Saarland University

Faculty for Mathematics and Computer Science

Department of Computer Science

Privacy Enhancing Technologies: Protocol verification, implementation and specification

Dissertation
zur Erlangung des Grades
des Doktors der Ingenieurwissenschaften
der Fakultät für Mathematik und Informatik
der Universität des Saarlandes

von

Fabian Aaron Bendun

Saarbrücken,
September 2020

Tag des Kolloquiums: 07. September 2020

Dekan: Prof. Dr. Thomas Schuster

Prüfungsausschuss:
Vorsitzender: Prof. Dr. Bernd Finkbeiner
Berichterstattende: Prof. Dr. Dr. h.c. Michael Backes
Prof. Dr. Cas Cremers
Akademischer Mitarbeiter: Dr. Zhikun Zhang

Zusammenfassung

In dieser Arbeit werden neue Methoden zur Verifikation, Implementierung und Spezifikation im von Protokollen vorgestellt. Ein besonderer Fokus liegt dabei auf Datenschutz-Eigenschaften und dem Schutz der Privatsphäre. Im ersten Teil dieser Arbeit geht der Author auf die Protokoll-Verifikation ein und stellt ein Modell zur Verifikation vor, dass sogenannte Zero-Knowledge (ZK) Beweise enthält. Diese ZK Beweise sind ein kryptographisches primitiv, dass insbesondere zum Verstecken von Informationen geeignet ist und somit zum Schutz der Privatsphäre dient. Das hier vorgestellte Modell gibt eine Liste von Kriterien, welche eine Implementierung der genutzten kryptographischen Primitive erfüllen muss, damit die verifikationen im Modell sich auf Implementierungen übertragen lassen. In Bezug auf ZK Beweise sind diese Kriterien schächer als die vorangegangener Arbeiten. Der zweite Teil der Arbeit wendet sich der Implementierung von Protokollen zu. Hierbei werden dann ZK Beweise verwendet um sichere Mehrparteienberechnungen zu verbessern. Im dritten und letzten Teil der Arbeit wird eine neuartige Art der Spezifikation von Datenschutz-Richtlinien erläutert. Diese geht nicht von Richtlinien aus, sondern von der Rechtsprechung. Der Vorteil ist, dass in der Rechtsprechung konkrete Abwägungen getroffen werden, die Gesetze und Richtlinien nicht enthalten.

Abstract

In this thesis, we present novel methods for verifying, implementing and specifying protocols. In particular, we focus properties modeling data protection and the protection of privacy. In the first part of the thesis, the author introduces protocol verification and presents a model for verification that encompasses so-called Zero-Knowledge (ZK) proofs. These ZK proofs are a cryptographic primitive that is particularly suited for hiding information and hence serves the protection of privacy. The here presented model gives a list of criteria which allows the transfer of verification results from the model to the implementation if the criteria are met by the implementation. In particular, the criteria are less demanding than the ones of previous work regarding ZK proofs. The second part of the thesis contributes to the area of protocol implementations. Hereby, ZK proofs are used in order to improve multi-party computations. The third and last part of the thesis explains a novel approach for specifying data protection policies. Instead of relying on policies, this approach relies on actual legislation. The advantage of relying on legislation is that often a fair balancing is introduced which is typically not contained in regulations or policies.

Background of this Dissertation

This dissertation is based on three peer-reviewed papers that have been accepted at influential ans well-ranked conferences in the field of computer security, in each of which the author contributed as one of the main authors. In addition, the author was the co-author of other peer-reviewed papers in the field of cryptography and information security [**SiBeAsBaMaDr:15**, **BaBeMaMoPe:15**]. This dissertation focuses on the following three papers:

- In the paper "Computational Soundness of Symbolic Zero-Knowledge Proofs: Weaker Assumptions and Mechanized Verification" [**BaBeUn:13**] (published at POST'13), as well as in the extended technical report, the author contributed significantly to the design of the model, the proof of its soundness and its application on proof of concept protocols in order to show the symbolic model's amenability for automated verification. The symbolic model and used proof technique was influenced by "CoSP: a general framework for computational soundness proofs" [**BaHoUn:09:cosp**] of Backes, Hofheinz and Unruh, and the argumentation regarding zero-knowledge followed "Computational soundness of symbolic zero-knowledge proofs" [**BaUn:10:cs:zk**] of Backes and Unruh. However, the assumptions used in Y have weakened, leading to a smaller gap regarding practically deployed implementations of zero-knowledge protocols. This gap has been closed by follow up work [**BaBeMaMoPe:15**] together with Michael Backes, Esfandiar Mohammadi, Kim Pecina and Matteo Maffei and is not in this work.

- In the paper "Asynchronous MPC with a strict honest majority using non-equivocation" [**BaBeChKa:14**] (published at PODC'14), the author contributed in the exploration of the recently discovered principle of *non-equivocation* regarding the class of asynchronous multi-party computations (MPC). The principle is based on non-standard assumptions and achieves that every party in a multi-party setting has to stick to her send messages. Using this principle, it has been possible to circumvent impossibility results for asynchronous MPC that required a strict honest two-third majority. The author contributed as on of the main author in the design of the protocols and their security proofs. A first idea sketch of this work has also been published as brief announcement at PODC'12 [**BaBeKa:12**].

- In "PriCL: Creating a Precedent, a Framework for Reasoning about Privacy Case Law" [**BaBeHoMa:15**] (published at POST'15), the author worked on a new specification approach for privacy requirements. As one of the main authors, he formalized legal requirements in a framework for automated reasoning. In contrast to related work, the framework bases on dynamic legal precedents instead of static regulation texts. In addition to the specification, the author analyzed the logical properties as well as the complexity of common reasoning tasks. In this work, the requirement analysis with respect to the legal requirements was done by the co-author Ninja Marnau. Finally, the author supervised a Master's thesis that evaluated techniques in order to create a database for developed framework. The Master's thesis was written by Vikash Patel.

Contents

1 Introduction **1**

2 Privacy Protocol Verification: Zero-Knowledge Proofs in CoSP **3**

2.1 CoSP framework for computational soundness proofs 5
2.1.1 Symbolic Model 5
2.1.2 Computational Model 8
2.1.3 Computational soundness. 10
2.2 Zero Knowledge Proofs 12
2.3 The Symbolic Model 18
2.4 Computational Soundness 22
2.4.1 Theorem conditions 22
2.4.2 Proof of the Computational Soundess 29
2.5 Protocol Verification using the applied π-calculus 40
2.6 Example relations 42
2.7 An Impossibility Result for Computational Soundness of Symbolic ZK Proofs . 44
2.8 Conclusions 46

3 Privacy Protocol Implementation: Designing Protocols for Multi-Party Computations (MPC) **47**

3.1 Multi-Party Computations & Related Work 48
3.1.1 Contribution and Comparison 49
3.1.2 Preliminaries 50
3.1.3 Employed Primitives 53
3.1.4 Secret Sharing Notations 56
3.2 Overview of Our NeqAMPC Protocol 56
3.2.1 Pre-processing Phase 57
3.2.2 Important Sub-protocols for the Preprocessing Phase 58
3.3 Employed AVSS Protocol 60
3.4 Supervised Sharing Protocols 61
3.4.1 Protocol Sup-Sh: Supervised $[\cdot]$-sharing 62
3.4.2 Supervised Pre-multiplication Protocol 66
3.5 Supervised Triple Generation 68

3.5.1 Generating the Second Component of the Triple . . . 69
3.5.2 Generating First and Third Components of the Triple . . . 72
3.5.3 Sup-Second+Sup-FirAndThd⟹SupTripGen . . . 74
3.6 The NeqAMPC Protocol . . . 75
3.7 Non-equivocation Implementations . . . 76
3.7.1 Realizing the Neq mechanism using TrInc . . . 77
3.8 Instantiation of Various Primitives . . . 79
3.8.1 Encryption scheme Enc . . . 79
3.8.2 Zero-knowledge Proof Schemes . . . 80
3.8.3 Asynchronous Reliable Broadcast (r-broadcast) Using Transferable Non-equivocation . . . 82
3.9 Analysis of the AMPC Protocol of [32] . . . 83

4 Privacy Specification: Precedent-based Reasoning 87
4.1 Ingredients . . . 89
4.2 Defining The PriCL Framework . . . 92
4.2.1 Introducing Cases . . . 92
4.2.2 Combining Cases to Case Law Databases . . . 95
4.2.3 Deriving Legal Consequences: Deducibility and Permissibility . . . 99
4.2.4 General Properties of Case Law Databases . . . 104
4.2.5 Privacy Cases and Norms . . . 105
4.3 Reasoning Tasks . . . 107
4.4 Logic Selection . . . 114
4.5 Concluding PriCL . . . 116

A Appendix 127
A.1 Postponed Soundness Proof Details . . . 127
A.1.1 Proof of Claim 1 . . . 150
A.1.2 Proof of Claim 4 . . . 153

Chapter 1

Introduction

Privacy is a particularly delicate topic with a tremendous socio-economic role nowadays. Frequent data breaches or data misuses created a social awareness and demand for more privacy online. However, the current state of cybersecurity also requires a mind-shift towards a more accountable Internet. These seemingly contrasting requirements create a huge demand for privacy enhancing technologies that allow certain guarantees to hold even though the user's privacy is protected.

Privacy enhancing technologies could ensure that a user can only access certain systems if he is authorized but without revealing her identity. Such technologies could also enable users to share their genomic data for scientific purposes without fully leaking it. And last but not least, privacy enhancing technologies can be used to ensure compliance with regulations. The number of applications is enormous and continuously growing. The reason for the growth of applications is the growth of cyberphysical systems through trends such as the Internet of Things, smart homes or autonomous driving. All these trends introduce further sensors into our lives which are supposed to bring more comfort. However, these sensoric solutions also threaten the individual privacy. For example, in a case of speeding, car manufacturers could provide governmental authorities with data of speed and GPS on a fine granular interval.

In this work, we identified three key areas for the future development and improved of privacy enhancing technologies. Within each key area, we improve the state of the art towards a more privacy friendly use. These areas are protocol verification, protocol implementations as well as the specification of privacy.

Protocol verification The abstraction of cryptographic operations by term algebras, called symbolic models, is essential in almost all tool-supported methods for analyzing security protocols. Significant progress was made in proving that symbolic models offering basic cryptographic operations such as encryption and digital signatures can be sound with respect to actual cryptographic realizations and security definitions. Even abstractions of sophisticated modern cryptographic primitives such as zero-knowledge (ZK) proofs were shown to have a computationally sound cryptographic realization, but only in ad-hoc formalism and at the cost of placing strong assumptions on the underlying cryptography, which leaves only highly inefficient

realizations.

In this work, we make two contributions to this problem space. First, we identify weaker cryptographic assumptions that we show to be sufficient for computational soundness of symbolic ZK proofs. These weaker assumptions are fulfilled by existing efficient ZK schemes as well as generic ZK constructions. Second, we conduct all computational soundness proofs in CoSP, a recent framework that allows for casting computational soundness proofs in a modular manner, independent of the underlying symbolic calculi. Moreover, all computational soundness proofs conducted in CoSP automatically come with mechanized proof support through an embedding of the applied π-calculus. This result is presented in chapter 2.

Protocol implementation Verifiable secret sharing (VSS) and secure multiparty computation (MPC) are the fundamental problems in secure distributed computing. It is well known that in the computational setting, asynchronous VSS (AVSS) and asynchronous MPC (AMPC) among n parties can tolerate up to $t < n/3$ active faults. Recently, assuming a synchronous broadcast round, Beerliová-Trubíniová, Hirt and Nielsen (PODC'10) improved the resiliency bound for AMPC by presenting a protocol with $t < n/2$. Nevertheless, their requirement of one synchronous broadcast round is non-trivial and possibly not realizable in some application scenarios.

In this work, we observe that it is possible to improve the resiliency bound for both AVSS and AMPC to handle $t < n/2$ active faults, without making any synchrony assumption, using *non-equivocation*. Non-equivocation is a mechanism to restrict a corrupted party from making conflicting statements to different (honest) parties, and it can be implemented using a trusted counter, realizable with trusted hardware modules readily available in commodity computers and smartphone devices. In particular, we present an AVSS protocol and an AMPC protocol in a (completely) asynchronous setting, tolerating $t < n/2$ faults, using non-equivocation. Apart from providing better resilience, our protocols are also efficient: specifically, our AMPC protocol provides a gain of $\Theta(n)$ in the communication complexity (per multiplication gate) over the AMPC protocol by Beerliová-Trubíniová et al. This result is presented in chapter 3.

Protocol specification We introduce PriCL: the first framework for expressing and automatically reasoning about privacy case law by means of precedent. PriCL is parametric in an underlying logic for expressing world properties, and provides support for court decisions, their justification, the circumstances in which the justification applies as well as court hierarchies. Moreover, the framework offers a tight connection between privacy case law and the notion of norms that underlies existing rule-based privacy research. In terms of automation, we identify the major reasoning tasks for privacy cases such as deducing legal permissions or extracting norms. For solving these tasks, we provide generic algorithms that have particularly efficient realizations within an expressive underlying logic. Finally, we derive a definition of deducibility based on legal concepts and subsequently propose an equivalent characterization in terms of logic satisfiability. This result is presented in chapter 4.

Chapter 2

Privacy Protocol Verification: Zero-Knowledge Proofs in CoSP

Proofs of security protocols are known to be error-prone and, owing to the distributed-system aspects of multiple interleaved protocol runs, awkward for humans to make. Hence work towards the automation of such proofs started soon after the first protocols were developed. From the start, the actual cryptographic operations in such proofs were idealized into so-called symbolic models, following [64, 67, 99], e.g., see [87, 111, 2, 94, 26]. This idealization simplifies proof construction by freeing proofs from cryptographic details such as computational restrictions, probabilistic behavior, and error probabilities. It was not at all clear whether symbolic models are a sound abstraction from real cryptography with its computational security definitions. Existing work has largely bridged this gap for symbolic models offering the core cryptographic operations such as encryption and digital signatures, e.g., see [3, 20, 91, 100, 57, 47].

While symbolic models traditionally comprised only basic cryptographic operations, recent work has started to extend them to more sophisticated primitives with unique security features that go far beyond the traditional goal of cryptography to solely offer secrecy and authenticity of communication. Zero-knowledge (ZK) proofs[1] constitute arguably the most prominent such primitive.[2] This primitive's unique security features, combined with the recent advent of efficient cryptographic implementations of this primitive for special classes of problems, have paved the way for its deployment in modern applications. For instance, ZK proofs can guarantee authentication yet preserve the anonymity of protocol participants, as in the Civitas electronic voting protocol [53] or the Pseudo Trust protocol [95], or they can prove the reception of a certificate from a trusted server without revealing the actual content, as in the Direct

[1] A zero-knowledge proof [73] consists of a message or a sequence of messages that combines two seemingly contradictory properties: First, it constitutes a proof of a statement x (e.g, $x =$ "the message within this ciphertext begins with 0") that cannot be forged, i.e., it is impossible, or at least computationally infeasible, to produce a zero-knowledge proof of a wrong statement. Second, a zero-knowledge proof does not reveal any information besides the bare fact that x constitutes a valid statement.

[2] Examples of other primitives studied in the symbolic setting are blind-signatures (e.g., in [88]), Diffie-Hellman-style exponentiation (e.g., in [1]), or private contract signatures (e.g., in [83]).

Anonymous Attestation (DAA) protocol [41]. More recently, ZK proofs have been used to develop novel schemes for anonymous webs of trust [17] as well as privacy-aware proof-carrying authorization [96].

A symbolic abstraction of (non-interactive) ZK proofs has been put forward in [19]. The proposed abstraction is suitable for mechanized proofs [19, 15] and was already successfully used to produce the first fully mechanized proof of central properties of the DAA protocol. A computational soundness result for such symbolic ZK proofs has recently been achieved as well [21]. However, this work imposes strong assumptions on the underlying cryptographic implementation of zero-knowledge proofs: Among other properties, the zero-knowledge proof is required to satisfy the notion of extraction zero-knowledge; so far, only one (inefficient) scheme is known that fulfills this notion [76]. Thus the vast number of recently proposed, far more efficient zero-knowledge schemes, and particularly those schemes that stem from generic ZK constructions, are not comprised by this result. Hence they do not serve as sound instantiations of symbolic zero-knowledge proofs, leaving all actually deployed ZK protocols without any computational soundness guarantee. In addition, the result in [21] casts symbolic ZK proofs within an ad-hoc formalism that is not accessible to existing formal proof tools.

The contribution of this thesis to the verification problem space is the following:

- First, we identify weaker cryptographic assumptions that we show to be sufficient for obtaining a computational soundness result for symbolic ZK proofs. Essentially, we show that the strong notion of extraction zero-knowledge required in [21] can be replaced by the weaker notion of simulation-sound extractability. In contrast to extraction zero-knowledge, simulation-sound extractability constitutes an established property that many existing cryptographic constructions satisfy. In particular, there exist generic constructions for transforming any non-interactive ZK proof into a ZK proof that satisfies simulation-sound extractability (and the remaining properties that we impose for computational soundness) [107], as well as several efficient schemes that are known to satisfy simulation-sound extractability (and the remaining properties), e.g., [93, 75, 108]. Thus requiring simulation-sound extractability instead of extraction zero-knowledge greatly extends the pool of cryptographic constructions for ZK proofs that constitute sound implementations, and it for the first time enables the computationally sound deployment of efficient ZK realizations.

- Second, we conduct all computational soundness proofs in CoSP [14], a recent framework that allows for casting computational soundness proofs in a conceptually modular and generic way: proving x cryptographic primitives sound for y calculi only requires $x + y$ proofs (instead of $x \cdot y$ proofs without this framework), and the process of embedding calculi is conceptually decoupled from computational soundness proofs of cryptographic primitives. In particular, computational soundness proofs conducted in CoSP are automatically valid for the applied π-calculus, and hence accessible to existing mechanized verification techniques.

  The conduction in CoSP has the drawback that the computational soundness is shown

for trace properties. However, trace properties are sufficient to verify weak anonymity. Consequently, we can show central properties of the DAA protocol.

Outline. In this chapter, we prove computational soundness for a symbolic model containing modern cryptographic building blocks, specifically zero-knowledge proofs. We first introduce the symbolic and computational models that we relate with the soundness proof. Thereafter, we introduce the notion of zero-knowledge that we require for the proof. Since computational soundness proofs are lengthy in nature, we first give insights in the proof strategy before detailing the proof. Repetitive parts of the proof are postponed to the appendix for the sake of readability. We also give some application for soundness result as well as insights in the necessity of requirements for the ZK proofs.

2.1 CoSP framework for computational soundness proofs

Computational soundness needs to link three components mathematically. First, we need a symbolic model to specify the protocols. Second, we need a symbolic protocol execution as well as computational execution. The third necessary component is the specification of properties that executions can have. The soundness result then states that any property holds for a symbolic execution if and only if it holds for the computational one.

CoSP gives an answer for these basic needs in a general way. The advantage of such a generic framework is that the symbolic models can be embedded in the actual representation of verification tools, i.e., any proof for a specific model can then be leveraged by the verification results of all tools for which an embedding exists. As a consequence, the number of required proofs is reduced to $x + y$ for x symbolic models and y verification languages instead of $x \cdot y$ proofs following the naive approach to the problem. This has been done for several verification languages such as the applied π calculus [14] or type-systems verification via F# [15].

2.1.1 Symbolic Model

We start with the symbolic model. The purpose of the symbolic model is to have an abstract representation of the protocol that is easy to design for the human user and in addition amenable for automated verification tools.

