* Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8873. Lecture Notes in Computer Science. Kaoshiung, Taiwan, R.O.C.: Springer, Heidelberg, Germany, Dec. 2014, pp. 491–511. DOI: 10.1007/978-3-662-45611-8_26 (cit. on pp. 4, 31, 40, 42).

[HW15] Pavel Hubacek and Daniel Wichs. “On the Communication Complexity of Secure Function Evaluation with Long Output.” In: *ITCS 2015: 6th Conference on Innovations in Theoretical Computer Science.* Ed. by Tim Roughgarden. Rehovot, Israel: Association for Computing Machinery, Jan. 2015, pp. 163–172. DOI: 10.1145/2688073.2688105 (cit. on pp. 124, 125, 132).

[Jac+19] Dennis Jackson, Cas Cremers, Katriel Cohn-Gordon, and Ralf Sasse. “Seems Legit: Automated Analysis of Subtle Attacks on Protocols that Use Signatures.” In: *ACM CCS 2019: 26th Conference on Computer and Communications Security.* Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. ACM Press, Nov. 2019, pp. 2165–2180. DOI: 10.1145/3319535.3339813 (cit. on p. 15).

[Joh+02] Robert Johnson, David Molnar, Dawn Xiaodong Song, and David Wagner. “Homomorphic Signature Schemes.” In: *Topics in Cryptology – CT-RSA 2002.* Ed. by Bart Preneel. Vol. 2271. Lecture Notes in Computer Science. San Jose, CA, USA: Springer, Heidelberg, Germany, Feb. 2002, pp. 244–262. DOI: 10.1007/3-540-45760-7_17 (cit. on p. 81).

[KY05a] Aggelos Kiayias and Moti Yung. “Efficient Secure Group Signatures with Dynamic Joins and Keeping Anonymity Against Group Managers.” In: *Mycrypt 2005.* 2005, pp. 151–170. DOI: 10.1007/11554868_11. URL: https://doi.org/10.1007/11554868_11 (cit. on p. 118).

[KY05b] Aggelos Kiayias and Moti Yung. “Group Signatures with Efficient Concurrent Join.” In: *Advances in Cryptology – EUROCRYPT 2005.* Ed. by Ronald Cramer. Vol. 3494. Lecture Notes in Computer Science. Aarhus, Denmark: Springer, Heidelberg, Germany, May 2005, pp. 198–214. DOI: 10.1007/11426639_12 (cit. on pp. 84, 98).

[KY06] Aggelos Kiayias and Moti Yung. “Secure scalable group signature with dynamic joins and separable authorities.” In: *IJSN* 1.1/2 (2006), pp. 24–45. DOI: 10.1504/IJSN.2006.010821. URL: https://doi.org/10.1504/IJSN.2006.010821 (cit. on pp. 84, 98, 119).

[KZ07] Aggelos Kiayias and Hong-Sheng Zhou. "Hidden Identity-Based Signatures." In: *Financial Cryptography and Data Security, 11th International Conference, FC 2007, and 1st International Workshop on Usable Security, USEC 2007, Scarborough, Trinidad and Tobago, February 12-16, 2007. Revised Selected Papers*. Ed. by Sven Dietrich and Rachna Dhamija. Vol. 4886. Lecture Notes in Computer Science. Springer, 2007, pp. 134–147. ISBN: 978-3-540-77365-8. DOI: 10.1007/978-3-540-77366-5_14. URL: https://doi.org/10.1007/978-3-540-77366-5%5C_14 (cit. on p. 117).

[LAZ19] Xingye Lu, Man Ho Au, and Zhenfei Zhang. "Raptor: A Practical Lattice-Based (Linkable) Ring Signature." In: *ACNS 19: 17th International Conference on Applied Cryptography and Network Security*. Ed. by Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung. Vol. 11464. Lecture Notes in Computer Science. Bogota, Colombia: Springer, Heidelberg, Germany, June 2019, pp. 110–130. DOI: 10.1007/978-3-030-21568-2_6 (cit. on pp. 126, 127).