The symbolic model describes four things. First, describes how abstract terms can be constructed, e.g., $\mathrm{enc}(\mathrm{ek}(N), m, N2)$ for the encryption of message m using encryption key $\mathrm{ek}(N)$ and randomness $N2$. This is modeled by a set of constructors $\mathbf{C}$ that can be applied to other terms creating new terms. In the example, ek and enc are such constructors.

Second, the symbolic model describes which terms consisting of constructors applied to each other are valid. Following the example, the first argument of enc needs to be an encryption key, to lead to a valid ciphertext. This set of valid objects is called terms $\mathbf{T}$.

Third, it describes how these terms can be transformed into other terms, in particular removing constructors to extract a subterm is such an operation. In the example above, decryption can

be modeled like this, e.g., dec is a function that takes a decryption key $\mathrm{dk}(N)$ and a ciphertext $\mathrm{enc}(\mathrm{ek}(N), m, N2)$ and returns m. Note that the N for $\mathrm{dk}(N)$ and $\mathrm{ek}(N)$ need to match for the operation. Summarized, a destructor is a function that transforms a set terms to a new term.

Finally, the symbolic model describes what an attacker can derive from terms. This derivation allows to model attacker capabilities that go beyond the constructors and destructors that the honest protocol user can use. For example, we could allow an attacker to derive the randomness of a decryption key $\mathrm{dk}(N)$ symbolically although there is not necessarily a real-world implementation for this functionality. Consequently, in order to get good verification results, the derivation of the attacker should be as limited as the computational soundness proof allows.

These aspects lead to the following definition of a symbolic model, consisting of constructors, destructors, and deduction relations.

Definition 2.1.1 (Symbolic model)**.** *A* symbolic model *is a 5-tuple* $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$ *that consists of a set of constructors* $\mathbf{C}$*, a set of nonces* $\mathbf{N}$*, a message type* $\mathbf{T}$ *over* $\mathbf{C}$ *and* $\mathbf{N}$ *with* $\mathbf{N} \subseteq \mathbf{T}$*, a set of destructors* $\mathbf{D}$ *over* $\mathbf{T}$*, and a deduction relation* $\vdash$ *over* $\mathbf{T}$ *where constructor, nonce, message type, destructor and deduction relation are defined as follows:*

- *A* constructor C *is a symbol with an arity. We write* $C/n \in \mathbf{C}$ *to say that the set* $\mathbf{C}$ *contains a constructor* C *with arity* n*.*
- *A* nonce N *is a symbol with zero arity.*
- *A* message type $\mathbf{T}$ *over* $\mathbf{C}$ *and* $\mathbf{N}$ *is a set of terms over constructors* $\mathbf{C}$ *and nonces* $\mathbf{N}$*.*
- *A* destructor D *of arity* n *over a message type* $\mathbf{T}$ *is a partial map* $\mathbf{T}^n \to \mathbf{T}$*. If* D *is undefined on* $\underline{t} := (t_1, \ldots, t_n)$*, we write* $D(\underline{t}) = \bot$*.*
- *A* deduction relation $\vdash$ *over a message type* $\mathbf{T}$ *is a relation between* $2^{\mathbf{T}}$ *and* $\mathbf{T}$*.*

For the sake of simpler, unified notation, we define a function eval_F for F being a constructor, nonce or destructor. If F is a constructor or nonce, we write $\mathrm{eval}_F(t_1, \ldots, t_n) := F(\underline{t})$ if $F(\underline{t}) \in \mathbf{T}$ and $\mathrm{eval}_F(\underline{t}) := \bot$ otherwise. If F is a destructor, we write $\mathrm{eval}_F(\underline{t}) := F(\underline{t})$ if $F(\underline{t}) \neq \bot$ and $\mathrm{eval}_F(\underline{t}) := \bot$ otherwise.

The previous definition specified how the symbolic world looks like and how to move within it. In the next step, we need to define the actual object of interest: the protocol. In addition to transforming terms, the protocol has to take branches which might depend on specific input and the protocol has to specify what is communicated at which point in time.

The start of every protocol is a unique point and the progressing protocol might branch into different states. Thus, the protocol is modelled as an annotated tree. In details, the following definition specifies CoSP protocols:

Definition 2.1.2 (CoSP protocol)**.** *A CoSP protocol* Π_s *is a tree with a distinguished root and labels on edges and nodes. Each node has a unique identifier* N *and one of the following types:*

- *Computation nodes are annotated with a constructor, nonce, or destructor F/n together with the identifiers of n (not necessarily distinct) nodes. Computation nodes have exactly two successors; the corresponding edges are labeled with yes and no, respectively.*

- *Output nodes are annotated with the identifier of one node. An output node has exactly one successor.*

- *Input nodes have no further annotation.An input node has exactly one successor.*

- *Control nodes are annotated with a bitstring l. A control node has at least one and up to countably many successors annotated with distinct bitstrings $l' \in \{0,1\}^*$. (We call l the out-metadata and l' the in-metadata.)*

- *Nondeterministic nodes have no further annotation. Nondeterministic nodes have at least one and at most finitely many successors; the corresponding edges are labeled with distinct bitstrings.*

The computation node models the usage of the symbolic model, the output and input nodes model communication and the control nodes model decisions. The nondeterministic nodes can be used to model probabilistic decisions in protocol executions. Thus, assigning each nondeterministic node a probability distribution over its successors yields the notion of a *probabilistic CoSP protocol*. As the notion of efficiency is crucial for cryptographic building blocks and in particular usually the limitation of the attacker, we define efficient protocol executions for probabilistic CoSP protocols. In particular, this ensures that a common attacker can execute the protocol.

Definition 2.1.3 (Efficient protocol)**.** *We call a probabilistic CoSP protocol efficient if:*

- *There is a polynomial p such that for any node N, the length of the identifier of N is bounded by $p(m)$ where m is the length (including the total length of the edge-labels) of the path from the root to N.*

- *There is a deterministic polynomial-time algorithm that, given the identifiers of all nodes and the edge labels on the path to a node N, computes the label of N.*

The last piece missing on the symbolic side with respect to computational soundness are the properties that we want to verify. These are trace properties, i.e., possible execution traces that occur in one model can also occur in the other. Consequently, traces are part of executions and we need to define what it means to symbolically execute a protocol.

The definition of symbolic execution is a natural consequence of a CoSP protocol and symbolic model definitions. It resembles the path taken in the protocol tree Π_s. In the definition, we specify every position in the path by the attacker's current knowledge S, the identifier of the current node ν, as well as the history of received and computed messages via a partial function f. Formally, this leads to the following definition:

Definition 2.1.4 (Symbolic execution)**.** *Let a symbolic model* $(\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$ *and a CoSP protocol* Π_s *be given. A full trace is a (finite) list of tuples* (S_i, ν_i, f_i) *such that the following conditions hold:*

- *Correct start:* $S_1 = \emptyset, \nu_1$ *is the root of* Π_s, f_1 *is a totally undefined partial function mapping node identifiers to terms.*
- *Valid transition: For every two consecutive tuples* (S, ν, f) *and* (S', ν', f') *in the list, let* $\underline{\tilde{\nu}}$ *be the node identifiers in the annotation of* ν *and define* $\underline{\tilde{t}}$ *through* $\tilde{t}_j := f(\tilde{\nu}_j)$*. We have:*
  - *If* ν *is a computation node with constructor, destructor or nonce* F*, then* $S' = S$*. If* $m := \mathrm{eval}_F(\underline{\tilde{t}}) \neq \bot$*,* ν' *is the yes-successor of* ν *in* Π_s*, and* $f' = f(\nu := m)$*. If* $m = \bot$*, then* ν' *is the no-successor of* ν *and* $f' = f$*.*
  - *If* ν *is an input node, then* $S' = S$ *and* ν' *is the successor of* ν *in* Π_s *and there exists an* m *with* $S \vdash m$ *and* $f' = f(m := m)$*.*
  - *If* ν *is an output node, then* $S' = S \cup \{\tilde{t}_1\}$*,* ν *is the successor of* ν *in* Π_s *and* $f' = f$*.*
  - *If* ν *is a control node or a nondeterministic node, then* ν' *is a successor of* ν *and* $f' = f$ *and* $S' = S$*.*

A list of node identifiers (ν_i) *is a node trace if there is a full node trace with these node identifiers.*

2.1.2 Computational Model

In this subsection, we define the counterpart of the symbolic execution, the computational implementation. In the overall picture, this is the execution that we want to analyse and secure by leveraging analysis results from the symbolic model we defined.

The computational execution for a protocol needs to shift the symbolic parts to the computational world, i.e., from applying constructors to nonces to bitstrings. In order to do so, we define the computational counterpart of the symbolic model and apply the definition afterwards to the definition of the symbolic execution, leading to a computational execution.

Definition 2.1.5 (Computational implementation)**.** *Let a symbolic model* $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$ *be given. A computational implementation* A *is a family of functions* $A = (A_x)_{x \in \mathbf{C} \cup \mathbf{D} \cup \mathbf{N}}$ *such that* A_F *for* $F/n \in \mathbf{C} \cup \mathbf{D}$ *is a partial deterministic function* $\mathbb{N} \times (\{0,1\}^*)^n \to \{0,1\}^*$*, and* A_N *for* $N \in \mathbf{N}$ *is a total probabilistic function with domain* $\mathbb{N}$ *and range* $\{0,1\}^*$ *(i.e. it specifies a probability distribution on bitstrings that depends on its argument). The first argument of* A_F *and* A_N *represents the security parameter. All functions* A_F *have to be computable in deterministic polynomial-time, and all* A_N *have to be computable in probabilistic polynomial-time.*

Note that the definition only allows A_N to be probabilistic for a nonce N. The consequence is that randomness has to be modelled explicitly. The advantage is that, for example, two ciphertext of the same string can be compared symbolically in a meaningful way,i.e., they are equal if also the same randomness is used and unequal otherwise.

The computational execution essentially follows the same rules as the symbolic one, except that the function f stores bitstrings corresponding to nodes in the computational case, and that the implementations of symbolic constructors and destructors are used.

Definition 2.1.6 (Computational execution)**.** *Let a symbolic model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$, a computational implementation A of $\mathbf{M}$, and a probabilistic CoSP protocol Π_p be given. Let a probabilistic polynomial-time interactive machine E (the adversary) be given, and let p be a polynomial. We define a probability distribution $\mathrm{Nodes}^p_{\mathbf{M},A,\Pi_p,E}(k)$, the computational node trace, on (finite) lists of node identifiers (ν_i) according to the following probabilistic algorithm (both the algorithm and the adversary run on input k):*

- *Initial state: $\nu_1 := \nu$ is the root of Π_p. Let f be the empty partial function from node identifiers to bitstrings, and let n be an initially empty partial function from $\mathbf{N}$ to bitstrings.*
- *For $i = 2, 3, \ldots$ do:*
  - *Let $\underline{\tilde{\nu}}$ be the node identifiers in the annotation of ν. $\tilde{m}_j := f(\tilde{\nu}_j)$.*
  - *Proceed according to the type of node ν:*
    - *If ν is a computation node with nonce $N \in \mathbf{N}$: Let $m' := n(N)$ if $n(N) \neq \bot$ and sample m' according to $A_N(k)$ otherwise. Let ν' be the yes-successor of ν, $f' := f(\nu := m')$ and $n' := n(N := m')$.*
    - *If ν is a computation node with constructor or destructor F, then $m' := A_F(k, \tilde{m})$. If $m' \neq \bot$ then let ν' be the yes-successor of ν, otherwise ν' is the no-successor of ν. Let $f' := f(\nu := m')$ and $n' = n$.*
    - *If ν is an input node, ask for a bitstring m from E. Abort the loop if E halts. Let ν' be the successor of ν, $n' := n$, $f' := f(\nu := m)$.*
    - *If ν is an output node, send $\tilde{m}_1$ to E. Abort if E halts. Let ν' be the successor of ν, $f' = f$ and $n' = n$.*
    - *If ν is a control node, labelled with out-metadata l, send l to E. Abort if E halts. Upon receiving answer l', let ν' be the successor of ν along the edge labelled with l' (or the lexicographic smallest edge if there is no edge labelled with l'). Let $n' := n$ and $f' = f$.*
    - *If ν is a nondeterministic node, let $\mathcal{D}$ be the probability distribution on the label of ν. Pick ν' according to $\mathcal{D}$ and let $n' := n$, $f' := f$.*
  - *Let $\nu = \nu'$, $f = f'$ and $n = n'$.*
  - *Let $\nu_i := \nu$.*
  - *Let len be the number of nodes from the root to ν plus the total length of all bitstrings in the range of f. If $len > p(k)$, stop.*

The definition closely follows the symbolic execution. All differences come from computational artifacts such as efficiency and probabilistic sampling.

2.1.3 Computational soundness.

In this subsection, we formally define what computational soundness with respect to trace properties is. Afterwards, we present a general result about the CoSP framework that already lines out the proof strategy for the computational soundness result. We start by defining trace properties.

Definition 2.1.7 (Trace property)**.** *A trace property $\mathcal{P}$ is an efficiently decidable and prefix-closed set of (finite) lists of node identifiers.*

Let $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$ be a symbolic model and Π_s a CoSP protocol. Then Π_s symbolically satisfies a trace property $\mathcal{P}$ in $\mathbf{M}$ iff every node trace of Π_s is contained in $\mathcal{P}$. Let A be a computational implementation of $\mathbf{M}$ and let Π_p be a probabilistic CoSP protocol. Then (Π_p, A) computationally satisfies a trace property $\mathcal{P}$ in $\mathbf{M}$ iff for all probabilistic polynomial-time interactive machines E and all polynomials p, the probability is overwhelming that $\mathrm{Nodes}^p_{\mathbf{M},A,\Pi_p,E}(k) \in \mathcal{P}$.

The requirement of being efficiently decidable has a rather practical purpose. If the property is not decidable, it does not make any sense trying to verify the property using a verification tool. If the property is not efficiently decidable, the attacker might not be able to decide whether he succeeded.

Definition 2.1.8 (Computational soundness)**.** *A computational implementation A of a symbolic model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$ is computationally sound for a class P of CoSP protocols iff for every trace property $\mathcal{P}$ and for every efficient probabilistic CoSP protocol Π_p, we have that (Π_p, A) computationally satisfies $\mathcal{P}$ whenever the corresponding CoSP protocol Π_s of Π_p symbolically satisfies $\mathcal{P}$ and $\Pi_s \in P$.*

In the remainder of this section, we outline the common proof strategy that we will also use: a simulation based proof. The high-level idea is to assume we do not have soundness. Then there is a trace in the computational world that cannot be run symbolically. However, the constructed simulator can translate between the symbolic and computational world such that trace properties are preserved. This contradiction proves that the assumption was wrong, and hence we get computational soundness. We start by defining the simulator.

Definition 2.1.9 (Simulator)**.** *A simulator is an interactive machine* Sim *that satisfies the following syntactic requirements:*

- *When activated without input, it replies with a term $m \in \mathbf{T}$.*
- *When activated with some $t \in \mathbf{T}$, it replies with an empty output.*
- *When activated with a bitstring label l it answers with some bitstring.*
- *When activated with $(\mathbf{info}, \nu, t)$ where ν is a node identifier and $t \in \mathbf{T}$, it replies with $(\mathbf{proceed})$.*

- *At any point (especially instead of replying), it may terminate.*

The simulator combines aspects from the computational executions adversial environment E with the symbolic executions by using terms from the symbolic model. However, neither the symbolic execution nor the computational execution can be used in combination with the simulator. Hence, we need to define a hybrid combination to execute the simulator.

Definition 2.1.10 (Hybrid execution)**.** *Let Π_p be a probabilistic CoSP protocol, and let* Sim *be a simulator. We define a probability distribution* $\text{H-Trace}_{\mathbf{M},\Pi_p,\text{Sim}}(k)$ *on (finite) lists of tuples (S_i, ν_i, f_i) called the full hybrid trace according to the following probabilistic algorithm Π^C, run on input k, that interacts with* Sim*. (Π^C is called t he hybrid protocol machine associated with Π_p and internally runs a symbolic simulation of Π_p as follows:)*

- *Start: $S_1 := S := \emptyset, \nu_1 := \nu$ is the root of Π_p, and $f_1 := f$ is a totally undefined partial function mapping node identifiers to $\mathbf{T}$. Run Π_p on ν.*
- *Transition: For $i = 2, 3, \ldots$ do the following:*
  - *Let $\underline{\tilde{\nu}}$ be the node identifiers in the label of ν. Define $\underline{\tilde{t}}$ through $\tilde{t}_j := f(\tilde{\nu}_j)$.*
  - *Proceed depending on the type of ν.*
    - *If ν is a computation node with constructor destructor or nonce F, then let $m := F(\underline{\tilde{t}})$. If $m \neq \perp$, let ν' be the yes-successor and $f' := (\nu := m)$, otherwise let ν' be the no-successor and $f' := f$. Let $\nu := \nu'$ and let $f := f'$.*
    - *If ν is an output node, send $\tilde{t}_1$ to* Sim*, without handing over the control to* Sim*. Let ν' be the unique successor of ν and set $\nu := \nu'$.*
    - *If ν is an input node, hand control to* Sim *and wait to receive $m \in \mathbf{T}$ from it. Let $f' := f(\nu := m)$ and let ν' be the unique successor of ν. Set $\nu := \nu'$ and $f := f'$.*
    - *If ν is a control node labelled with out-metadata l, send l to* Sim*. Hand control to* Sim *and wait to receive a bitstring l' from* Sim*. Let ν; be the successor of ν along the edge labeled l', or the lexicographically smallest if there is no edge with label l'. Let $\nu := \nu'$.*
    - *If ν is a nondeterministic node, sample ν' according to the probability distribution specified in ν. Let $\nu := \nu'$.*
  - *Send $(\mathbf{info}, \nu, t)$ to* Sim*. When receiving an answer $(\mathbf{proceed})$ from* Sim*, continue.*
  - *If* Sim *has terminated, stop. Otherwise let $(S_i, \nu_i, f_i) := (S, \nu, f)$.*

The probability distribution of the (finite) list $\nu_1, \ldots$ produced by this algorithm we denote by $\text{H-Nodes}_{\mathbf{M},\Pi_p,\text{Sim}}(k)$*. We call this distribution the hybrid node trace.*

Analysing the differences between this hybrid execution and the other two definitions leads to sufficient criteria for computational soundness. Comparing the hybrid execution definition with the symbolic execution definition, the key difference is that there is no validity check for the

terms, i.e., the condition $S \vdash m$ is missing. If every message m from the simulator satisfies that criteria, it is called Dolev-Yao style (DY-style). In comparison to the computational execution, the Sim does not need to behave as the attacker at all. Thus, in addition to Dolev-Yao style, the simulator execution needs to be indistinguishable from the computational execution. In particular, indistinguishability is very common in cryptographic definitions as well.

The existence of a simulator that fulfills the two distinguished properties, DY-style and indistinguishability, has actually been shown sufficient for needs to fulfill to establish computational soundness. We formally define the two properties in the following definition and speak of a *good* simulator if both properties are fulfilled.

Definition 2.1.11 (Good simulator)**.** *A simulator* Sim *is* Dolev-Yao style *(short: DY) for* $\mathbf{M}$ *and* Π_p*, if with overwhelming probability the following holds: In an execution of* $\mathrm{Sim} + \Pi^C$*, for each* l*, let* $m_l \in \mathbf{T}$ *be the* l*-th term sent (during processing of one of* Π^C*'s input nodes) from* Sim *to* Π^C *in that execution. Let* $T_l \subset \mathbf{T}$ *be the set of all terms that* Sim *has received from* Π^C *(during processing of output nodes) prior to sending* m_l*. Then we have* $T_l \vdash m_l$*.*

A simulator Sim *is* indistinguishable *for* $\mathbf{M}$*,* Π_p*, an implementation* A*, an adversary* E*, and a polynomial* p*, if* $\mathrm{Nodes}^{p}_{\mathbf{M},A,\Pi_p,E}(k) \overset{C}{\approx} \text{H-Nodes}_{\mathbf{M},\Pi_p,\mathrm{Sim}}(k)$*, i.e., if the computational node trace and the hybrid node trace are computational indistinguishable.*

A simulator is good *if it is Dolev-Yao style and indistinguishable.*

Since the computational soundness result in this chapter makes use of this proof technique, we also quote the theorem that we leverage.

Theorem 2.1.1 (Good simulator implies soundness [14])**.** *Let* $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$ *be a symbolic model, let* P *be a class of CoSP protocols, and let* A *be a computational implementation of* $\mathbf{M}$*. Assume that for every efficient probabilistic CoSP protocol* Π_p *(whose corresponding CoSP protocol is in* P*), every probabilistic polynomial-time adversary* E*, and every polynomial* p*, there exists a good simulator for* $\mathbf{M}, \Pi_p, A, E$*, and* p*. Then* A *is computationally sound for protocols in* P*.*

2.2 Zero Knowledge Proofs

Zero-Knowledge (ZK) proofs are a cryptographic tool to mathematically prove a statement without revealing more information, for example a website might require an age verification. The zero-knowledge proof can be used to prove that the age is above 18 without revealing the birth date. This kind of mechanism is used in privacy protocols hiding the identity but proving eligibility, e.g., for e-voting protocols [53] or in cryptocurrencies such as ZCash in order to preserve anonymity.

These properties sound contradicting at first but let us clarify it with an example from the area of graph theory. Given two graphs G_1 and G_2 computing the isomorphism between them has quasipolynomial time and most practical algorithms have an exponential worst case. So assume Alice knows a graph isomorphism and wants to prove to Bob that the graphs G_1 and G_2

are isomorphic without giving him the isomorphism. In order to do so, they run a three round protocol:

1. Alice samples a random isomorph graph G' for G_1 and sends G' to Bob.
2. Bob flips a coin and sends 1 for head and 2 for tails to Alice.
3. When receiving 1 Alice sends the isomorphism between G' and G_1 to Bob, when receiving 2 she sends the isomorphism between G' and G_2 to Bob (that one she can compute since she has the isomorphism between G_1 and G_2).

Running this protocol Alice can always succeed if she has an isomorphism between G_1 and G_2. If she has not such an isomorphism Bob catches her with probability $\frac{1}{2}$ since she cannot send both isomorphisms in step three. However, Bob does not learn the isomorphism between G_1 and G_2. Note that the probability of $\frac{1}{2}$ can be amplified by repeating the experiment.

The given example is an interactive zero-knowledge proof, i.e., Bob needs to interact with Alice in order to make the proof work. If Alice knew upfront which coin-flip Bob does she could easily cheat by computing G' as a random isomorphism that she needs to reveal in step three.

In contrast to these interactive zero-knowledge proofs, there are also non-interactive zero-knowledge proofs which we consider in this work. The advantage of the non-interactive proofs is that they are transferable, i.e., if Alice creates a non-interactive ZK proof and sends it to Bob, he can also forward it to Charlie who can verify the truth of the statement in the proof.

There also is a transformation for interactive ZK proofs to non-interactive ones in the random oracle model, i.e., instead of using randomness given by Bob, Alice uses a random oracle that given k graphs (where k is the number of repetitions Bob would have done) outputs a random bitstring of length k that Alice uses as coin-flips. Then she can send Bob the k graphs, and the corresponding isomorphisms of round three, he can then use the random oracle to verify the isomorphisms.

Non-interactive ZK proofs can further be distinguished by the property of malleability, i.e., by the question *Given a proof, can it be transformed into another proof?*. For example, having a proof for the statement A and a proof for B, can it be combined leading to a proof for $A \wedge B$. The proof technique shown in here has also been used in the scenario of controlled malleability[13], however, verification algorithms struggle with the logical connection of statements which makes the search space grow exponentially.

In [21], it was shown that for getting computational soundness of (non-interactive) zero-knowledge proofs, we need at least the following properties:[3] *Completeness* (if prover and verifier are honest, the proof is accepted), *extractability* (given a suitable trapdoor, one can get a witness out of a valid proof – this models the fact that the prover knows the witness), *zero-knowledge* (given a suitable trapdoor and a true statement x, a ZK-simulator can produce proofs without knowing a witness that are indistinguishable from normally generated proofs for

[3]It was not shown that these are the minimal properties, but it was shown that none of these properties can be dropped without suitable substitute.

x), *unpredictability* (two proofs are equal only with negligible probability), *length-regularity* (the length of a proof only depends on the length of statement and witness), and some variant of *non-malleability*. Furthermore, they required for convenience that the verification and the extraction algorithm are deterministic.

The variant of non-malleability chosen in [21] was the notion of *extraction zero-knowledge* which is a strong variant of extractability (we are aware of only one scheme in the literature that has this property [76]). They left it as an open problem whether weaker variants also lead to computational soundness. We answer this question positively. We use the weaker and more popular notion of *simulation-sound extractability*. In a nutshell, this notion guarantees that the adversary cannot produce proofs from which no witness can be extracted, even when given access to a ZK-simulator.

We actually need an even weaker property: *honest simulation-sound extractability*. Here the adversary may ask the ZK-simulator to produce a simulated proof for x if he knows a witness w for x.

Before we come to our definition, we need to specify one more detail about the ZK proofs: the statements to be proven. The statement we have seen in the example was for graph isomorphism and is very specific. In order to have a widely applicable result, this part needs to be generic. A property that the statements all have in common is that they have a secret part and a public part. Since the secret is used to prove a statement about the public part x, it is often called *witness* w. For a statement we use a relation $\mathcal{R}$ and the statement is true if and only if $(w, x) \in \mathcal{R}$.

Proving computational soundness requires at this point to allow the attacker potentially more than the honest parties, thus we have distinguished two relations $R^{\mathrm{sym}}_{\mathrm{adv}}$ and $R^{\mathrm{sym}}_{\mathrm{honest}}$, the first modeling what the adversary is able to do, the second modeling what honest participants are allowed to do. Similarly, our definition of *weakly symbolically-sound zero-knowledge proof* distinguishes two relations $R^{\mathrm{comp}}_{\mathrm{adv}} \supseteq R^{\mathrm{comp}}_{\mathrm{honest}}$. ("Weakly" distinguishes our notion from that in [21] which requires extraction ZK.) All conditions assume that honest participants use $(x, w) \in R^{\mathrm{comp}}_{\mathrm{honest}}$. Hence, we call $R^{\mathrm{sym}}_{\mathrm{honest}}$ the *usage restriction*. Note that in the simplest case, we would have $R^{\mathrm{sym}}_{\mathrm{honest}} := R^{\mathrm{sym}}_{\mathrm{adv}}$.

In some cases, however, it may be advantageous to let $R^{\mathrm{sym}}_{\mathrm{honest}}$ be strictly smaller than $R^{\mathrm{sym}}_{\mathrm{adv}}$. This permits us to model a certain asymmetry in guarantees given by a zero-knowledge proof system: To honestly generate a valid proof, we need a witness with $(x, w) \in R^{\mathrm{sym}}_{\mathrm{honest}}$, but given a malicious prover, we only have the guarantee that the prover knows a witness with $(x, w) \in R^{\mathrm{sym}}_{\mathrm{adv}}$.

Definition 2.2.1 (Weakly symbolically-sound ZK proofs)**.** *A weakly symbolically-sound zero-knowledge proof system for relations $R^{\mathrm{comp}}_{\mathrm{honest}}, R^{\mathrm{comp}}_{\mathrm{adv}}$ is a tuple of polynomial-time algorithms $(\mathbf{K}, \mathbf{P}, \mathbf{V})$ such that there exist polynomial-time algorithms $(\mathbf{E}, \mathbf{S})$ and the following properties hold:*

- Completeness: *Let a polynomial-time adversary $\mathcal{A}$ be given. Let* $(\mathrm{crs}, \mathrm{simtd}, \mathrm{extd}) \leftarrow \mathbf{K}(1^\eta)$. *Let* $(x, w) \leftarrow \mathcal{A}(1^\eta, \mathrm{crs})$. *Let* $\mathrm{proof} \leftarrow \mathbf{P}(x, w, \mathrm{crs})$. *Then with overwhelming probability in η, it holds* $(x, w) \notin R^{\mathrm{comp}}_{\mathrm{adv}}$ *or* $\mathbf{V}(x, \mathrm{proof}, \mathrm{crs}) = 1$.

- Zero-Knowledge: *Fix a polynomial-time oracle adversary $\mathcal{A}$. For given* $\mathrm{crs}, \mathrm{simtd}$*, let* $\mathcal{O}_{\mathbf{P}}(x,w) := \mathbf{P}(x,w,\mathrm{crs})$ *if* $(x,w) \in R^{\mathrm{comp}}_{\mathrm{honest}}$ *and* $\mathcal{O}_{\mathbf{P}}(x,w) := \perp$ *otherwise, and let* $\mathcal{O}_{\mathbf{S}}(x,w) := \mathbf{S}(x,\mathrm{crs},\mathrm{simtd})$ *if* $(x,w) \in R^{\mathrm{comp}}_{\mathrm{honest}}$ *and* $\mathcal{O}_{\mathbf{S}}(x,w) := \perp$ *otherwise. Then*

$$|\Pr[\mathcal{A}^{\mathcal{O}_{\mathbf{P}}}(1^\eta,\mathrm{crs}) = 1 : (\mathrm{crs},\dots) \leftarrow \mathbf{K}(1^\eta)] - \Pr[\mathcal{A}^{\mathcal{O}_{\mathbf{S}}}(1^\eta,\mathrm{crs}) = 1 : (\mathrm{crs},\dots) \leftarrow \mathbf{K}(1^\eta)]|$$

  is negligible in η*.*

- Honest simulation-sound extractability: *Let a polynomial-time oracle adversary $\mathcal{A}$ be given. Let* $(\mathrm{crs},\mathrm{simtd},\mathrm{extd}) \leftarrow \mathbf{K}(1^\eta)$*. Let* $\mathcal{O}(x,w) := \mathbf{S}(x,\mathrm{crs},\mathrm{simtd})$ *if* $(x,w) \in R^{\mathrm{comp}}_{\mathrm{honest}}$ *and* $\perp$ *otherwise. Let* $(x,\mathrm{proof}) \leftarrow \mathcal{A}^{\mathcal{O}}(1^\eta,\mathrm{crs})$*. Let* $w \leftarrow \mathbf{E}(x,\mathrm{proof},\mathrm{extd})$*. Then with overwhelming probability, if* $\mathbf{V}(x,\mathrm{proof},\mathrm{crs}) = 1$ *and* proof *was not output by* $\mathcal{O}$ *then* $(x,w) \in R^{\mathrm{comp}}_{\mathrm{adv}}$*.*

- Unpredictability: *Let a polynomial-time adversary $\mathcal{A}$ be given. Let* $(\mathrm{crs},\mathrm{simtd},\mathrm{extd}) \leftarrow \mathbf{K}(1^\eta)$*. Let* $(x,w,\mathrm{proof}') \leftarrow \mathcal{A}(1^\eta,\mathrm{crs},\mathrm{simtd},\mathrm{extd})$*. Then with overwhelming probability, it holds* $\mathrm{proof}' \neq \mathbf{P}(x,w,\mathrm{crs})$ *or* $(x,w) \notin R^{\mathrm{comp}}_{\mathrm{honest}}$*.*

- Length-regularity: *Let two witnesses w and w', and statements x and x' be given such that* $|x| = |x'|$*, and* $|w| = |w'|$*. Let* $(\mathrm{crs},\mathrm{simtd},\mathrm{extd}) \leftarrow \mathbf{K}(1^\eta)$*. Then let* $\mathrm{proof} \leftarrow \mathbf{P}(x,w,\mathrm{crs})$ *and* $\mathrm{proof}' \leftarrow \mathbf{P}(x',w',\mathrm{crs})$*. Then we get* $|\mathrm{proof}| = |\mathrm{proof}'|$ *with probability* 1*.*

- Deterministic verification and extraction: *The algorithms V and E are deterministic.*

(We do not explicitly list soundness because it is implied by honest simulation-sound extractability.) $\diamond$

We stress that using the the construction in [107] on a length-regular and extractable NIZK leads to weakly symbolically-sound ZK proof system.

Constructions of weakly symbolically-sound zero-knowledge proof schemes

In this section we give a construction of zero-knowledge schemes satisfying the definition that we require for the soundness result. The given construction is generic in the sense that it takes a zero-knowledge scheme with weaker properties and transforms it. In order to do so, we use a known transformation [107] and show that if we add properties to the base scheme, then the out-coming scheme satisfies our definition. The construction is done by repeating the proof, which makes the out-coming proof scheme less efficient.

Construction [107]. The construction yields weakly symbolically-sound zero-knowledge proofs given any non-interactive zero-knowledge proof which is length-regular and extractable.

This is summarized in the following theorem:

Theorem 2.2.1. *Let π be a length regular, extractable non-interactive zero-knowledge proof system and assume that one way functions exist. Then the construction in [107] leads to a weakly symbolically-sound zero-knowledge proof system* Π.

In paper [107], they proved that the construction satisfies all properties listed in definition 2.2.1 except honest simulation-extractability and length-regularity. However, we show that it also satisfies these properties.

The construction basically uses polynomially many zero-knowledge proofs to construct a single one, but the number of proofs used in the combined proof is always the same. So length-regular follows immediately. They show unpredictability and simulation-soundness. Simulation-soundness is shown by a reduction to the soundness property of the underlying zero-knowledge proof system. The same way one can reduce the simulation-extractability property to extractability.

The construction in [107] uses a strong one-time signature scheme $(\mathbf{K}, \mathrm{sig}, \mathrm{verify}_{\mathrm{sig}})$ which is strong existential unforgeable when it is used only once and length regular. Denote $q(k)$ the length of the verification key.

In addition there is assumed to be an efficiently computable function $g : \{0,1\}^{q(k)} \to 2^{q'(k)}$ that maps verification keys to subsets of $\{1, \ldots, q'(k)\}$. Let $t(k)$ be a polynomial upper bound of the proof occurring in a protocol execution, $l(k) = q(k) \cdot t(k)$, and $q'(k) = l(k)^2$. Then holds for any set $m^1, \ldots, m^{t(k)}$ different from m that $|g(m) \setminus \bigcup_{i=1}^{t(k)} g(m^i| \geq \frac{l(k)}{2}$. Details on the construction of such functions can be found in [107].

We show that the construction given in [107] satisfies the properties. The construction is the following:

Reference String Let σ be a reference string of the proof system π. Then the reference string of Π is $\Sigma = \sigma_0 \circ \sigma_1 \circ \ldots \circ \sigma_{q'(k)}$. The same way is the simulation trapdoor the concatenation of the simulation trapdoors of π, and the extraction trapdoor the concatenation of extraction trapdoors of π.

Prover $\mathbf{P}_\Pi(x, w\Sigma)$:

(1) Run $\mathbf{K}(1^\nu)$ to obtain a key pair $(\mathrm{vk}, \mathrm{sk})$ for the one-time signature scheme.

(2) For each i in the set $g(\mathrm{vk})$ prove $p_i = \mathbf{P}_\pi(x, w, \sigma_i)$. For $i \notin g(\mathrm{vk})$, define $p_i := \epsilon$, i.e. the empty string.

(3) Let $P := p_1 \circ \ldots \circ p_{q'(k)}$.

(4) Output $(\mathrm{vk}, x, P, \mathrm{sig}_{\mathrm{sk}}(x, P))$.

Verifier $\mathbf{V}_\Pi(x, p = (\mathrm{vk}, x', P, z), \Sigma)$:

(1) Check $x = x'$, and $\mathrm{verify}_{\mathrm{sig}}((x, P), z) = 1$.

(2) Decompose P into the p_i for $i \in g(\mathrm{vk})$.

(3) Return 1 if $\mathbf{V}_\pi(x, p_i, \sigma_i) = 1$ for all $i \in g(\mathrm{vk})$, and 0 otherwise.

Simulator $\mathbf{S}_\Pi(x, \Sigma, \mathrm{simtd})$:

(1) Generate vk, sk as the prover does.

(2) For $i \in g(\mathrm{vk})$ construct $p_i = \mathbf{S}_\pi(x, \sigma_i, \mathrm{simtd}_i)$ and otherwise $p_i = \epsilon$.

(3) Let $P := p_1 \circ \ldots \circ p_{g'(k)}$.

(4) Output $(\mathrm{vk}, x, P, \mathrm{sig}_{\mathrm{sk}}(x, P))$.

Extractor $\mathbf{E}(p = (\mathrm{vk}, x', P, z), \Sigma, \mathrm{extd})$:

(1) Check $\mathbf{V}_\Pi(x', p, \Sigma)$. If the outcome is 0 return $\perp$.

(2) For each $i \in g(\mathrm{vk})$ run $\mathbf{E}(p_i, \sigma_i, \mathrm{extd}_i) = w_i$. If w_i is a witness weakly symbolically-sound zero-knowledge proof system for x' then return w_i, otherwise go on.

(3) If no witness was found return $\perp$.

Proof. 1. Completeness, Zero-knowledge, Unpredictability: These properties were already shown in [107].

2. Simulation-Extractability:

Let $S_1, \ldots, S_n$ be the simulated proofs that the adversary has queried and $p = (\mathrm{vk}, x, P, z)$ the outputted proof. If the adversaries output has not this form, the verification would not succeed and there is nothing to show. In addition we may assume that $p \neq S_i$ for all $1 \leq i \leq n$, because otherwise there is again nothing to show. The case that $\mathrm{vk}_i = \mathrm{vk}_j$ for two different simulated proofs $S_i \neq S_j$ occurs with negligible probability, so we can exclude this case. Consider the following two cases:

Case (i): $\mathrm{vk} = \mathrm{vk}_i$ for some $1 \leq i \leq n$. Then one of x, P, z is different from the corresponding one in $S_i = (\mathrm{vk}_i, x_i, P_i, z_i)$. If $x \neq x_i$ or $P \neq P_i$ then $z \neq z_i$ or the verification fails. Thus w.l.o.g. $z \neq z_i$. But this means that the adversary was able to forge a signature for the one-time signature scheme which can only happen with negligible probability.

Case (ii): $\mathrm{vk} \neq \mathrm{vk}_i$ for all i. In this case holds that $g(\mathrm{vk}) \setminus \bigcup_{1 \leq i \leq n} g(\mathrm{vk}_i) \neq \emptyset$. This means there is some $j \in \{1, \ldots, q'(k)\}$ such that $j \in g(\mathrm{vk})$ but $j \notin \bigcup_{1 \leq i \leq n} g(\mathrm{vk}_i)$. If the extraction fails, then the extraction for p_j fails, too, by construction. But then, this is a successful adversary for the extractability of π. Thus this case can only occur with negligible probability, too.

Together this means that an adversary for the simulation-extractability property of Π can only succeed with negligible probability, what we wanted to show.

3. Length-regularity:

The function g always selects the same number of i from vk. So the number of proofs which are done is always the same, independent of vk. Since the proof system π is length

regular, each proof has the same length. For a security parameter ν the verification keys vk have the same length Because x and P have are length-regular we can conclude that $\mathrm{sig}_{\mathrm{sk}}(x, P)$ has the same length for all x, too. Thus the overall proof is length-regular.