[Lin+18] San Ling, Khoa Nguyen, Huaxiong Wang, and Yanhong Xu. "Constant-Size Group Signatures from Lattices." In: *PKC 2018: 21st International Conference on Theory and Practice of Public Key Cryptography, Part II*. Ed. by Michel Abdalla and Ricardo Dahab. Vol. 10770. Lecture Notes in Computer Science. Rio de Janeiro, Brazil: Springer, Heidelberg, Germany, Mar. 2018, pp. 58–88. DOI: 10.1007/978-3-319-76581-5_3 (cit. on p. 119).

[LPQ18] Benoît Libert, Thomas Peters, and Chen Qian. "Logarithmic-Size Ring Signatures with Tight Security from the DDH Assumption." In: *ESORICS 2018: 23rd European Symposium on Research in Computer Security, Part II*. Ed. by Javier López, Jianying Zhou, and Miguel Soriano. Vol. 11099. Lecture Notes in Computer Science. Barcelona, Spain: Springer, Heidelberg, Germany, Sept. 2018, pp. 288–308. DOI: 10.1007/978-3-319-98989-1_15 (cit. on p. 165).

[LPY12a] Benoît Libert, Thomas Peters, and Moti Yung. "Group Signatures with Almost-for-Free Revocation." In: *Advances in Cryptology – CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2012, pp. 571–589. DOI: 10.1007/978-3-642-32009-5_34 (cit. on p. 118).

[LPY12b] Benoît Libert, Thomas Peters, and Moti Yung. "Scalable Group Signatures with Revocation." In: *Advances in Cryptology – EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. Lecture Notes in Computer Science. Cambridge, UK: Springer, Heidelberg, Germany, Apr. 2012, pp. 609–627. DOI: 10.1007/978-3-642-29011-4_36 (cit. on p. 118).

[LPY15] Benoît Libert, Thomas Peters, and Moti Yung. “Short Group Signatures via Structure-Preserving Signatures: Standard Model Security from Simple Assumptions.” In: *Advances in Cryptology – CRYPTO 2015, Part II*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9216. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2015, pp. 296–316. DOI: 10.1007/978-3-662-48000-7_15 (cit. on pp. 80, 86, 116, 119, 167).

[LWW04] Joseph K. Liu, Victor K. Wei, and Duncan S. Wong. “Linkable Spontaneous Anonymous Group Signature for Ad Hoc Groups (Extended Abstract).” In: *ACISP 04: 9th Australasian Conference on Information Security and Privacy*. Ed. by Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan. Vol. 3108. Lecture Notes in Computer Science. Sydney, NSW, Australia: Springer, Heidelberg, Germany, July 2004, pp. 325–335. DOI: 10.1007/978-3-540-27800-9_28 (cit. on pp. 6, 165).

[MS17] Giulio Malavolta and Dominique Schröder. “Efficient Ring Signatures in the Standard Model.” In: *Advances in Cryptology – ASIACRYPT 2017, Part II*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10625. Lecture Notes in Computer Science. Hong Kong, China: Springer, Heidelberg, Germany, Dec. 2017, pp. 128–157. DOI: 10.1007/978-3-319-70697-9_5 (cit. on pp. 32, 81, 165, 168).

[MVO91] Alfred Menezes, Scott A. Vanstone, and Tatsuaki Okamoto. “Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field.” In: *23rd Annual ACM Symposium on Theory of Computing*. New Orleans, LA, USA: ACM Press, May 1991, pp. 80–89. DOI: 10.1145/103418.103434 (cit. on p. 32).

[Noe15] Shen Noether. *Ring Signature Confidential Transactions for Monero*. Cryptology ePrint Archive, Report 2015/1098. http://eprint.iacr.org/2015/1098. 2015 (cit. on p. 6).

[Oka+15] Tatsuaki Okamoto, Krzysztof Pietrzak, Brent Waters, and Daniel Wichs. “New Realizations of Somewhere Statistically Binding Hashing and Positional Accumulators.” In: *Advances in Cryptology – ASIACRYPT 2015, Part I*. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9452. Lecture Notes in Computer Science. Auckland, New Zealand: Springer, Heidelberg, Germany, Nov. 2015, pp. 121–145. DOI: 10.1007/978-3-662-48797-6_6 (cit. on pp. 125, 135).

[Reg05] Oded Regev. “On lattices, learning with errors, random linear codes, and cryptography.” In: *37th Annual ACM Symposium on Theory of Computing*. Ed. by Harold N. Gabow and Ronald Fagin. Baltimore, MA, USA: ACM Press, May 2005, pp. 84–93. DOI: 10.1145/1060590.1060603 (cit. on pp. 123, 129, 132).

[RST01] Ronald L. Rivest, Adi Shamir, and Yael Tauman. "How to Leak a Secret." In: *Advances in Cryptology – ASIACRYPT 2001*. Ed. by Colin Boyd. Vol. 2248. Lecture Notes in Computer Science. Gold Coast, Australia: Springer, Heidelberg, Germany, Dec. 2001, pp. 552–565. DOI: 10.1007/3-540-45682-1_32 (cit. on pp. 3, 31, 164).

[Ruf+18] Tim Ruffing, Sri Aravinda Thyagarajan, Viktoria Ronge, and Dominique Schröder. "Burning Zerocoins for Fun and for Profit - A Cryptographic Denial-of-Spending Attack on the Zerocoin Protocol." In: *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. 2018, pp. 116–119. DOI: 10.1109/CVCBT.2018.00023 (cit. on p. 129).

[Sak+12] Yusuke Sakai, Jacob C. N. Schuldt, Keita Emura, Goichiro Hanaoka, and Kazuo Ohta. "On the Security of Dynamic Group Signatures: Preventing Signature Hijacking." In: *PKC 2012: 15th International Conference on Theory and Practice of Public Key Cryptography*. Ed. by Marc Fischlin, Johannes Buchmann, and Mark Manulis. Vol. 7293. Lecture Notes in Computer Science. Darmstadt, Germany: Springer, Heidelberg, Germany, May 2012, pp. 715–732. DOI: 10.1007/978-3-642-30057-8_42 (cit. on pp. 84, 98).

[Sch90] Claus-Peter Schnorr. "Efficient Identification and Signatures for Smart Cards." In: *Advances in Cryptology – CRYPTO'89*. Ed. by Gilles Brassard. Vol. 435. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1990, pp. 239–252. DOI: 10.1007/0-387-34805-0_22 (cit. on p. 7).

[SS10] Sven Schäge and Jörg Schwenk. "A CDH-Based Ring Signature Scheme with Short Signatures and Public Keys." In: *FC 2010: 14th International Conference on Financial Cryptography and Data Security*. Ed. by Radu Sion. Vol. 6052. Lecture Notes in Computer Science. Tenerife, Canary Islands, Spain: Springer, Heidelberg, Germany, Jan. 2010, pp. 129–142 (cit. on p. 165).

[SW07] Hovav Shacham and Brent Waters. "Efficient Ring Signatures Without Random Oracles." In: *PKC 2007: 10th International Conference on Theory and Practice of Public Key Cryptography*. Ed. by Tatsuaki Okamoto and Xiaoyun Wang. Vol. 4450. Lecture Notes in Computer Science. Beijing, China: Springer, Heidelberg, Germany, Apr. 2007, pp. 166–180. DOI: 10.1007/978-3-540-71677-8_12 (cit. on pp. 165, 168).

[Tor+18] Wilson Abel Alberto Torres, Ron Steinfeld, Amin Sakzad, Joseph K. Liu, Veronika Kuchta, Nandita Bhattacharjee, Man Ho Au, and Jacob Cheng. "Post-Quantum One-Time Linkable Ring Signature and Application to Ring Confidential Transactions in Blockchain (Lattice RingCT v1.0)." In: *ACISP 18: 23rd Australasian Conference on Information Security and Privacy*. Ed. by Willy Susilo and Guomin Yang. Vol. 10946. Lecture Notes in Computer

Science. Wollongong, NSW, Australia: Springer, Heidelberg, Germany, July 2018, pp. 558–576. DOI: 10.1007/978-3-319-93638-3_32 (cit. on p. 165).

[TW04] Patrick P. Tsang and Victor K. Wei. *Short Linkable Ring Signatures for E-Voting, E-Cash and Attestation.* Cryptology ePrint Archive, Report 2004/281. http://eprint.iacr.org/2004/281. 2004 (cit. on p. 6).