□

Taking a closer look at the proof one can see that almost all properties P of the form "P has to holds even when the adversary gets access to a simulation oracle" can be derived this way when the origin proof system has the property P.

2.3 The Symbolic Model

In this section, we introduce the symbolic abstraction for zero-knowledge proofs. The model also contains standard cryptographic building blocks such as asymmetric encryptions and digital signatures. We define the constructors and destructors for every cryptographic building block separately and later on give a summary containing all terms and the adversarial capabilities (the relation $\vdash$). We finish this section with an example protocol that we also verify using the computational soundness result.

Destructors are partial functions from $\mathbf{T}^n$ to $\mathbf{T}$ where n depends on the specific destructor. We define these destructors by syntactic patterns, e.g., given constructors C_1, C_2, a destructor D could be described as $D(C_1(t_1), t_2) = t_1$ where t_1, t_2 are arbitrary terms. If no rule applies the destructor outputs $\bot$. In the example that would be the case for $D(C_2(t_1), t_2)$.

General constructors and destructors. Nonces, denoted by N, ranges over $\mathbf{N}_P \cup \mathbf{N}_E$, two disjoint infinite sets of nonces, the protocol nonces and adversary nonces, respectively.

Strings, denoted by S, can be constructed using an empty string empty and applying zeros and ones to it by the constructors $\mathrm{string}_0(S)$ and $\mathrm{string}_1(S)$. While these constructors can be used to construct strings there are also their counterparts on the destructor side to compute substrings, these destructors are defined by

$$\mathrm{unstring}_0(\mathrm{string}_0(s)) = s \text{ and } \mathrm{unstring}_1(\mathrm{string}_1(s)) = s$$

Pairings of terms are modeled by the constructor $\mathrm{pair}/2$ that takes arbitrary terms. The extraction counterparts are also part of the symbolic model and defined by

$$\mathrm{fst}(\mathrm{pair}(t_1, t_2))) = t_1 \text{ and } \mathrm{snd}(\mathrm{pair}(t_1, t_2)) = t_2$$

The strings and pairs can be used in order to model computations that are irrelevant for the cryptographic properties of the protocol.

In addition, there is a destructor to check equality which is defined by $\mathrm{equals}(x, x) = x$. Note that this is a fully syntactic equality on the symbolic side as well as on the computational side (bitstrings) and does not have any semantic properties.

Encryptions. In order to model encryptions, we need the keys $\mathrm{ek}/1$ and $\mathrm{dk}/1$ which take nonces as arguments. ek models the public encryption key and dk the private decryption key. The ciphertext itself is modeled by $\mathrm{enc}(\mathrm{ek}(N), t, N')$ and represents a ciphertext for message t encrypted using the encryption key $\mathrm{ek}(N)$ and algorithmic randomness N'. Symbolically, the algorithmic randomness just allows to distinguish different encryptions of the same plaintext; computationally, it will actually be the randomness used by the encryption algorithm.

The decryption of a ciphertext is done using the destructor $\mathrm{dec}(\mathrm{dk}(N), \mathrm{enc}(\mathrm{ek}(N), t, N')) = t$. Note that the N in the decryption key has to match the encryption key. We also model a destructor that can extract the encryption key from a ciphertext $\mathrm{ekof}(\mathrm{enc}(\mathrm{ek}(N), t, N')) = \mathrm{ek}(N)$.

Signatures. Digital signatures use two keys, similar to asymmetric encryptions, a verification key $\mathrm{vk}/1$ and a signing key $\mathrm{sk}/1$. The signature itself is then modeled by a constructor $\mathrm{sig}(\mathrm{sk}(N), t, N')$ and the verification of the signature by a destructor

$$\mathrm{verify}_{\mathrm{sig}}(\mathrm{vk}(N), \mathrm{sig}(\mathrm{sk}(N), t, N')) = t$$

Symbolically, the main difference to an encryption is that the private part is contained in the signature and the public part is used to verify whereas the ciphertext can be constructed with the public part but the decryption requires the private one. Similar to the destructor ekof we require a destructor vkof extracting the verification key from a signature, formally $\mathrm{vkof}(\mathrm{sig}(\mathrm{sk}(N), t, N')) = \mathrm{vk}(N)$.

Zero-Knowledge proofs. The zero-knowledge proofs use a common reference string (CRS) modeled by $\mathrm{crs}(N)$ corresponding to the vk of signatures. The proof itself has the form $\mathrm{ZK}(\mathrm{crs}(N), x, w, N')$. The N' is the randomness used to create the proof, x is the proven statement and w the witness. Similar to vkof and ekof we use a destructor crsof the extract the CRS from a proof. The statement x also needs to be extractable from the proof getPub. This method is required for the protocol to verify that the statement proven actually is the statement the recipient expects. However, the verification destructor is more complicated than in the cases before. The reason for the complication is that we now need to involve the relation between statements and witnesses.

We denote the statement-witness relation by $R_{\mathrm{adv}}^{\mathrm{sym}}$. The relation is part of the symbolic model, however, the computational soundness result is parametric in the relation. An example for such a relation would be $R_{\mathrm{adv}}^{\mathrm{sym}} := \{(x, w) : \exists N, M, t.x = \mathrm{enc}(\mathrm{ek}(N), t, M), w = \mathrm{dk}(N)\}$.The relation specifies what a valid witness is and what semantic meaning the statement has. In the given example, the meaning could be that a party that has access to $\mathrm{dk}(N)$ received the ciphertext x. A valid proof satisfies $(x, w) \in R_{\mathrm{adv}}^{\mathrm{sym}}$. Note that our symbolic model does not ensure that any term $\mathrm{ZK}(\mathrm{crs}(N), x, w, M)$ is a valid proof. This validity check is done by destructor $\mathrm{verify}_{\mathrm{ZK}}$ defined by

$$\mathrm{verify}_{\mathrm{ZK}}(\mathrm{crs}(t_1), \mathrm{ZK}(\mathrm{crs}(t_1), t_2, t_3, t_4)) = \mathrm{ZK}(\mathrm{crs}(t_1), t_2, t_3, t_4) \text{ if } (t_2, t_3) \in R_{\mathrm{adv}}^{\mathrm{sym}}$$

Garbage terms. The gap between the symbolic world and the computational world also includes that an adversary can send bitstrings that seem like valid terms, but only at the first glance. To close that gap, we also need to model those *garbage* terms. However, they will not be used by the protocol.

The constructor $\mathrm{garbage}/1$ represents an unspecific invalid term, $\mathrm{garbageEnc}(t, N)$ and $\mathrm{garbageSig}(t, N)$ represent invalid encryptions and signatures, respectively. Here, t represents the encryptions/signatures with public key t. For ZK proofs, we add $\mathrm{garbageZK}(t_1, t_2, N)$ where t_1 represents the CRS and t_2 the statement of the proof.

Terms and constructors. We summarize the definitions of the constructors and $\mathbf{T}$ by the following grammar:

$$\begin{aligned}\mathbf{T} ::= &\mathrm{enc}(\mathrm{ek}(N), t, N) \mid \mathrm{ek}(N) \mid \mathrm{dk}(N) \mid \mathrm{sig}(\mathrm{sk}(N), t, N) \mid \mathrm{vk}(N) \mid \mathrm{sk}(N) \mid \\ &\mathrm{crs}(N) \mid \mathrm{ZK}(\mathrm{crs}(N), t, t, N) \mid \mathrm{pair}(t, t) \mid S \mid N \mid \\ &\mathrm{garbage}(N) \mid \mathrm{garbageEnc}(t, N) \mid \mathrm{garbageSig}(t, N) \mid \mathrm{garbageZK}(t, t, N) \\ S ::= &\,\mathrm{empty} \mid \mathrm{string}_0(S) \mid \mathrm{string}_1(S)\end{aligned}$$

Note that the grammar also defines the constructors and their arity:

$$\begin{aligned}\mathbf{C} ::= \{&\mathrm{enc}/3, \mathrm{ek}/1, \mathrm{dk}/1, \mathrm{sig}/3, \mathrm{sk}/1, \mathrm{vk}/1, \mathrm{crs}/1, \mathrm{ZK}/4, \mathrm{pair}/2, \mathrm{string}_0/1, \mathrm{string}_1/1, \\ &\mathrm{garbage}/1, \mathrm{garbageEnc}/2, \mathrm{garbageSig}/2, \mathrm{garbageZK}/3\}\end{aligned}$$

Destructors. All destructors are specified in Figure 2.1. Note that there are a number of destructors that do not modify their input ($\mathrm{isek}, \mathrm{iszk}, \ldots$). These are useful for testing properties of terms: The protocol can, e.g., compute $\mathrm{isek}(t)$ and then branch depending on whether the destructor succeeds.

The adversary. The capabilities of the adversary are described by the deduction relation $\vdash$. $S \vdash t$ means that from the terms S, the adversary can deduce t. $\vdash$ is defined by the following rules:

$$\frac{m \in S}{S \vdash m} \qquad \frac{N \in \mathbf{N}_E}{S \vdash N} \qquad \frac{S \vdash t_1, \ldots, t_n \qquad t_1, \ldots, t_n \in \mathbf{T} \qquad F \text{ constructor or destructor} \qquad F(t_1, \ldots, t_n) \in \mathbf{T}}{S \vdash F(t_1, \ldots, t_n)}$$

Note that the adversary cannot deduce protocol nonces. These are secret until explicitly revealed. The capabilities of the adversaries with respect to the network (intercept/modify messages) are modeled explicitly by the protocol: if the adversary is allowed to intercept a message, the protocol explicitly communicates it through the adversary. Furthermore, the definition of $\vdash$ is very generic and in particular independent from the actual constructors, destructors or the specific definition of $\mathbf{T}$.

$$
\begin{aligned}
\mathrm{dec}(\mathrm{dk}(t_1), \mathrm{enc}(\mathrm{ek}(t_1), m, t_2)) &= m \\
\mathrm{verify}_{\mathrm{sig}}(\mathrm{vk}(t_1), \mathrm{sig}(\mathrm{sk}(t_1), t_2, t_3)) &= t_2 \\
\mathrm{isek}(\mathrm{ek}(t)) &= \mathrm{ek}(t) \\
\mathrm{isvk}(\mathrm{vk}(t)) &= \mathrm{vk}(t) \\
\mathrm{isenc}(\mathrm{enc}(\mathrm{ek}(t_1), t_2, t_3)) &= \mathrm{enc}(\mathrm{ek}(t_1), t_2, t_3) \\
\mathrm{isenc}(\mathrm{garbageEnc}(t_1, t_2)) &= \mathrm{garbageEnc}(t_1, t_2) \\
\mathrm{issig}(\mathrm{sig}(\mathrm{sk}(t_1), t_2, t_3)) &= \mathrm{sig}(\mathrm{sk}(t_1), t_2, t_3) \\
\mathrm{issig}(\mathrm{garbageSig}(t_1, t_2)) &= \mathrm{garbageSig}(t_1, t_2) \\
\mathrm{iscrs}(\mathrm{crs}(t_1)) &= \mathrm{crs}(t_1) \\
\mathrm{iszk}(\mathrm{ZK}(t_1, t_2, t_3, t_4)) &= \mathrm{ZK}(t_1, t_2, t_3, t_4) \\
\mathrm{iszk}(\mathrm{garbageZK}(t_1, t_2, t_3)) & \\
&= \mathrm{garbageZK}(t_1, t_2, t_3) \\
\mathrm{ekof}(\mathrm{enc}(\mathrm{ek}(t_1), t_2, t_3)) &= \mathrm{ek}(t_1) \\
\mathrm{ekof}(\mathrm{garbageEnc}(t_1, t_2)) &= t_1 \\
\mathrm{crsof}(\mathrm{ZK}(\mathrm{crs}(t_1), t_2, t_3, t_4)) &= \mathrm{crs}(t_1) \\
\mathrm{crsof}(\mathrm{garbageZK}(t_1, t_2, t_3)) &= t_1 \\
\mathrm{vkof}(\mathrm{sig}(\mathrm{sk}(t_1), t_2, t_3)) &= \mathrm{vk}(t_1) \\
\mathrm{vkof}(\mathrm{garbageSig}(t_1, t_2)) &= t_1 \\
\mathrm{fst}(\mathrm{pair}(t_1, t_2))) &= t_1 \\
\mathrm{snd}(\mathrm{pair}(t_1, t_2)) &= t_2 \\
\mathrm{unstring}_0(\mathrm{string}_0(s)) &= s \\
\mathrm{unstring}_1(\mathrm{string}_1(s)) &= s \\
\mathrm{getPub}(\mathrm{ZK}(t_1, t_2, t_3, t_4)) &= t_2 \\
\mathrm{getPub}(\mathrm{garbageZK}(t_1, t_2, t_3)) &= t_2 \\
\mathrm{equals}(x, x) &= x \\
\mathrm{verify}_{\mathrm{ZK}}(\mathrm{crs}(t_1), \mathrm{ZK}(\mathrm{crs}(t_1), t_2, t_3, t_4)) &= \mathrm{ZK}(\mathrm{crs}(t_1), t_2, t_3, t_4) \text{ if } (t_2, t_3) \in R_{\mathrm{adv}}^{\mathrm{sym}}
\end{aligned}
$$

Figure 2.1: Definition of all destructors. If no rule matches, a destructor returns $\perp$.

Protocol example. To show the usability of our modeling, we give a small protocol example. The protocol is a variant of the Needham-Schroeder-Protocol in which the recipient proves that he knows a nonce instead of sending that nonce back.

$$A \xrightarrow{m_1 := \mathrm{enc}(\mathrm{ek}_B, N_1, r_1)} B$$
$$A \xleftarrow{\mathrm{ZK}(\mathrm{crs}, (\mathrm{enc}(\mathrm{ek}_A, N_2, r_2), m_1), \mathrm{dk}_B, r_3)} B$$
$$A \xrightarrow{\mathrm{enc}(\mathrm{ek}_B, N_2, r_4)} B$$

The relations used for the ZK-proofs are $R^{\mathrm{sym}}_{\mathrm{honest}} = \{((m', m_1), (\mathrm{dk})) : \mathrm{dec}(\mathrm{dk}, m_1) \neq \bot\}$ and $R^{\mathrm{sym}}_{\mathrm{adv}} = R^{\mathrm{sym}}_{\mathrm{honest}} \cup \{((m', m_1), (\mathrm{dk})) : m_1 = \mathrm{garbageEnc}(t, N), t \in \mathbf{T}, N \in \mathbf{N}\}$, i.e., B proves that he can decrypt m_1. The part m' of the statement is not used in the relation, but the non-malleability of our ZK proofs ensures that the adversary cannot change m' in an existing proof.

In the section Section 2.5, we show how to use the computational soundness result to analyse the example given here in the applied π-calculus using the verification tool ProVerif [38].

2.4 Computational Soundness

In this section, we prove the computational soundness of the model. We start by stating the main theorem. The rest of the section is used to prove the theorem. However, for the sake of readability, some lengthy parts of the proof are postponed to the Appendix A.1. The key ideas of those proofs are kept present in the section. Since the soundness result cannot hold for arbitrary protocols and implementations, we define the criteria for soundness in the subsection after the theorem.

Theorem 2.4.1 (Computational soundness of ZK proofs)**.** *Let* Π *be a protocol satisfying the protocol conditions listed in figure 2.2. Let* A_F *be a computational implementation satisfying the implementation conditions listed in figure 2.3. Then for any node trace* $\mathcal{P}$*, if* Π *symbolically satisfies* $\mathcal{P}$*, then* Π *computationally satisfies* $\mathcal{P}$*.*

2.4.1 Theorem conditions

We first give the protocol conditions and then continue with the implementation conditions separately. This simplifies the future referencing for verifying or implementing protocols using the soundness result.

Protocol conditions. Most of the conditions simply resemble common sense, however, it is important to precisely state them for the proof. There are two conditions regarding the randomness: one is that randomness of keys, encryptions and signatures is only used once; the other is that no adversarial randomness $\mathbf{N}_E$ is used.

There is an exception for the randomness in ciphertext and signatures when it comes to the witness of zero-knowledge proofs. Here, the randomness might occur again in the witness part. We need to allow this usage since real-world schemes require the knowledge of the used algorithmic randomness.

As stated when introduced, the garbage terms are to symbolically reflect computationally malformed terms. We require the protocol to only produce well-formed terms. Furthermore, the protocol has to use decryption, as well as the verification of signatures and zero-knowledge the correct way, i.e., the first argument has to be the corresponding key.

The remaining conditions are regarding zero-knowledge. In the case of non-interactive ZK proofs that use a CRS, this CRS has to be constructed in an honest way. Consequently, we require that the used CRS makes use of a protocol nonce $\mathbf{N}_P$.

So far, all protocol conditions have been of a rather syntactic nature. The following, most interesting, protocol condition is a semantic one. We call it the *valid proofs condition* and it defines the usage of the relations we introduced for ZK proofs $R^{\mathrm{sym}}_{\mathrm{honest}}$ and $R^{\mathrm{sym}}_{\mathrm{adv}}$ in which our symbolic model is parameterized. It requires that the protocol only generates proofs that are actually true, i.e., whenever the protocol constructs a ZK proof $\mathrm{ZK}(c, x, w, N)$ we have $(x, w) \in R^{\mathrm{sym}}_{\mathrm{honest}}$. Then the valid proofs condition simply requires that the protocol never tries to construct a ZK-proof with an invalid witness. Note that, we only impose this condition on the honest protocol, not on the adversary.

We summarized all conditions in Figure 2.2.

Implementation conditions. We now describe how to implement the constructors and destructors computationally. We do so by specifying a partial deterministic function $A_F : (\{0,1\}^*)^n \to \{0,1\}^*$ (the *computational implementation of* F) for each constructor or destructor $F : \mathbf{T}^n \to \mathbf{T}$. Intuitively, A_F should behave as F, only on bitstrings, e.g., $A_{\mathrm{enc}}(ek, m, r)$ should encrypt m using encryption key ek and algorithmic randomness r. The distribution A_N specifies the distribution according to which nonces are picked. We split the conditions in three parts, general conditions, cryptographic conditions and more specifically zero-knowledge conditions. We list all conditions in a formal specification in Figure 2.3.

General conditions. In the symbolic scenario there is implicitly a type given with every constructor. This is particularly used in the destructor definitions. Consequently, we require the output strings to be typed. This type has to be efficiently recognizable.

Similar to the protocol conditions, the protocol conditions are mostly simple syntactic conditions such as $A_{\mathrm{fst}}(A_{\mathrm{pair}}(x, y)) = x$ or that implementations fail on wrong types. For example, the verification key extraction should only work on strings having the type signature.

Moreover, we require length regularity, i.e., if two different inputs to an implementation have the same length then the two output have the same length, too.

Cryptographic conditions. The implementation of nonce has to be a uniformly random distribution.