[TX03] Gene Tsudik and Shouhuai Xu. "Accumulating Composites and Improved Group Signing." In: *Advances in Cryptology – ASIACRYPT 2003.* Ed. by Chi-Sung Laih. Vol. 2894. Lecture Notes in Computer Science. Taipei, Taiwan: Springer, Heidelberg, Germany, Nov. 2003, pp. 269–286. DOI: 10.1007/978-3-540-40061-5_16 (cit. on p. 118).

[Ver01] Eric R. Verheul. "Self-Blindable Credential Certificates from the Weil Pairing." In: *Advances in Cryptology – ASIACRYPT 2001.* Ed. by Colin Boyd. Vol. 2248. Lecture Notes in Computer Science. Gold Coast, Australia: Springer, Heidelberg, Germany, Dec. 2001, pp. 533–551. DOI: 10.1007/3-540-45682-1_31 (cit. on pp. 66, 81).

[Wat05] Brent R. Waters. "Efficient Identity-Based Encryption Without Random Oracles." In: *Advances in Cryptology – EUROCRYPT 2005.* Ed. by Ronald Cramer. Vol. 3494. Lecture Notes in Computer Science. Aarhus, Denmark: Springer, Heidelberg, Germany, May 2005, pp. 114–127. DOI: 10.1007/11426639_7 (cit. on p. 38).

Formal Oracle Definitions

We state the full formal definition of the oracles used in the model of fully dynamic group signatures laid out in chapter 4. We have highlighted differences from the original presentation due to [Boo+16], which arise from our treatment of membership privacy and functional tracing soundness.

AddUser(uid)

```
$\textbf{if } \mathsf{uid} \in \mathbf{H} \cup \mathbf{C} \textbf{ then return } \bot$
$(\mathsf{usk}[\mathsf{uid}], \mathsf{upk}[\mathsf{uid}]) \leftarrow \mathsf{DGS.UserKeyGen}(1^\lambda)$
$\mathbf{H} := \mathbf{H} \cup \{\mathsf{uid}\}$
$\vec{\mathsf{gsk}}[\mathsf{uid}] := \bot;$
$\mathsf{proceed}_{\mathsf{issuer}}[\mathsf{uid}]? := \texttt{continue}$
$\mathsf{state}_{\mathsf{user}}[\mathsf{uid}] := (\eta_{now}, \mathsf{gpk}, \mathsf{uid}, \mathsf{usk}[\mathsf{uid}])$
$\mathsf{state}_{\mathsf{issuer}}[\mathsf{uid}] := (\eta_{now}, \mathsf{msk}, \mathsf{uid}, \mathsf{upk}[\mathsf{uid}])$
$\textbf{while } \mathsf{proceed}_{\mathsf{user}}[\mathsf{uid}]? = \texttt{continue}$
          $\textbf{and } \mathsf{proceed}_{\mathsf{issuer}}[\mathsf{uid}]? = \texttt{continue} \textbf{ do}$
   $(\mathsf{state}_{\mathsf{issuer}}[\mathsf{uid}], M_{\mathsf{DGS.Join}}, \mathsf{proceed}_{\mathsf{issuer}}[\mathsf{uid}]?) \leftarrow \mathsf{DGS.Issue}(\mathsf{state}_{\mathsf{issuer}}[\mathsf{uid}], M_{\mathsf{DGS.Issue}})$
   $(\mathsf{state}_{\mathsf{user}}[\mathsf{uid}], M_{\mathsf{DGS.Issue}}, \mathsf{proceed}_{\mathsf{user}}[\mathsf{uid}]?) \leftarrow \mathsf{DGS.Join}(\mathsf{state}_{\mathsf{user}}[\mathsf{uid}], M_{\mathsf{DGS.Join}})$
$\textbf{if } \mathsf{proceed}_{\mathsf{issuer}}[\mathsf{uid}]? = \texttt{accept}$
 $\textbf{then } \mathsf{reg}[\mathsf{uid}] := \mathsf{state}_{\mathsf{issuer}}[\mathsf{uid}]$
$\textbf{if } \mathsf{proceed}_{\mathsf{user}}[\mathsf{uid}]? = \texttt{accept}$
 $\textbf{then } \vec{\mathsf{gsk}}[\mathsf{uid}] := \mathsf{state}_{\mathsf{user}}[\mathsf{uid}]$
$\textbf{return } (\mathsf{info}_{now}, \mathsf{upk}[\mathsf{uid}])$
```

$\mathsf{Challenge}_b(\mathsf{info}_\eta, \mathsf{uid}_0, \mathsf{uid}_1, m)$

```
if $\{\mathsf{uid}_0, \mathsf{uid}_1\} \cap \mathbf{H} \neq \{\mathsf{uid}_0, \mathsf{uid}_1\}$
 or $\exists b \in \{0,1\}$ s.t. $\vec{\mathsf{gsk}}[\mathsf{uid}_b] = \bot$
 or $\mathsf{DGS.Active?}(\mathsf{info}_\eta, \mathsf{reg}, \mathsf{uid}_b) = \mathsf{false}$
    then return $\bot$
$\sigma \leftarrow \mathsf{DGS.Sign}(\mathsf{gpk}, \vec{\mathsf{gsk}}[\mathsf{uid}_b], \mathsf{info}_\eta, m)$
$\mathbf{Q}^* := \mathbf{Q}^* \cup \{(m, \sigma, \eta)\}$
return $\sigma$
```

$\mathsf{Privacy}_{b,\mathsf{uid}_0,\mathsf{uid}_1,\eta^*}(m, \eta)$

```
if $\eta < \eta^*$ then return $\bot$
if $\mathsf{invert} = \mathbf{true}$
   $\sigma \leftarrow \mathsf{DGS.Sign}(\mathsf{gpk}, \vec{\mathsf{gsk}}[\mathsf{uid}_{(1-b)}], \mathsf{info}_\eta, m)$
else
   $\sigma \leftarrow \mathsf{DGS.Sign}(\mathsf{gpk}, \vec{\mathsf{gsk}}[\mathsf{uid}_b], \mathsf{info}_\eta, m)$
$\mathbf{Q}^* := \mathbf{Q}^* \cup \{(m, \sigma, \eta)\}$
return $\sigma$
```

$\mathsf{ReadReg}(\mathsf{uid})$

```
return $\mathsf{reg}[\mathsf{uid}]$
```

$\mathsf{ModifyReg}(\mathsf{uid}, \mathit{val})$

```
$\mathsf{reg}[\mathsf{uid}] := \mathit{val}$
```

$\mathsf{Reveal}(\mathsf{uid})$

```
if $\mathsf{uid} \notin \mathbf{H} \setminus (\mathbf{C} \cup \mathbf{B})$
or $\mathsf{uid} \in \mathbf{U}$
   then return $\bot$
$\mathbf{B} := \mathbf{B} \cup \{\mathsf{uid}\}$
return $(\mathsf{usk}[\mathsf{uid}], \vec{\mathsf{gsk}}[\mathsf{uid}])$
```

$\mathsf{CorruptUser}(\mathsf{uid}, \mathsf{pk})$

```
if $\mathsf{uid} \in \mathbf{H} \cup \mathbf{C} \cup \mathbf{U}$ then return $\bot$
$\mathbf{C} := \mathbf{C} \cup \{\mathsf{uid}\}$
$\mathsf{upk}[\mathsf{uid}] := \mathsf{pk}$
return accept
```

$\mathsf{Sign}(\mathsf{uid}, m, \eta)$

```
if $\mathsf{uid} \notin \mathbf{H} \setminus \mathbf{U}$
    or $\vec{\mathsf{gsk}}[\mathsf{uid}] = \perp$
    or $\mathsf{info}_\eta = \perp$
    or $\mathsf{DGS.Active?}(\mathsf{info}_\eta, \mathsf{reg}, \mathsf{uid}) = \mathsf{false}$
    then return $\perp$
$\sigma \leftarrow \mathsf{DGS.Sign}(\mathsf{gpk}, \vec{\mathsf{gsk}}[\mathsf{uid}], \mathsf{info}_\eta, m)$
$\mathbf{Q} := \mathbf{Q} \cup \{(\mathsf{uid}, m, \sigma, \eta)\}$
return $\sigma$
```