The *encryption scheme* has to be IND-CCA secure. In addition, we require that two ciphertexts are equal with negligible probability, given different randomnesses and the same plaintext. Note that this is not directly implied by the definition of IND-CCA security since the definition only considers overwhelming probability. However, we require that this holds for

1. The annotation of each crs-node, each key-pair $(\mathrm{ek}, \mathrm{dk})$ and $(\mathrm{vk}, \mathrm{sk})$ is a fresh nonce which does not occur anywhere else.
2. There is no node annotated with a $\mathrm{garbage}, \mathrm{garbageEnc}$, $\mathrm{garbageSig}$, $\mathrm{garbageZK}$, or $N \in \mathbf{N}_E$ constructor in the protocol.
3. The last argument of a $\mathrm{enc}, \mathrm{sig}, \mathrm{ZK}$ constructor are fresh nonces. These nonces are not used anywhere else except in case of enc and sig as part of a subterm of the third argument in a ZK-node.
4. A dk-node is only used as first argument for dec-node or as subterm of the third argument in a ZK-node.
5. A sk-node is only used as first argument for sig-node or as subterm of the third argument in a ZK-node.
6. The first argument of a dec-computation node is a dk-node.
7. The first argument of a sig-computation node is a sk-node.
8. The first argument of a ZK-computation is a crs-computation node which is annotated by a nonce $N \in \mathbf{N}_P$. This nonce is only used as annotation of this crs node and nowhere else.
9. The first argument of a $\mathrm{verify}_{\mathrm{ZK}}$-computation is a crs-computation node which is annotated by a nonce $N \in \mathbf{N}_P$. This nonce is only used as annotation of this crs node and nowhere else.
10. The protocol respects the usage restriction $R^{\mathrm{sym}}_{\mathrm{honest}}$ in the following sense: In the symbolic execution of the protocol, whenever a ZK-computation-node ν is reached, then $(f(\nu_x), f(\nu_w)) \in R^{\mathrm{sym}}_{\mathrm{honest}}$ where f is the function mapping nodes to terms (cf. the definition of the symbolic execution) and ν_x and ν_w are the second and third argument of ν.
11. For the relation $R^{\mathrm{sym}}_{\mathrm{adv}}$ it holds: There is an efficient algorithm $\mathbf{SymbExtr}$, that given a term M together with a set S of terms, outputs a term N, such that $S \vdash N$ and $(N, M) \in R^{\mathrm{sym}}_{\mathrm{adv}}$ or outputs $\bot$ if there is no such term N. We call a relation satisfying this property symbolically extractable.
12. The relation $R^{\mathrm{sym}}_{\mathrm{adv}}$ is efficiently decidable.

Figure 2.2: List of all protocol conditions for computational soundness

1. The implementation is an implementation according to Definition 2.1.5 (see Section 2.1).

2. There are disjoint and efficiently recognizable sets of bitstrings representing the node types nonce, ciphertext, encryption key, decryption key, signature, verification key, signing key, common reference string, zero-knowledge proof, pair and payload-string.

   The images of A_N have type nonce (for all $N \in \mathbf{N}$), A_{enc} have type ciphertext, A_{ek} have type encryption key, A_{dk} have type decryption key, A_{sig} have type signature, A_{vk} have type verification key, A_{sk} have type signing key, A_{crs} have type common reference string, A_{ZK} have type zero-knowledge proof, A_{pair} have type pair, and $A_{\mathrm{string}_0}, A_{\mathrm{string}_1}, A_{\mathrm{empty}}$ have type payload string.

3. The implementation A_N for nonces $N \in \mathbf{N}_P$ compute uniform distributions on $\{0,1\}^\eta$ and output the sampled value tagged as nonce (here η is the security parameter).

4. If $A_{\mathrm{dec}}(\mathrm{dk}_N, m) \neq \perp$ then $A_{\mathrm{ekof}}(m) = \mathrm{ek}_N$, i.e. the decryption only succeeds if the corresponding encryption key can be extracted out of the ciphertext.

5. $A_{\mathrm{vkof}}(A_{sig}(A_{\mathrm{sk}}(x), y, z)) = A_{\mathrm{vk}}(x)$ for all $y \in \{0,1\}^*$ and x, z nonces. If e is of type signature then $A_{\mathrm{vkof}}(e) \neq \perp$, otherwise $A_{\mathrm{vkof}}(e) = \perp$.

6. For all $m, k \in \{0,1\}^*$, k having type encryption key, and $r \neq r' \in \{0,1\}^*$ with $|r| = |r'|$ holds that $A_{\mathrm{enc}}(k, m, r)$ and $A_{\mathrm{enc}}(k, m, r')$ are equal with negligible probability.

7. For all $m, k \in \{0,1\}^*$, k having type signing key, and $r \neq r' \in \{0,1\}^*$ with $|r| = |r'|$ holds that $A_{\mathrm{sig}}(k, m, r)$ and $A_{\mathrm{sig}}(k, m, r')$ are equal with negligible probability.

8. The implementations A_{ek}, A_{dk}, A_{enc}, and A_{dec} belong to an encryption scheme $(\mathrm{KeyGen}_{\mathrm{enc}}, \mathrm{ENC}, \mathrm{DEC})$ which is IND-CCA secure.

9. The implementations A_{vk},A_{sk},A_{sig}, and $A_{\mathrm{verify}_{\mathrm{sig}}}$ belong to a signature scheme $(\mathrm{KeyGen}_{\mathrm{sig}}, \mathrm{SIG}, \mathrm{VER}_{\mathrm{sig}})$ which is strongly existential unforgeable.

10. All implementations are length regular, i.e. if the input has the same length the output will have the same too.

11. For $m_1, m_2 \in \{0,1\}^*$ holds $A_{\mathrm{fst}}(A_{\mathrm{pair}}(m_1, m_2)) = m_1$ and $A_{\mathrm{snd}}(A_{\mathrm{pair}}(m_1, m_2)) = m_2$

12. $A_{\mathrm{dec}}(A_{\mathrm{dk}}(r), A_{\mathrm{enc}}(A_{\mathrm{ek}}(r), m, r')) = m$ for all r, r' nonces.

13. Let $k \in \{0,1\}^*$ be an encryption key and $m, n \in \{0,1\}^*$ such that n is of type nonce. Then holds $A_{\mathrm{ekof}}(A_{\mathrm{enc}}(k, m, n)) = k$. If $c \in \{0,1\}^*$ is not of type ciphertext then $A_{\mathrm{ekof}}(c) = \perp$.

Figure 2.3: List of all implementation conditions for computational soundness

14. Let $\mathrm{vk}, \mathrm{sk} \in \{0,1\}^*$ be a keypair, i.e. $(\mathrm{vk}, \mathrm{sk})$ is in the image of $\mathrm{KeyGen}_{\mathrm{sig}}$, then holds for all $m, n \in \{0,1\}^*$: $A_{\mathrm{vkof}}(A_{\mathrm{sig}}(\mathrm{sk}, m, n)) = \mathrm{vk}$.

15. $A_{\mathrm{verify}_{\mathrm{sig}}}(A_{\mathrm{vk}}(r), A_{\mathrm{sig}}(A_{\mathrm{sk}}(r), m, r')) = m$ for all r, r' nonces.

16. For all $p, s \in \{0,1\}^*$ we have that $A_{\mathrm{verify}_{\mathrm{sig}}}(p, s) \neq \bot$ implies $A_{\mathrm{vkof}}(s) = p$.

17. For $m \in \{0,1\}^*$ holds $A_{\mathrm{unstring_i}}(A_{\mathrm{string_i}}(m)) = m$ for $i \in \{0,1\}$ and $A_{\mathrm{string_0}}(m) \neq A_{\mathrm{string_1}}(m)$.

18. For all $m \in \{0,1\}^*$ of type zero-knowledge proof holds that $\mathrm{iszk}(m) = m$ and if m has not type zero-knowledge proof, then $\mathrm{iszk}(m) = \bot$. The same holds for issig w.r.t. the type signature and isenc w.r.t. the type ciphertext.

19. If $k \in \{0,1\}^*$ is not of the type encryption key then holds for all $m, n \in \{0,1\}^*$ that $A_{\mathrm{enc}}(k, m, n) = \bot$. The same has to hold for the type signing key and the implementation of signatures.

20. The implementation $A_{\mathrm{crs}}, A_{\mathrm{ZK}}, A_{\mathrm{verify}_{\mathrm{ZK}}}$ belongs to a zero-knowledge proof system $(\mathbf{K}, \mathbf{P}, \mathbf{V})$ which is a weak symbolically-sound zero-knowledge proof system.

21. For all $z \in \{0,1\}^*$ holds $A_{\mathrm{verify}_{\mathrm{ZK}}}(\mathrm{crsof}(z), z) \in \{\bot, z\}$, where $A_{\mathrm{verify}_{\mathrm{ZK}}}(\mathrm{crsof}(z), z) = z$ if and only if z is correct w.r.t. to the verifier of the proof system.

22. If $z \in \{0,1\}^*$ is not of type zero-knowledge, then $\mathrm{verify}_{\mathrm{ZK}}(\mathrm{crsof}(z), z) = \bot$.

23. For all $p, q, r, s \in \{0,1\}^*$ we have that $z = A_{\mathrm{ZK}}(p, q, r, s) \neq \bot$ implies $A_{\mathrm{crsof}}(z) = p$.

24. For all $z \in \{0,1\}^*$ holds: If z is not of type zero-knowledge proof then $A_{\mathrm{crsof}}(z) = \bot$.

25. If $z := A_{\mathrm{ZK}}(\bar{m}) \neq \bot$ then $A_{\mathrm{verify}_{\mathrm{ZK}}}(A_{\mathrm{crsof}}(z), z) = 1$.

26. If $(x, w) \notin R^{\mathrm{comp}}_{\mathrm{honest}}$ then for all $c, r \in \{0,1\}^*$, it holds $A_{\mathrm{ZK}}(c, x, w, r) = \bot$.

27. Let $c, x, w, n \in \{0,1\}^*$ such that c is of type crs and let $z = A_{\mathrm{ZK}}(c, x, w, n)$. If $z \neq \bot$ then holds that $x = A_{\mathrm{getPub}}(z)$.

28. We require that the relations $R^{\mathrm{comp}}_{\mathrm{honest}}, R^{\mathrm{comp}}_{\mathrm{adv}}$ are an implementation of $R^{\mathrm{sym}}_{\mathrm{adv}}$ with usage restriction $R^{\mathrm{sym}}_{\mathrm{honest}}$ in the sense of Definition 2.4.2.

29. For $d \in \{0,1\}^*$ of type decryption key there is a efficiently computable function $p : \{0,1\}^* \to \{0,1\}^*$ such that for all $m, n \in \{0,1\}^*$, n of type nonce, it holds $A_{\mathrm{dec}}(d, A_{\mathrm{enc}}(p(d), m, n)) = m$, i.e. p computes the encryption key corresponding to d. The analogous statement has to hold for signing keys and verification keys.

Figure 2.3: List of all implementation conditions for computational soundness (cont.)

every randomness. The *signature scheme* is strongly existential unforgeable. Both properties are very common cryptographic requirements for encryption schemes and signatures schemes, respectively.

For a discussion of the properties, we refer to the related work section. Here, we will focus on the cryptographic properties the implementation of ZK proofs should satisfy.

Zero-Knowledge. We then require that $A_{\mathrm{crs}}, A_{\mathrm{ZK}}, A_{\mathrm{verify_{ZK}}}$ correspond to the key generation $\mathbf{K}$, prover $\mathbf{P}$, and verifier $\mathbf{V}$ of a weakly symbolically-sound ZK proof system for some relations $R^{\mathrm{comp}}_{\mathrm{honest}}, R^{\mathrm{comp}}_{\mathrm{adv}}$.

Although this requirement is already sufficient from an implementation point of view, we also need to link the computational relations $R^{\mathrm{comp}}_{\mathrm{honest}}, R^{\mathrm{comp}}_{\mathrm{adv}}$ and the symbolic relations $R^{\mathrm{sym}}_{\mathrm{honest}}, R^{\mathrm{sym}}_{\mathrm{adv}}$. Obviously, we cannot expect computational soundness if we allow arbitrary $R^{\mathrm{comp}}_{\mathrm{honest}}, R^{\mathrm{comp}}_{\mathrm{adv}}$. Instead, we need to formulate the fact that $R^{\mathrm{comp}}_{\mathrm{honest}}, R^{\mathrm{comp}}_{\mathrm{adv}}$ somehow correspond to the symbolic relations $R^{\mathrm{sym}}_{\mathrm{honest}}, R^{\mathrm{sym}}_{\mathrm{adv}}$. We thus give minimal requirements on the relationship between those relations. Essentially, we want that whenever $(x, w) \in R^{\mathrm{sym}}_{\mathrm{honest}}$ then for the corresponding computational bitstrings m_x, m_w we have $(m_x, m_w) \in R^{\mathrm{comp}}_{\mathrm{honest}}$; this guarantees that if symbolically, we respect the usage restriction $R^{\mathrm{sym}}_{\mathrm{honest}}$, then computationally we only use witnesses the honest protocol is allowed to use. And whenever $(m_x, m_w) \in R^{\mathrm{comp}}_{\mathrm{adv}}$ we have $(x, w) \in R^{\mathrm{sym}}_{\mathrm{adv}}$; this guarantees that a computational adversary will not be able to prove statements m_x that do not also correspond to statements x that can be proven symbolically.

Formally specifying these conditions has the difficulty that two relations are over bitstrings and the others over terms. Thus, we need to translate between terms and bitstring without using the simulator in the proof that essentially implements such a translation. Hence, we define a function img_η that translates a term to a bitstring which is basically done by applying A_F for each constructor. Furthermore, the function img_η depends on an environment η, a partial function $\mathbf{T} \to \{0,1\}^*$ that assigns bitstrings to nonces and adversary-generated terms.

Since, we cannot expect the result to work for an arbitrary function η, we list the properties an environment has to satisfy and call such an environment *consistent*.

Definition 2.4.1. *Let $\mathcal{E}$ be the set of all partial functions $\eta : \mathbf{T} \to \{0,1\}^*$. We will call such an η an environment. Let an implementation A for the symbolic model by given. Define the partial function $\mathrm{img}_\eta : \mathbf{T} \to \{0,1\}^*$ for $\eta \in \mathcal{E}$ by taking the first matching rule:*

- *For a nonce N define $\mathrm{img}_\eta(N) := \eta(N)$*
- *For a term $t = \mathrm{crs}(N)$ define $\mathrm{img}_\eta(\mathrm{crs}(N)) := \eta(t)$*
- *For a term $t = \mathrm{ZK}(\mathrm{crs}(N), x, w, M)$ define $\mathrm{img}_\eta(t) := \eta(t)$*
- *Let C be a constructor from $\{\mathrm{ek}, \mathrm{dk}, \mathrm{vk}, \mathrm{sk}, \mathrm{enc}, \mathrm{sig}, \mathrm{crs}, \mathrm{garbageZK}, \mathrm{garbageSig}, \mathrm{garbageEnc}, \mathrm{garbage}\}$. For $t = C(t_1, \ldots, t_{n-1}, N)$ with $N \in \mathbf{N}_E$ define $\mathrm{img}_\eta(t) := \eta(t)$.*

- *For a term* $C(t_1, \ldots, t_n)$ *define* $\mathrm{img}_\eta(C(t_1, \ldots, t_n)) := A_C(\mathrm{img}_\eta(t_1), \ldots, \mathrm{img}_\eta(t_n))$, *if for all* i *we have* $\mathrm{img}_\eta(t_i) \neq \bot$*, and* $\bot$ *otherwise.*

An environment η *is consistent if the following conditions are satisfied:* [4]

- η *is injective.*
- *For each constructor* C *we require that the bitstring* $\mathrm{img}_\eta(C(t_1, \ldots, t_n))$ *has the type as follows: The constructors* $\mathrm{enc}, \mathrm{garbageEnc}$ *are mapped to type ciphertext,* $\mathrm{sig}, \mathrm{garbageSig}$ *to signatures,* $\mathrm{ZK}, \mathrm{garbageZK}$ *to ZK proofs,* $\mathrm{ek}, \mathrm{dk}, \mathrm{vk}, \mathrm{sk}$ *to encryption, decryption, verification, signing key, respectively.* crs *to common reference string,* pair *to pair,* $\mathrm{string}_0, \mathrm{string}_1, \mathrm{empty}$ *to payload-string,* $\mathbf{N}$ *to nonce, earlier version of* img_η*.* $\mathrm{garbage}$ *has none of these types.*
- $A_{\mathrm{ekof}}(\mathrm{img}_\eta(\mathrm{enc}(\mathrm{ek}(N), t, M))) = \mathrm{img}_\eta(\mathrm{ek}(N))$ *for all* $N, M \in \mathbf{N}_P$*,* $t \in \mathbf{T}$*.*
- *For all* $t = \mathrm{sig}(\mathrm{sk}(N), u, M)$ *with* $N, M \in \mathbf{N}, u \in \mathbf{T}$ *it holds:* $\mathrm{verify}_{\mathrm{sig}}(\mathrm{vkof}(t), t) \neq \bot$ *implies that* $A_{\mathrm{verify}_{\mathrm{sig}}}(\mathrm{img}_\eta(\mathrm{vkof}(t)), \mathrm{img}_\eta(t))\, \mathrm{img}_\eta(u)$*.*
- *For* $t = \mathrm{ZK}(\mathrm{crs}(N), x, , w, M)$ *with* $M \in \mathbf{N}$ *holds:*
  1. $A_{\mathrm{verify}_{\mathrm{ZK}}}(\mathrm{img}_\eta(\mathrm{crs}(N)), \eta(t)) = \eta(t)$
  2. $A_{\mathrm{getPub}}(\eta(t)) = \mathrm{img}_\eta(x)$
  3. $A_{\mathrm{crsof}}(\eta(t)) = \mathrm{img}_\eta(\mathrm{crs}(N))$
- *For all* $t_1, t_2 \in \mathbf{T}$ *it holds that* $A_{\mathrm{verify}_{\mathrm{sig}}}(\mathrm{img}_\eta(\mathrm{garbageSig}(t_1, t_2))) = \bot$
- *For all* $N, M \in \mathbf{N}$*,* $t \in \mathbf{T}$ *it holds that* $A_{\mathrm{dec}}(\mathrm{img}_\eta(\mathrm{dk}(N)), \mathrm{img}_\eta(\mathrm{enc}(\mathrm{ek}(N), t, M))) = \mathrm{img}_\eta(t)$ *and* $\mathrm{img}_\eta(t) \neq \bot$*.*
- *For all* $\mathrm{enc}(\mathrm{ek}(N), t, M) \in \mathbf{T}$ *it holds: If* $\mathrm{img}_\eta(\mathrm{enc}(\mathrm{ek}(N), t, M)) =: c \neq \bot$*, then it follows* $A_{\mathrm{ekof}}(c) = \mathrm{img}_\eta(\mathrm{ek}(N))$*.*
- *For all* $\mathrm{enc}(\mathrm{ek}(N), t, M) \in \mathbf{T}$ *it holds: If* $\mathrm{img}_\eta(\mathrm{enc}(\mathrm{ek}(N), t, M)) \neq \bot$ *and* $d \in \{0,1\}^*$ *such that* $\mathrm{img}_\eta(\mathrm{ek}(N)) = p(d)$[5]*, then it follows that* $A_{\mathrm{dec}}(d, \mathrm{img}_\eta(\mathrm{enc}(\mathrm{ek}(N), t, M))) = \mathrm{img}_\eta(t)$*.*

Given these notions, we can formalize the conditions $R^{\mathrm{comp}}_{\mathrm{honest}}, R^{\mathrm{comp}}_{\mathrm{adv}}$ should satisfy in three simple requirements that reflect the verbalization above:

Definition 2.4.2 (Implementation of relations)**.** *A pair of relations* $R^{\mathrm{comp}}_{\mathrm{honest}}, R^{\mathrm{comp}}_{\mathrm{adv}}$ *on* $\{0,1\}^*$ *implement a relation* $R^{\mathrm{sym}}_{\mathrm{adv}}$ *on* $\mathbf{T}$ *with usage restriction* $R^{\mathrm{sym}}_{\mathrm{honest}}$ *if the following conditions hold for any consistent environment* η*:*

[4] We consider a condition in which a term t occurs such that $\mathrm{img}_\eta(t) = \bot$ as satisfied.