$\mathsf{SendM}(\mathsf{uid}, M_{\mathsf{in}})$

```
if $\mathsf{uid} \notin \mathbf{C}$ or $\mathsf{proceed}_{\mathsf{issuer}}[\mathsf{uid}]? \neq \mathtt{continue}$
or $\mathsf{uid} \in \mathbf{U}$ then return $\perp$
$\mathsf{state}_{\mathsf{issuer}}[\mathsf{uid}] := (\eta_{now}, \mathsf{msk}, \mathsf{uid}, \mathsf{upk}[\mathsf{uid}])$
$(\mathsf{state}_{\mathsf{issuer}}[\mathsf{uid}], M_{\mathsf{out}}, \mathsf{proceed}_{\mathsf{issuer}}[\mathsf{uid}]?) \leftarrow \mathsf{DGS.Issue}(\mathsf{state}_{\mathsf{issuer}}[\mathsf{uid}], M_{\mathsf{in}})$
if $\mathsf{proceed}_{\mathsf{issuer}}[\mathsf{uid}]? = \mathtt{accept}$ then $\mathsf{reg}[\mathsf{uid}] := \mathsf{state}_{\mathsf{issuer}}[\mathsf{uid}]$
return $(M_{\mathsf{out}}, \mathsf{proceed}_{\mathsf{issuer}}[\mathsf{uid}]?)$
```

$\mathsf{SendU}(\mathsf{uid}, M_{\mathsf{in}})$

```
if $\mathsf{uid} \in \mathbf{C} \cup \mathbf{B}$ then return $\perp$
if $\mathsf{uid} \notin \mathbf{H}$ then
    $\mathbf{H} := \mathbf{H} \cup \{\mathsf{uid}\}$
    $(\mathsf{usk}[\mathsf{uid}], \mathsf{upk}[\mathsf{uid}]) \leftarrow \mathsf{DGS.UserKeyGen}(1^\lambda)$
    $\vec{\mathsf{gsk}}[\mathsf{uid}] := \perp; M_{\mathsf{in}} := \perp$
if $\mathsf{proceed}_{\mathsf{user}}[\mathsf{uid}]? \neq \mathtt{continue}$ then return $\perp$
if $\mathsf{state}_{\mathsf{user}}[\mathsf{uid}] = \perp$ then $\mathsf{state}_{\mathsf{user}}[\mathsf{uid}] := (\eta_{now}, \mathsf{gpk}, \mathsf{uid}, \mathsf{usk}[\mathsf{uid}])$
$(\mathsf{state}_{\mathsf{user}}[\mathsf{uid}], M_{\mathsf{out}}, \mathsf{proceed}_{\mathsf{user}}[\mathsf{uid}]?) \leftarrow \mathsf{DGS.Join}(\mathsf{state}_{\mathsf{user}}[\mathsf{uid}], M_{\mathsf{in}})$
if $\mathsf{proceed}_{\mathsf{user}}[\mathsf{uid}]? = \mathtt{accept}$ then $\vec{\mathsf{gsk}}[\mathsf{uid}] := \mathsf{state}_{\mathsf{user}}[\mathsf{uid}]$
return $(M_{\mathsf{out}}, \mathsf{proceed}_{\mathsf{user}}[\mathsf{uid}]?)$
```

Trace$(m, \sigma, \mathsf{info}_\eta)$

```
**if** $\mathsf{DGS.Verify}(\mathsf{gpk}, \mathsf{info}_\eta, m, \sigma) = \texttt{reject}$ **or** $(m, \sigma, \eta) \in \mathbf{Q}^*$
    **then** **return** $(\perp, \perp)$
**return** $\mathsf{DGS.Trace}(\mathsf{gpk}, \mathsf{tsk}, \mathsf{info}_\eta, \mathsf{reg}, m, \sigma)$
```

UpdateGroup(R)

```
**if** $\mathbf{U} \cap R \neq \emptyset$ **then** **return** $\perp$
**return** $\mathsf{DGS.Update}(\mathsf{gpk}, \mathsf{msk}, \mathsf{info}_{now}, R, \mathsf{reg})$
```