[5] Where p is the function defined in implementation condition 29.

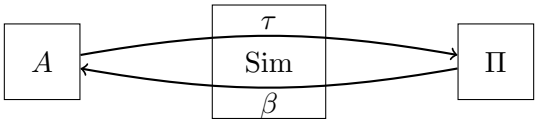

Figure 2.4: A typical CoSP simulator

(i) $(x, w) \in R^{\text{sym}}_{\text{honest}}$ *and* $\text{img}_\eta(x) \neq \bot \neq \text{img}_\eta(w) \implies (\text{img}_\eta(x), \text{img}_\eta(w)) \in R^{\text{comp}}_{\text{honest}}$

(ii) $(\text{img}_\eta(x), \text{img}_\eta(w)) \in R^{\text{comp}}_{\text{adv}} \implies (x, w) \in R^{\text{sym}}_{\text{adv}}$

(iii) $R^{\text{sym}}_{\text{honest}} \subseteq R^{\text{sym}}_{\text{adv}}$ *and* $R^{\text{comp}}_{\text{honest}} \subseteq R^{\text{comp}}_{\text{adv}}$ ◇

In the Section 2.5 we give some practical examples satisfying this definition and show how to add more properties to the list of *consistent environments* in order to include more relations. Note that the list of conditions is minimized to the ones used in the soundness proof. However, we need to extend the list carefully in order to prove that the example relations satisfy the definition 2.4.2.

2.4.2 Proof of the Computational Soundess

We now have all definitions and conditions to prove the stated main theorem 2.4.1 of this chapter. Since the proof in total is lengthy, we structure the rest of the section as follows: we first explain the general proof strategy, then we go into detail regarding the constructions and lemmata. However, we also put some less insightful but necessary parts of the proof in the Appendix A.1.

Proof Strategy

In the CoSP section 2.1, we already stated that the proof will be simulator based. That means, we define a simulator Sim that (i) produces execution traces which are indistinguishable from real-world adversary executions and in addition, (ii) the simulator execution does only produce terms can be derived from previous knowledge (i.e., the simulator is Dolev-Yao style). The existence of such a simulator then guarantees computational soundness: Dolev-Yaoness guarantees that only node traces occur in the hybrid execution that are possible in the symbolic execution, and indistinguishability guarantees that only node traces occur in the computational execution that can occur in the hybrid one.

The simulator is a machine that interacts with a symbolic execution of the protocol Π on the one hand, and with the adversary A on the other hand; we call this a hybrid execution. As depicted in Figure 2.4, we model the simulator by two stateful functions τ and β which translate bitstrings to terms and terms to bitstrings, respectively.

The properties indistinguishability and Dolev-Yaoness can also be interpreted in the following way: indistinguishability ensures that the execution of the simulator is connected to real-world adversarial execution and Dolev-Yaoness ensures that the execution of the simulator is compliant with the symbolic execution. As both, the computational and the symbolic world,

need a different argumentation, we show the properties for different simulators and show that the property of Dolev-Yaoness carries over from one to the other simulator. More precisely, we use 7 different simulators.

Indistinguishability requires that the simulator behaves as applying the implementation, consequently, we use a simulator that basically just applies the implementation to the terms for that purpose. The difficulty here is to show that cases in which symbolic and computational executions behave differently, e.g., if two terms get transformed to the same bitstring, occur with negligible probability.

Dolev-Yaoness requires a faking simulator that does not use the terms that cannot be derived by the adversary and thus are information-theoretically not contained in the adversaries input. This is essentially achieved by replacing computations with oracles that handle the secrets and thus enables us to use the simulator as an attacker in the cryptographic definition of the corresponding building block, e.g., we can show that if the simulator is not DY then the used encryption scheme does not have the security property that we required in section 2.4.1.

The other 5 intermediate simulators are used to encapsulate specific cryptographic arguments. Here, the order is of particular importance. For example, the zero-knowledge simulation only works for true statements. However, the security definition of encryptions replaces the content and consequently makes statements computationally wrong that are still symbolically true. In particular, the security definition 2.2.1 of weakly symbolically-sound ZK proofs can only be applied before such a transformation.

The lengthiness of the proof comes from the number of constructors and destructors. For several statements, it is necessary to do structural induction over the protocol tree. The used case distinction might thus grow to 25 cases for destructors and 16 cases for constructors with a factor 2 in the cases that adversarial terms need to be distinguished from protocol terms. Many of these cases do not bring new insights and will be postponed to the appendix. But since the order of arguments has a significant impact on the proof, it is hard to modularize computational soundness proofs and reuse the arguments.

Proof Details

In this part of the section, we describe our proof of computational soundness (Theorem 2.4.1). First, we describe how the computational soundness proof for encryptions and signatures is done in the CoSP framework. To understand our proof it is essential to understand that proof first. Then, we sketch how computational soundness of zero-knowledge proofs that have the extraction zero-knowledge property was shown in [21] (section 2.4.2). It is instructive to compare their approach to ours. In section 2.4.2, we describe the idea underlying our proof (using simulation-sound extractability instead of extraction-zero knowledge). Finally, in section 2.4.2 we give an overview of our proof. The full proof is given in appendix A.1. The lemmas in this overview are simplified for readability and informal.

How to construct a simulator? In [14], the simulator Sim is constructed as follows: Whenever it gets a term from the protocol, it constructs a corresponding bitstring and sends it to the

adversary, and when receiving a bitstring from the adversary it parses it and sends the resulting term to the protocol. Constructing bitstrings is done using a function β, parsing bitstrings to terms using a function τ. (See Figure 2.4.) The simulator picks all random values and keys himself: For each protocol nonce N, he initially picks a bitstring r_N. He then translates, e.g., $\beta(N) := r_N$ and $\beta(\mathrm{ek}(N)) := A_{\mathrm{ek}}(r_N)$ and $\beta(\mathrm{enc}(\mathrm{ek}(N), t, M)) := A_{\mathrm{enc}}(A_{\mathrm{ek}}(r_N), \beta(t), r_M)$. Translating back also is natural: Given $m = r_N$, we let $\tau(m) := N$, and if c is a ciphertext that can be decrypted as m using $A_{\mathrm{dk}}(r_N)$, we set $\tau(c) := \mathrm{enc}(\mathrm{ek}(N), \tau(m), M)$. However, in the last case, a subtlety occurs: what nonce M should we use as symbolic randomness in $\tau(c)$? Here we distinguish two cases:

If c was earlier produced by the simulator: Then c was the result of computing $\beta(t)$ for some $t = \mathrm{enc}(\mathrm{ek}(N), t', M)$ and some nonce M. We then simply set $\tau(c) := t$ and have consistently mapped c back to the term it came from.

If c was not produced by the simulator: In this case it is an adversary generated encryption, and M should be an adversary nonce to represent that fact. We could just use a fresh nonce $M \in \mathbf{N}_E$, but that would introduce the need of additional bookkeeping: If we compute $t := \tau(c)$, and later $\beta(t)$ is invoked, we need to make sure that $\beta(t) = c$ in order for the Sim to work consistently (formally, this is needed in the proof of the indistinguishability of Sim). And we need to make sure that when computing $\tau(c)$ again, we use the same M. This bookkeeping can be avoided using the following trick: We identify the adversary nonces with symbols N^m annotated with bitstrings m. Then $\tau(c) := \mathrm{enc}(\mathrm{ek}(N), \tau(m), N^c)$, i.e., we set $M := N^c$. This ensures that different c get different randomness nonces N^c, the same c is always assigned the same N^c, and $\beta(t)$ is easy to define: $\beta(\mathrm{enc}(\mathrm{ek}(N), m, N^c)) := c$ because we know that $\mathrm{enc}(\mathrm{ek}(N), m, N^c)$ can only have been produced by $\tau(c)$. To illustrate, here are excerpts of the definitions of β and τ (the first matching rule counts):

- $\tau(c) \coloneqq \mathrm{enc}(\mathrm{ek}(M), t, N)$ if c has earlier been output by $\beta(\mathrm{enc}(\mathrm{ek}(M), t, N))$ for some $M \in \mathbf{N}, N \in \mathbf{N}_P$
- $\tau(c) \coloneqq \mathrm{enc}(\mathrm{ek}(M), \tau(m), N^c)$ if c is of type ciphertext and $\tau(A_{\mathrm{ekof}}(c)) = \mathrm{ek}(M)$ for some $M \in \mathbf{N}_P$ and $m \coloneqq A_{\mathrm{dec}}(A_{\mathrm{dk}}(r_M), c) \neq \bot$
- $\beta(\mathrm{enc}(\mathrm{ek}(N), t, M)) \coloneqq A_{\mathrm{enc}}(A_{\mathrm{ek}}(r_N), \beta(t), r_M)$ if $M \in \mathbf{N}_P$
- $\beta(\mathrm{enc}(\mathrm{ek}(M), t, N^m)) \coloneqq m$ if $M \in \mathbf{N}_P$

Bitstrings m that cannot be suitably parsed are mapped into terms $\mathrm{garbage}(N^m)$ and similar that can then be mapped back by β using the annotation m.

Showing indistinguishability. Showing indistinguishability essentially boils down to showing that the functions β and τ consistently translate terms back and forth. More precisely, we show that $\beta(\tau(m)) = m$ and $\tau(\beta(t)) = t$. Furthermore, we need to show that in any protocol step where a constructor or destructor F is applied to terms $t_1, \ldots, t_n$, we have that $\beta(F(t_1, \ldots, t_n)) = A_F(\beta(t_1), \ldots, \beta(t_n))$. This makes sure that the computational execution

(where A_F is applied) stays in sync with the hybrid execution (where F is applied and the result is translated using β). The proofs of these facts are lengthy (involving case distinctions over all constructors and destructors) but do not provide much additional insight; they are very important though because they are responsible for most of the implementation conditions that are needed for the computational soundness result.

Showing Dolev-Yaoness. The proof of Dolev-Yaoness is where most of the actual cryptographic assumptions come in. In this sketch, we will slightly deviate from the original proof in [14] for easier comparison with the proof in the present paper. The differences are, however, inessential. Starting from the simulator Sim, we introduce a sequence of simulators Sim_4, Sim_5, Sim_f. (We start the numbering with 4 because we later introduce additional simulators.)

In Sim_4, we change the function β as follows: When invoked as $\beta(\mathrm{enc}(\mathrm{ek}(N), t, M))$ with $M \in \mathbf{N}_P$, instead of computing $A_{\mathrm{enc}}(A_{\mathrm{ek}}(r_N), \beta(t), r_M)$, β invokes an encryption oracle $\mathcal{O}^N_{\mathrm{enc}}$ to produce the ciphertext c. Similarly, $\beta(\mathrm{ek}(N))$ returns the public key provided by the oracle $\mathcal{O}^N_{\mathrm{enc}}$. The hybrid executions of Sim and Sim_4 are then indistinguishable. (Here we use that the protocol conditions guarantee that no randomness is used in two places.) Also, the function τ is changed to invoke $\mathcal{O}^N_{\mathrm{enc}}$ whenever it needs to decrypt a ciphertext while parsing. Notice that if c was returned by $\beta(t)$ with $t := \mathrm{enc}(\dots)$, then $\tau(c)$ just recalls the term t without having to decrypt. Hence $\mathcal{O}^N_{\mathrm{enc}}$ is never asked to decrypt a ciphertext it produced.

In Sim_5, we replace the encryption oracle $\mathcal{O}^N_{\mathrm{enc}}$ by a fake encryption oracle $\mathcal{O}^N_{fake}$ that encrypts zero-plaintexts instead of the true plaintexts. Since $\mathcal{O}^N_{\mathrm{enc}}$ is never asked to decrypt a ciphertext it produced, IND-CCA security guarantees that the hybrid executions of Sim_4 and Sim_5 are indistinguishable. Since the plaintexts given to $\mathcal{O}^N_{fake}$ are never used, we can further change $\beta(\mathrm{enc}(N, t, M))$ to never even compute the plaintext $\beta(t)$.

Finally, in Sim_f, we additionally change β to use a signing oracle in order to produce signatures. As in the case of Sim_4, the hybrid executions of Sim_5 and Sim_f are indistinguishable.

Since the hybrid executions of Sim and Sim_f are indistinguishable, in order to show Dolev-Yaoness of Sim, it is sufficient to show Dolev-Yaoness of Sim_f.

The first step to showing this is to show that whenever Sim_f invokes $\beta(t)$, then $S \vdash t$ holds (where S are the terms received from the protocol). This follows from the fact that β is invoked on terms t_0 sent by the protocol (which are then by definition in S), and recursively descends only into subterms that can be deduced from t_0. In particular, in Sim_5 we made sure that $\beta(t)$ is not invoked by $\beta(\mathrm{enc}(\mathrm{ek}(N), t, M))$; t would not be deducible from $\mathrm{enc}(\mathrm{ek}(N), t, M)$.

Next we prove that whenever $S \nvdash t$, then t contains a visible subterm t_{bad} with $S \nvdash t_{bad}$ such that t_{bad} is a protocol nonce, or a ciphertext $\mathrm{enc}(\dots, N)$ where N is a protocol nonces, or a signature, or a few other similar cases. (Visibility is a purely syntactic condition and essentially means that t_{bad} is not protected by an honestly generated encryption.)

Now we can conclude Dolev-Yaoness of Sim_f: If it does not hold, Sim_f sends a term $t = \tau(m)$ where m was sent by the adversary A. Then t has a visible subterm t_{bad}. Visibility implies that the recursive computation of $\tau(m)$ had a subinvocation $\tau(m_{bad}) = t_{bad}$. For each possible case of t_{bad} we derive a contradiction. For example, if t_{bad} is a protocol nonce, then

$\beta(t_{bad})$ was never invoked (since $S \not\vdash t_{bad}$) and thus $m_{bad} = r_N$ was guessed by the simulator without ever accessing r_N which can happen only with negligible probability. Other cases are excluded, e.g., by the unforgeability of the signature scheme and by the unpredictability of encryptions.

Thus, Sim_f is Dolev-Yao, hence Sim is indistinguishable and Dolev-Yao. Computational soundness follows.

Computational Soundness Based on Extraction ZK

We now describe how computational soundness for zero-knowledge proofs was shown in [21], based on the strong assumption of extraction zero-knowledge. Our presentation strongly deviates from the details of the proof in [21]; we explain what their proof would be like if recast in the CoSP framework. This makes it easier to compare the proof to our proof and the proof described in the preceding section.

Extraction zero-knowledge is a strong property that guarantees the following: It is not possible to distinguish a prover-oracle from the a simulator-oracle, even when given access to an extraction oracle that extracts the witnesses from arbitrary proofs except the ones produced by the prover/simulator-oracle. Notice that there is a strong analogy to IND-CCA secure encryption. The prover-oracle corresponds to an encryption-oracle, the witness to the plaintext, the simulator-oracle to a fake encryption-oracle encrypting zero-plaintexts, and the extractor-oracle to a decryption-oracle.

This analogy allows us to adapt the idea for proving computational soundness of encryptions to the case of ZK proofs. As in the proof without ZK proofs, we construct a simulator Sim with translation functions τ and β. We extend β and τ to deal with ZK proofs in the obvious way: $\beta(\mathrm{ZK}(\mathrm{crs}(N), t, t', N)) := A_{\mathrm{ZK}}(A_{\mathrm{crs}}(r_N), \beta(t), \beta(t'), r_N))$ and $\beta(\mathrm{ZK}(\ldots, N^m)) := m$. When parsing a ZK proof z, we set $\tau(z) := t$ if t was earlier output by $\beta(z)$. Otherwise, we obtain the statement x from z by applying A_{getPub}, we identify from which r_N the CRS used in z was computed, and we get the witness w by applying the extraction algorithm. (If z was produced with respect to a CRS that was not produced by the simulator, we set $\tau(z) := \mathrm{garbageZK}(\ldots)$.). Finally, $\tau(z)$ returns $\mathrm{ZK}(\mathrm{crs}(N), \tau(x), \tau(w), N^z)$.

The proof of indistinguishability is analogous to that without ZK proofs, except that we use the extractability property of the proof system to make sure that the simulator does not abort when invoking the extraction algorithm while trying to parse a ZK proof z in $\tau(z)$. Notice that plain extractability (as opposed to simulation-sound extractability) can be used here since we do not use a ZK-simulator in the construction of Sim.

To prove Dolev-Yaoness, we proceed as in the case without ZK proofs, except that we introduce three more intermediate simulators Sim_1, Sim_2, and Sim_3. (See Figure 2.5.) In Sim_1, we invoke a prover-oracle $\mathcal{O}^N_{\mathrm{ZK}}$ with statement $\beta(t)$ and witness $\beta(t')$ in $\beta(\mathrm{ZK}(\mathrm{crs}(N), t, t', M))$ instead of computing $A_{\mathrm{ZK}}(A_{\mathrm{crs}}(r_N), \beta(t), \beta(t'), r_M)$. (This is analogous to Sim_4 above.) $\mathcal{O}^N_{\mathrm{ZK}}$ aborts if the witness is not valid.

In Sim_2, we replace the prover-oracle $\mathcal{O}^N_{\mathrm{ZK}}$ by a ZK-simulator-oracle $\mathcal{O}^N_{\mathrm{sim}}$. That oracle runs the ZK-simulator (after checking that the witness is valid). Extraction zero-knowledge

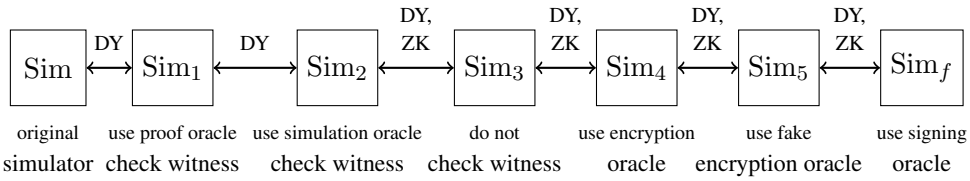

Figure 2.5: Simulators used in the proof. An arrow marked DY means Dolev-Yaoness is propagated from one simulator to the other. An arrow marked ZK means ZK-breaks are propagated (needed in section 2.4.2).

guarantees that this replacement leads to an indistinguishable hybrid execution. (We need that the witness is checked before running the simulator because extraction zero-knowledge gives no guarantees in the case of invalid witnesses, even if the witness is not actually used by the ZK-simulator.)

Finally, in Sim_3 we modify the ZK-simulator-oracle $\mathcal{O}^N_{\mathrm{sim}}$ such that it does not check the witness any more. A protocol condition guarantees that this check would succeed anyway, so this change leads to an indistinguishable hybrid execution. Furthermore, since witnesses given to $\mathcal{O}^N_{\mathrm{sim}}$ are never used, we can further change $\beta(\mathrm{ZK}(\mathrm{crs}(N), t, t', M))$ to never even compute the witness $\beta(t')$.

The rest of the proof is analogous to the case without ZK proofs. I.e., we continue with the simulator $\mathrm{Sim}_3, \mathrm{Sim}_4, \mathrm{Sim}_f$ as described there and show that Sim_f is Dolev-Yao. When showing that in Sim_f, $\beta(t)$ is only invoked when $S \vdash t$, we also make use of the fact that $\beta(\mathrm{ZK}(\mathrm{crs}(N), t, t', M))$ does not descend into the witness $\beta(t')$ any more.

Note that this computational soundness proof crucially depends on the extraction ZK property. We need to use the extractor in the construction of τ, and we need to replace the prover-oracle by a ZK-simulator-oracle in order to make sure that β does not descend into witnesses. And that replacement takes place in a setting where the parsing function τ and thus the extractor is used.

Proof Idea

We now describe the idea of our approach that allows us to get rid of extraction ZK. As explained in section 2.4.2, we cannot use the extractor as part of the parsing function τ if we do not have extraction ZK. However, the following observation shows that we might not need to run the extractor: Although in the computational setting, the only way to compute a witness is to extract it (unless the relation is trivial), in the symbolic setting, given a symbolic statement x, it is typically easy to compute a corresponding symbolic witness w. (E.g., when proving the knowledge of a secret key that decrypts a term $c = \mathrm{enc}(\mathrm{ek}(N), t, M)$, then the witness is $\mathrm{dk}(N)$ which can just be read off c.) We stress that we do not claim that the witness can be deduced (in the sense of $\vdash$) from x, only that its symbolic representation can be efficiently computed from the statement.

Thus, for an adversary-generated proof z with CRS $A_{\mathrm{crs}}(r_N)$ and statement m_x and that

passes verification, we define $\tau(z)$ as follows: We run $w := \mathbf{SymbExtr}(S, x)$ and return $\tau(z) := \mathrm{ZK}(\mathrm{crs}(N), x, w, N^z)$. Here S is the list of terms send by the protocol so far, $\mathbf{SymbExtr}(S, x)$ denotes an arbitrary witness w satisfying the following two conditions: w is a valid witness for x (i.e., $(x, w) \in R_{\mathrm{adv}}^{\mathrm{sym}}$) and $S \vdash w$. (Our result assumes that $w = \mathbf{SymbExtr}(S, x)$ is efficiently computable whenever w exists, this will be the case for most natural relations.)

The condition $S \vdash w$ is necessary since otherwise the simulator Sim would produce a proof that the adversary could not have deduced (since he could not have deduced the witness), and thus the simulator would not be Dolev-Yao.

Assume for the moment that $\mathbf{SymbExtr}(S, x)$ always succeeds (i.e., in the hybrid execution, there always is a w with $(x, w) \in R_{\mathrm{adv}}^{\mathrm{sym}}$ and $S \vdash w$). In this case, we can finish the proof analogously to that in section 2.4.2: Indistinguishability of Sim follows by carefully checking all cases, and the Dolev-Yaoness by the same sequence of simulators as in section 2.4.2. We do not need extraction zero-knowledge when going from Sim_1 to Sim_2, though, because in Sim_1, no extractor is used (we use symbolic extraction instead). Thus the zero-knowledge property is sufficient instead of extraction zero-knowledge.

But how to show that $\mathbf{SymbExtr}(S, x)$ always succeeds? Two things might go wrong. First, no valid witness w with $(x, w) \in R_{\mathrm{adv}}^{\mathrm{sym}}$ might exist. Note that the extractability property only guarantees that computationally, a valid witness for the computational statement m_x exists. This does not necessarily imply that translating that witness into a term (e.g., using τ) yields a valid symbolic witness. Second, there might be a valid witness w, but that witness is not deducable ($S \nvdash w$). Again, extractability only guarantees that the adversary "knows" a witness in the computational setting, this does not imply deducability in the symbolic setting.

In essence, to show that $\mathbf{SymbExtr}(S, x)$ succeeds, we need a kind of computational soundness result: Whenever computationally, the adversary knows a valid witness, then symbolically, the adversary knows a valid witness. This seems problematic, because it seems that we need to use a computational soundness result within our proof of computational soundness – a seeming circularity. Fortunately, this circularity can be resolved: The fact that $\mathbf{SymbExtr}(S, x)$ succeeds is used only when proving that Sim is indistinguishable (i.e., mimics the computational execution well). But the fact that $\mathbf{SymbExtr}(S, x)$ succeeds does not relate to the computational execution at all. In fact, it turns out to be closely related to the Dolev-Yaoness and can be handled in the same proof. And that proof does not use the fact that symbolic extraction succeeds.

Proof Overview

We now give a more detailed walk-through through our proof. This exposition can also be seen as a guide through the full proof in appendix A.1.

The simulator. The first step is to define the simulator Sim, i.e., the translation function β and τ. Here, we only present the parts of the definition related to ZK proofs (the first matching rule counts):

1. $\tau(z) := \mathrm{crs}(N)$ if $z = A_{\mathrm{crs}}(r_N)$ for some N that occurred in a subterm of the form $\mathrm{crs}(N)$ before

2. $\tau(z) := \mathrm{crs}(N^z)$ if z is of type common reference string

3. $\tau(z) := \mathrm{ZK}(\mathrm{crs}(N_1), t_1, t_2, N_2)$ if z has earlier been output by $\beta(\mathrm{ZK}(\mathrm{crs}(N_1), t_1, t_2, N_2))$ for some $N_1, N_2 \in \mathbf{N}_P$

4. $\tau(z) := \mathrm{ZK}(\mathrm{crs}(N), x, w, N^z)$ if z is of type zero-knowledge proof and $\tau(z)$ was computed earlier and has output $\mathrm{ZK}(\mathrm{crs}(N), x, w, N^z)$

5. $\tau(z) := \mathrm{ZK}(\mathrm{crs}(N), x, w, N^z)$ if z is of type zero-knowledge proof, $\tau(A_{\mathrm{crsof}}(z)) = \mathrm{crs}(N)$ for some $N \in \mathbf{N}_P$, $A_{\mathrm{verify}_{\mathrm{ZK}}}(A_{\mathrm{crsof}}(z), z) = z$, $m_x := A_{\mathrm{getPub}}(z) \neq \bot$, $x := \tau(m_x) \neq \bot$ and $w := \mathbf{SymbExtr}(S, x)$ where S is the set of terms sent to the adversary so far.

6. $\tau(z) := \mathrm{garbageZK}(c, x, N^z)$ if z is of type zero-knowledge proof, $c := \tau(A_{\mathrm{crsof}}(z))$ and $x := \tau(A_{\mathrm{getPub}}(z))$.

7. $\beta(\mathrm{crs}(N)) := A_{\mathrm{crs}}(r_N)$ if $N \in \mathbf{N}_P$

8. $\beta(\mathrm{crs}(N^c)) := c$

9. $\beta(\mathrm{ZK}(\mathrm{crs}(N_1), t_1, t_2, N_2)) := A_{\mathrm{ZK}}(A_{\mathrm{crs}}(r_{N_1}), \beta(t_1), \beta(t_2), r_{N_2})$ if $N_1, N_2 \in \mathbf{N}_P$

10. $\beta(\mathrm{ZK}(\mathrm{crs}(t_0), t_1, t_2, N^s)) := s$

11. $\beta(\mathrm{garbageZK}(t_1, t_2, N^z)) := z$

Here $\mathbf{SymbExtr}(S, x)$ returns a witness w with $(x, w) \in R_{\mathrm{adv}}^{\mathrm{sym}}$ and $S \vdash w$ if such w exists, and $\bot$ otherwise. A key point is what to do when $\mathbf{SymbExtr}(S, x)$ fails. We will later show that this happens with negligible probability only, but for now we need to specify the behavior in this case:

When $\mathbf{SymbExtr}(S, x)$ returns $\bot$ in the rule 5), we say an *extraction failure* occurred. In this case, the simulator runs the extractor (using the extraction trapdoor corresponding to $A_{\mathrm{crs}}(r_N)$) to get a (computational) witness m_w for m_x. Then Sim computes $w := \tau^*(m_w)$ where τ^* is defined like τ, except that the rule 5) is dropped (hence τ^* will map an adversary-generated ZK-proof always to a $\mathrm{garbageZK}$-term). Then the simulator aborts. If $(x, w) \notin R_{\mathrm{adv}}^{\mathrm{sym}}$, we say a *ZK-break* occurred.

The reader may wonder why we let the simulator compute a symbolic witness w in case of an extraction failure even though w is never used. The reason is that we later show that this w always has $(x, w) \in R_{\mathrm{adv}}^{\mathrm{sym}}$ and $S \vdash w$, which contradicts the fact that we get an extraction failure in the first place. The reason for using τ^* instead of τ is that we have to avoid getting extraction failures within extraction failures.

The sequence of simulators. As in section 2.4.2, we construct a sequence of simulators (see Figure 2.5): Sim_1 differs from Sim by using a prover-oracle for constructing ZK-proofs instead of invoking A_{ZK} directly in β. We also use that oracle to obtain the CRS, and for extracting m_w after an extraction failure. In Sim_2, we replace the prover-oracle by a ZK-simulator-oracle. If the oracle is invoked with an invalid witness, it aborts instead of running the ZK-simulator. In Sim_3, we still use a ZK-simulator-oracle, but we do not check the witness first. Thus $\beta(w)$ is not invoked on witnesses any more in rule 9). Sim_4 replaces A_{enc} and A_{dec} by calls to an encryption oracle, Sim_5 replaces that encryption oracle by a fake encryption oracle using zero-plaintexts, and Sim_f finally uses a signing oracle instead of A_{sig}.

We can now show that Sim_f is Dolev-Yao. The proof of this fact is analogous to the case the proof sketched in section 2.4.2. We even show something slightly stronger, namely that neither τ nor τ^* outputs an undeducable term:

Lemma 2.4.2 (Sim_f is Dolev-Yao)**.** *For any invocation $t := \tau(m)$ or $t := \tau^*(m)$, we have $S \vdash t$ where S are the terms sent to the simulator so far. In particular, Sim_f is Dolev-Yao.*

As in section 2.4.2, we show Sim is Dolev-Yao iff Sim_f is Dolev-Yao. We will also need preservation of the property that ZK-breaks occur with negligible probability.

Lemma 2.4.3 (Preservation of simulator-properties)**.** *The properties of the simulators are preserved, more precisely the following statements hold:*

- Sim *is Dolev-Yao iff* Sim_f *is.*
- *In the hybrid execution of* Sim *extraction failures occur with negligible probability iff the same holds for* Sim_f*.*
- *In the hybrid execution of* Sim_2 *(not* Sim*!) ZK-breaks occur with negligible probability iff the same holds for* Sim_f*.*

Dolev-Yaoness, extraction failures, and ZK-breaks carry over from Sim_3 to Sim_4 and from Sim_5 to Sim_f because the randomness used in encrypting and signing is not re-used by protocol condition 3. (Notice that randomness might have occurred within a witness, but due to the change in Sim_3, we do not invoke $\beta(w)$ on witnesses any more.) Dolev-Yaoness, extraction failures, and ZK-breaks carry over from Sim_4 to Sim_5 due to the IND-CCA property. Dolev-Yaoness and extraction failures carry over from Sim to Sim_1 because the randomness used for constructing ZK-proofs is not reused by protocol condition 3.

Furthermore, Dolev-Yaoness and extraction failures carry over from Sim_1 to Sim_2 because of the zero-knowledge property of the proof system. There is a subtlety here: Sim_1 does use the extractor (namely after an extraction failure). So usually, we would not be allowed to apply the zero-knowledge property (we would need extraction ZK). But fortunately, after an extraction failure, no terms are sent by the simulator. Thus, anything that happens after an extraction failure has no impact on whether the simulator is Dolev-Yao or not. Thus, for analyzing whether Dolev-Yaoness carries over from Sim_1 to Sim_2, we can assume that those simulators abort

directly after incurring an extraction failure (without invoking the extractor afterwards). Then no extractions occur in the simulator, and we can use the zero-knowledge property. Analogously, extraction failures carry over from Sim_1 to Sim_2.

Notice that we cannot use the same trick to show that ZK-breaks carry over from Sim_1 to Sim_2: Whether ZK-breaks occur is determined only after the invocation of the extractor. Fortunately, we only need that ZK-breaks carry over from Sim_2 to Sim_f.

To show Theorem 2.4.3, it remains to show that Dolev-Yaoness, extraction failures, and ZK-breaks carry over from Sim_2 to Sim_3. The only difference between these simulators is that Sim_3 does not check whether the witness m_w given to the ZK-simulation-oracle is valid (i.e., $(\beta(t_1), \beta(t_2)) \in R^{\mathrm{comp}}_{\mathrm{honest}}$ in rule 9). Thus, to conclude the proof of Theorem 2.4.3, we need to show that the probability that the ZK-simulation-oracle is called with an invalid witness is negligible.

No invalid witnesses. To show that the ZK-simulation-oracle is only called by Sim_2 with valid computational witnesses $\beta(t_1)$, we need to show two things:

Lemma 2.4.4 (No invalid symbolic witnesses)**.** *If* Sim_3 *is Dolev-Yao, then in rule 9), we have* $(t_1, t_2) \in R^{\mathrm{sym}}_{\mathrm{honest}}$ *with overwhelming probability. The same holds for* Sim*.*

Lemma 2.4.5 (Relating the relations, part 1)**.** *In an execution of* Sim_3 *the following holds with overwhelming probability: if* $(x, w) \in R^{\mathrm{sym}}_{\mathrm{honest}}$ *then* $(\beta(x), \beta(w)) \in R^{\mathrm{comp}}_{\mathrm{honest}}$*. The same holds for* Sim*.*

Once we have these lemmas, Theorem 2.4.3 follows: We know from Theorem 2.4.2 that Sim_f is Dolev-Yao. We have already shown that this property carries over to Sim_3. Thus by Lemmas 2.4.4 and 2.4.5, $(\beta(t_1), \beta(t_2)) \in R^{\mathrm{comp}}_{\mathrm{honest}}$ in rule 9).

To show Theorem 2.4.4, we observe the following: If the simulator sends only terms that are deducible (i.e., that a symbolic adversary could also have sent), then in the hybrid execution, no execution trace occurs that could not have occurred in the symbolic execution either. By protocol condition 10, in a symbolic execution, $(t_1, t_2) \in R^{\mathrm{sym}}_{\mathrm{honest}}$ whenever the protocol constructs an $\mathrm{ZK}(\mathrm{crs}(N), t_1, t_2, M)$-term. Since rule 9) only applies to such protocol-generated terms (ZK-terms from τ have $M \in \mathbf{N}_E$), it follows that $(t_1, t_2) \in R^{\mathrm{sym}}_{\mathrm{honest}}$ in rule 9). Theorem 2.4.4 follows. Theorem 2.4.5 follows because we required that $R^{\mathrm{comp}}_{\mathrm{honest}}, R^{\mathrm{comp}}_{\mathrm{adv}}$ implement $R^{\mathrm{sym}}_{\mathrm{adv}}$ with usage restriction $R^{\mathrm{sym}}_{\mathrm{honest}}$; 2.4.2 was designed to make Theorem 2.4.5 true.

Thus, Lemmas 2.4.4 and 2.4.5 hold, thus Theorem 2.4.3 follows. Since Sim_f is Dolev-Yao by Theorem 2.4.2, it follows with Theorem 2.4.3 that Sim is Dolev-Yao. It remains to show that Sim is indistinguishable.

Indistinguishability of Sim**.** As described stated above, to show indistinguishability of Sim, the main subproof is to show (a) that $\beta(F(t_1, \ldots, t_n)) = A_F(\beta(t_1), \ldots, \beta(t_n))$ when the protocol computes $F(t_1, \ldots, t_n)$. And, of course, we need (b) that the simulator does not abort. The proof of (a) is, as before, done by careful checking of all cases. The only interesting

case is $F = \mathrm{verify}_{\mathrm{ZK}}$. Here we need that an honestly-generated ZK proof with statement x and witness w passes verification symbolically ($x, w \in R^{\mathrm{sym}}_{\mathrm{honest}}$) iff it passes verification computationally (($\beta(x), \beta(w) \in R^{\mathrm{comp}}_{\mathrm{honest}}$). Fortunately, we have already derived all needed facts: By Lemmas 2.4.2 and 2.4.4, $(x, w) \in R^{\mathrm{sym}}_{\mathrm{honest}}$ with overwhelming probability. And then by Theorem 2.4.5, $(\beta(x), \beta(w)) \in R^{\mathrm{sym}}_{\mathrm{honest}}$.

To show (b), we need to show that extraction failures occur with negligible probability. The approach for this is a bit roundabout, we first analyze Sim_2:

Lemma 2.4.6 (No ZK-breaks)**.** *In the hybrid execution of* Sim_2*, ZK-breaks occur with negligible probability.*

To show this, we use the simulation-sound extractability property of the proof system to show that the values m_x, m_w extracted by the extractor after an extraction failure satisfy $(m_x, m_w) \in R^{\mathrm{comp}}_{\mathrm{adv}}$. And then it follows that $(x, w) \in R^{\mathrm{sym}}_{\mathrm{adv}}$ with $x := \tau(m_x)$, $w := \tau^*(m_w)$ by the converse of Theorem 2.4.5:

Lemma 2.4.7 (Relating the relations, part 2)**.** *In an execution of* Sim_2 *the following holds with overwhelming probability: if* $(m_x, m_w) \in R^{\mathrm{comp}}_{\mathrm{adv}}$ *then* $(\tau(m_x), \tau^*(m_w)) \in R^{\mathrm{sym}}_{\mathrm{adv}}$.

Thus Theorem 2.4.6 is shown. From this, with Theorem 2.4.3 we get that ZK-breaks occur with negligible probability also for Sim_f. Based on this fact, we can show the following lemma:

Lemma 2.4.8 (No extraction failures)**.** *In the hybrid execution of* Sim_f*, extraction failures occur with negligible probability.*

To see this, we use that ZK-breaks only occur with negligible probability in the execution of Sim_f. Thus, by definition of ZK-breaks, this means that $(x, w) \in R^{\mathrm{sym}}_{\mathrm{adv}}$ for the terms $x := \tau(m_x)$ and $w := \tau(m_w)$ computed after the extraction failure. Furthermore, by Theorem 2.4.2, it follows that $S \vdash w$. But then, by definition, $\mathbf{SymbExtr}(x, S)$ would have output a w or another witness, but not $\bot$. Thus the extraction failure would not have occurred. This shows Theorem 2.4.8.

Finally, from Lemmas 2.4.6 and 2.4.3 we get that extraction failures occur with negligible probability in the execution of Sim, too. Thus property (b) also holds, thus we have shown Sim to be indistinguishable.

Notice that the roundabout way through Sim_2 and Sim_f to show that extraction failures occur with negligible probability in the execution of Sim is necessary: We cannot directly show Theorem 2.4.6 for Sim_f because Sim_f uses the simulator to prove untrue statements (e.g., it may prove that a ciphertext contains a certain value, but since we use a fake encryption oracle, that ciphertext actually contains a zero-plaintext), so simulation-sound extractability cannot be applied. Also, we cannot use the fact $S \vdash \tau^*(x)$ directly on Sim because this fact cannot be propagated from Sim_f to Sim (since τ^* is executed after the extractor is used, we would need extraction ZK to bridge from Sim_2 to Sim_1).

Concluding the proof. We have shown that Sim is Dolev-Yao and indistinguishable. From [14] we then immediately get Theorem 2.4.1.

2.5 Protocol Verification using the applied π-calculus

In [14] it was shown how to use computational soundness results in the CoSP framework (such as our result) and derive computational soundness results for a dialect of the applied π-calculus (see [14] for a description of the calculus together with semantics for symbolic and computational execution). Basically, they present a generic transformation that translates a process in the applied π-calculus into a CoSP process (generic means that the transformation works for any symbolic model, including the one presented here). Thus, all that needs to be done to get a computational soundness result for zero-knowledge proofs in the applied π-calculus is to write down what conditions a process needs to satisfy such that the translated process satisfies the protocol conditions (listed in the appendix):

Definition 2.5.1 (Valid processes)**.** *A process $\tilde{P}$ in the applied π-calculus is* valid *if it satisfies the following two properties:*

(i) The process $\tilde{P}$ matches the following grammar: Let x, x_d, x_s, x_c stand for different sets of variables (general purpose, decryption key, signing key, and CRS variables). Let a, r, r_z stand for three sets of names (general purpose, randomness, and ZK randomness names). $\tilde{M}, \tilde{N} ::= x \mid x_c \mid a \mid \mathrm{pair}(\tilde{M}, \tilde{N}) \mid \tilde{S}$ and $\tilde{S} ::= \mathit{string}_0(\tilde{S}) \mid \mathit{string}_1(\tilde{S}) \mid \mathit{empty}$, and let $\hat{D}$ be an arbitrary term consisting of constructors, destructors, variables, and names except r_z and $\tilde{D} ::= \tilde{M} \mid \mathrm{isek}(\tilde{D}) \mid \mathrm{isenc}(\tilde{D}) \mid \mathrm{dec}(x_d, \tilde{D}) \mid \mathrm{fst}(\tilde{D}) \mid \mathrm{snd}(\tilde{D}) \mid \mathrm{ekof}(\tilde{D}) \mid \mathrm{equals}(\tilde{D}, \tilde{D}) \mid \mathrm{isvk}(\tilde{D}) \mid \mathrm{issig}(\tilde{D}) \mid \mathrm{verify}_{\mathrm{sig}}(\tilde{D}, \tilde{D}) \mid \mathrm{vkof}(\tilde{D}) \mid \mathrm{iscrs}(\tilde{D}) \mid \mathrm{crsof}(\tilde{D}) \mid \mathrm{verify}_{\mathrm{ZK}}(x_c, \tilde{D}) \mid \mathrm{iszk}(\tilde{D}) \mid \mathrm{getPub}(\tilde{D}) \mid \mathit{unstring}_0(\tilde{D}) \mid \mathit{unstring}_1(\tilde{D})$ and

$$\begin{aligned}\tilde{P},\tilde{Q} ::= {} & \overline{\tilde{M}}\langle\tilde{N}\rangle.\tilde{P} \mid \tilde{M}(x).\tilde{P} \mid 0 \mid (\tilde{P} \mid \tilde{Q}) \mid !\tilde{P} \mid \nu a.\tilde{P} \mid \mathit{let}\ x = \tilde{D}\ \mathit{in}\ \tilde{P}\ \mathit{else}\ \tilde{Q} \mid \\ & \mathit{event}(e).\tilde{P} \mid \nu r.\mathit{let}\ x = \mathrm{ek}(r)\ \mathit{in}\ \mathit{let}\ x_d = \mathrm{dk}(r)\ \mathit{in}\ \tilde{P} \mid \\ & \nu r.\mathit{let}\ x = \mathrm{enc}(\mathrm{isek}(\tilde{D}_1), \tilde{D}_2, r)\ \mathit{in}\ \tilde{P}\ \mathit{else}\ \tilde{Q} \mid \\ & \nu r.\mathit{let}\ x = \mathrm{vk}(r)\ \mathit{in}\ \mathit{let}\ x_s = \mathrm{sk}(r)\ \mathit{in}\ \tilde{P} \mid \\ & \nu r.\mathit{let}\ x = \mathrm{sig}(x_s, \tilde{D}_1, r)\ \mathit{in}\ \tilde{P}\ \mathit{else}\ \tilde{Q} \mid \\ & \nu r.\mathit{let}\ x_c = \mathrm{crs}(r_z)\ \mathit{in}\ \tilde{P} \mid \nu r.\mathit{event}\ \mathit{zk}.\mathit{let}\ x = \mathrm{ZK}(x_c, \tilde{D}_1, \hat{D}, r_z)\ \mathit{in}\ \tilde{P}\ \mathit{else}\ \tilde{Q}\end{aligned}$$

(Note that in each of the last six production rules, several occurrences of r or r_z denote the same name.)

(ii) For any process Q that does not contain events, if $\tilde{P}|Q \to^* E[\mathit{event}\ \mathit{zk}.\mathit{let}\ x = \mathrm{ZK}(t_1, t_2, t_3, t_4)\ \mathit{in}\ P_1\ \mathit{else}\ P_2]$ with an evaluation context E, then $(t_2\eta, t_3\eta) \in R^{\mathrm{sym}}_{\mathrm{honest}}$ for any bijective mapping η from names to nonces.

◇

Analogous to [14, Thm. 4], we obtain:

Theorem 2.5.1 (Computational soundness of ZK proofs)**.** *Let $\tilde{P}$ be a closed valid process and A_F a computational implementation satisfying the implementation conditions from subsection 2.4.1. Then for any π-trace property*[6] *$\wp$, if P symbolically satisfies $\wp$, then P computationally satisfies $\wp$.*

In the Appendix 2.6, we prove that this relation satisfies definition 2.4.2, thus the abstraction is sound. We express this protocol in the applied π-calculus:

$$
\begin{aligned}
P := {} & \nu r_A.\mathit{let}\ \mathrm{ek}_A = \mathrm{ek}(r_A)\ \mathit{in}\ \mathit{let}\ \mathrm{dk}_A = \mathrm{dk}(r_A)\ \mathit{in}\ \nu r_B.\mathit{let}\ \mathrm{ek}_B = \mathrm{ek}(r_B)\ \mathit{in}\\
& \mathit{let}\ \mathrm{dk}_B = \mathrm{dk}(r_B)\ \mathit{in} \nu r_C.\mathit{let}\ \mathrm{crs} = \mathrm{crs}(r_C)\ \mathit{in} \overline{ch}\langle(\mathrm{ek}_A, \mathrm{ek}_B, \mathrm{crs})\rangle.(!A|!B)\\
A := {} & \nu N_1.\mathit{event}\ \mathit{begin}_A(N_1).\nu r_1.\mathit{let}\ m_1 = \mathrm{enc}(\mathrm{ek}_B, N_1, r_1)\ \mathit{in}\ \overline{ch}\langle m_1\rangle.ch(m_2).\\
& \mathit{let}\ \mathit{stmt} = \mathrm{getPub}(\mathrm{verify}_{\mathrm{ZK}}(\mathrm{crs}, m_2))\ \mathit{in}\ \mathit{if}\ \ \mathrm{snd}(\mathit{stmt}) = m_1\ \mathit{then}\\
& \mathit{let}\ N_2 = \mathrm{dec}(\mathrm{dk}_A, \mathrm{fst}(\mathit{stmt}))\ \mathit{in}\ \nu r_4.\mathit{let}\ m_3 = \mathrm{enc}(\mathrm{ek}_B, N_2, r_4)\ \mathit{in}\\
& \overline{ch}\langle m_3\rangle.\mathit{event}\ \mathit{end}_A(N_1, N_2)\\
B := {} & ch(m_1).\mathit{let}\ N_1 = \mathrm{dec}(\mathrm{dk}_B, m_1)\ \mathit{in}\ \ \nu N_2.\mathit{event}\ \mathit{begin}_B(N_2).\ \nu r_2.\\
& \mathit{let}\ c = \mathrm{enc}(\mathrm{ek}_A, N_2, r_2)\ \mathit{in}\ \ \nu r_3.\mathit{event}\ zk.\mathit{let}\ m_2 = \mathrm{ZK}(\mathrm{crs}, (c, m_1), \mathrm{dk}_B, r_3)\ \mathit{in}\\
& \overline{ch}\langle m_2\rangle.ch(m_3).\mathit{if}\ N_2 = \mathrm{dec}(\mathrm{dk}_B, m_3)\ \mathit{then}\ \mathit{event}\ \mathit{end}_B(N_1, N_2)
\end{aligned}
$$

This protocol can be directly encoded in ProVerif. The definition of the destructor $\mathrm{verify}_{\mathrm{ZK}}$ depends on the relation $R_{\mathrm{adv}}^{\mathrm{sym}}$, we can encode it in ProVerif as

reduc verifyZK(crs(t_1), zk(crs(t_1),(c,enc(ek(r_1),x,r_2)),dk(r_1),t_4))
= zk(crs(t_1),(c,enc(ek(r_1),x,r_2)),dk(r_1),t_4);
verifyZK(crs(t_1), zk(crs(t_1),(ciph,garbageEnc(t_2,t_3)),t_4,t_5))
= zk(crs(t_1),(ciph,garbageEnc(t_2,t_3)),t_4,t_5).

ProVerif can automatically show that P symbolically satisfies the trace properties $\mathit{end}_A(x, y) \Rightarrow \mathit{begin}_B(y)$ and $\mathit{end}_B(x, y) \Rightarrow \mathit{begin}_A(x)$. These trace properties state that each party can not end its protocol execution without communicating with the other one (the other party has to begin before the end). Hence the parties authenticate to each other. To show that P also computationally satisfies that trace property, we need to show that P is valid. P satisfies the syntactic condition (2.5.1(i)). To check that a process P satisfies the dynamic condition (ii), we use ProVerif again: We replace every occurrence of $P' = \mathit{let}\ x = \mathrm{ZK}(t_1, t_2, t_3, t_4)\ \mathit{in}\ P_1\ \mathit{else}\ P_2$ by $\mathit{let}\ x' = \mathit{checkzk}(t_2, t_3)\ \mathit{in}\ P'\ \mathit{else}\ \mathit{event}\ \mathit{badzk}$ where *checkzk* is a destructor that checks if its arguments are in $R_{\mathrm{honest}}^{\mathrm{sym}}$:

reduc checkzk((c,enc(ek(r_1),x,r_2)),dk(r_1)) = empty

ProVerif automatically shows that the event *badzk* does not occur. It follows that P is valid.

[6] A π-trace property is essentially a prefix-closed set of sequences of events that are allowed to occur. See [14] for a precise definition and for the definition of "symbolically/computationally satisfying" a π-trace property.

2.6 Example relations

In this section briefly give some examples for symbolic relations and their implementation. Then we prove that these relations satisfy definition 2.4.2, i.e. that the computational relations actually implement the symbolic ones. In both examples, we make use of the fact that the all symbolic relations are implicitly restricted to $\mathbf{T}$, and all computational ones to non-$\perp$.

Valid ciphertexts First, we consider the example of proving that a ciphertext is valid using the randomness as witness. The relations are defined as follows:

$$\begin{aligned} R^{\mathrm{sym}}_{\mathrm{honest}} &:= \{((\mathrm{enc}(k,m,r),k,m),r) : k,m \in \mathbf{T}, r \in \mathbf{N}_P\} \\ R^{\mathrm{sym}}_{\mathrm{adv}} &:= \{((\mathrm{enc}(k,m,r),k,m),r^*) : k,m,r^* \in \mathbf{T}, r \in \mathbf{N}\} \\ R^{\mathrm{comp}}_{\mathrm{honest}} &:= \{((A_{\mathrm{enc}}(k,m,r),k,m),r) : k,m,r \in \{0,1\}^*\} \\ R^{\mathrm{comp}}_{\mathrm{adv}} &:= \{((A_{\mathrm{enc}}(k,m,r),k,m),r^*) : k,m,r,r^* \in \{0,1\}^*\} \end{aligned}$$

For the proof, we need two additional requirements on the implementation. First, we need that A_{enc} in injective in the first two arguments. For the first argument, the encryption key, this can be achieved by concatenating it to the encryption. The second one is only implied by the IND-CCA property if the first argument is indeed of type encryption key, but we need it for all bitstrings. Finally, we require that the encoding of encryption keys is dense, i.e. that for every bitstring of type encryption key, there is a corresponding decryption key. Summarized, this leads to the following lemma.

Lemma 2.6.1. *If, in addition to the implementation conditions, it holds:*

1. *For all $k,k',m,m',r \in \{0,1\}^*$ $A_{\mathrm{enc}}(k,m,r) = A_{\mathrm{enc}}(k,m',r)$ implies $m = m'$ and $A_{\mathrm{enc}}(k,m,r) = A_{\mathrm{enc}}(k',m,r)$ implies $k = k'$*
2. *For all $k \in \{0,1\}^*$ of type encryption key, there is a $d \in \{0,1\}^*$ such that $p(d) = k$ according to implementation condition 29.*

Then follows that $R^{\mathrm{comp}}_{\mathrm{honest}}, R^{\mathrm{comp}}_{\mathrm{adv}}$ implement $R^{\mathrm{sym}}_{\mathrm{adv}}$ with usage restriction $R^{\mathrm{sym}}_{\mathrm{honest}}$.

Proof. Fix a consistent environment η and terms $x,w \in \mathbf{T}$.

We first show that if $(x,w) \in R^{\mathrm{sym}}_{\mathrm{honest}}$ and $\mathrm{img}_\eta(x) \neq \perp \neq \mathrm{img}_\eta(w)$, then $(\mathrm{img}_\eta(x), \mathrm{img}_\eta(w)) \in R^{\mathrm{comp}}_{\mathrm{honest}}$. Thus, fix $x = (\mathrm{enc}(k,m,r),k,m)$ and $w = r \in \mathbf{N}_P$. Then

$$\mathrm{img}_\eta(\mathrm{enc}(k,m,r)) = A_{\mathrm{enc}}(\mathrm{img}_\eta(k), \mathrm{img}_\eta(m), \mathrm{img}_\eta(r)) =: A_{\mathrm{enc}}(k',m',r')$$

Here it is crucial that $\mathrm{img}_\eta(x) \neq \perp$ and hence $A_{\mathrm{enc}}(\mathrm{img}_\eta(k), \mathrm{img}_\eta(m), \mathrm{img}_\eta(r)) \neq \perp$. So, it follows $(\mathrm{img}_\eta(x), \mathrm{img}_\eta(w)) = ((A_{\mathrm{enc}}(k',m',r'),m',r'),r') \in R^{\mathrm{sym}}_{\mathrm{honest}}$.

Now we show that if $(\mathrm{img}_\eta(x), \mathrm{img}_\eta(w)) \in R^{\mathrm{comp}}_{\mathrm{adv}}$, then $(x, w) \in R^{\mathrm{sym}}_{\mathrm{adv}}$. Fix some $x, w \in \mathbf{T}$ with $(\mathrm{img}_\eta(x), \mathrm{img}_\eta(w)) \in R^{\mathrm{comp}}_{\mathrm{adv}}$. Then

$$\mathrm{img}_\eta(x) = (c', k', m') := (\mathrm{img}_\eta(c), \mathrm{img}_\eta(k), \mathrm{img}_\eta(m))$$

with $c' = A_{\mathrm{enc}}(k', m', r')$ for some $c, k, m \in \mathbf{T}, r' \in \{0,1\}^*$. Since $\mathrm{img}_\eta(x) = (c', k', m')$, we have $x = (c, k, m)$ by definition of img_η and injectivity of A_{pair}. By implementation condition 2, $c' = \mathrm{img}_\eta(c)$ has type ciphertext. Thus, by definition of consistent environments, $c = \mathrm{enc}(k_1, m_1, r_1)$ for some $k_1, m_1, r_1 \in \mathbf{T}$. Since $c \in \mathbf{T}$, $r_1 \in \mathbf{N}$.

First, consider the case $r_1 \in \mathbf{N}_P$. Then

$$c' = \mathrm{img}_\eta(c) = A_{\mathrm{enc}}(\mathrm{img}_\eta(k_1), \mathrm{img}_\eta(m_1), \mathrm{img}(r_1))$$

Since A_{enc} is injective in its first two arguments (follows by the assumption (1) of the lemma), we have that $\mathrm{img}_\eta(k_1) = \mathrm{img}_\eta(k)$, $\mathrm{img}_\eta(m_1) = \mathrm{img}_\eta(m)$. Thus

$$(x, w) = ((\mathrm{enc}(k, m, r_1), k, m), w) \in R^{\mathrm{sym}}_{\mathrm{adv}}$$

Now consider the case $r_1 \in \mathbf{N}_E$. Since $A_{\mathrm{ekof}}(\mathrm{img}_\eta(c)) = \mathrm{img}_\eta(k_1)$ by consistency of η, it follows that $\mathrm{img}_\eta(k_1) = \mathrm{img}_\eta(k)$. Additionally, $\mathrm{img}_\eta(k_1)$ is of type encryption key. By assumption (2) there is a corresponding decryption key $d \in \{0,1\}^*$. By consistency it follows that $\mathrm{img}_\eta(m_1) = A_{\mathrm{dec}}(d, \mathrm{img}_\eta(\mathrm{enc}(k_1, m_1, r_1))) = m' = \mathrm{img}_\eta(m)$. Thus $(\mathrm{enc}(k, m, r_1), k, m), w) \in R^{\mathrm{sym}}_{\mathrm{adv}}$.

□

Ability of decryption. In the remaining section we consider the relation used in the pi-calculus example. Basically, we use the system to prove that a party is able to decrypt a given message. The additional m' is only used for freshness.

$$\begin{aligned}
R^{\mathrm{sym}}_{\mathrm{honest}} &:= \{((m', m_1), d) : m', m_1, d \in \mathbf{T} \\
&\qquad \text{such that } \mathrm{dec}(d, m_1) \neq \bot\} \\
R^{\mathrm{sym}}_{\mathrm{adv}} &:= R^{\mathrm{sym}}_{\mathrm{honest}} \cup \{((m', m_1), d) : \\
&\qquad m_1 = \mathrm{garbageEnc}(t, M), t \in \mathbf{T}, M \in \mathbf{N}\} \\
R^{\mathrm{comp}}_{\mathrm{adv}} := R^{\mathrm{comp}}_{\mathrm{honest}} &:= \{((m', m_1), d) : m', m_1, d \in \{0,1\}^* \\
&\qquad \text{such that } A_{\mathrm{dec}}(d, m_1) \neq \bot\}
\end{aligned}$$

Additional to the implementation conditions, we require that for each encryption key, there is exactly one decryption key accepted by the decryption algorithm.

Lemma 2.6.2. *If, in addition to the implementation conditions, it holds: For all $d, d', c \in \{0,1\}^*$ it holds: If $A_{\mathrm{dec}}(d, c) \neq \bot \neq A_{\mathrm{dec}}(d', c)$ then $d' = d$. Then follows that $R^{\mathrm{comp}}_{\mathrm{honest}}, R^{\mathrm{comp}}_{\mathrm{adv}}$ implement $R^{\mathrm{sym}}_{\mathrm{adv}}$ with usage restriction $R^{\mathrm{sym}}_{\mathrm{honest}}$.*

Proof. First, we observe that by implementation conditions 4 and 13, it follows that if $c \in \{0,1\}^*$ is not of type ciphertext, then for all $d \in \{0,1\}^*$ it holds $A_{\mathrm{dec}}(d,m) = \bot$.

Fix a consistent environment η and terms $x, w \in \mathbf{T}$. We start showing that if $(x,w) \in R^{\mathrm{sym}}_{\mathrm{honest}}$ and $\mathrm{img}_\eta(x) \neq \bot \neq \mathrm{img}_\eta(w)$, then $(\mathrm{img}_\eta(x), \mathrm{img}_\eta(w)) \in R^{\mathrm{comp}}_{\mathrm{honest}}$. By definition it follows that $x = (m', m_1)$ and $w = d$. Since $\mathrm{dec}(d, m_1) \neq \bot$ it follows that $d = \mathrm{dk}(N)$ and $m_1 = \mathrm{enc}(\mathrm{ek}(N), t, M)$ for some $N, M \in \mathbf{N}$. By consistency of η it follows $A_{\mathrm{dec}}(\mathrm{img}_\eta(\mathrm{dk}(N)), \mathrm{img}_\eta(m_1)) = \mathrm{img}_\eta(t) \neq \bot$. Thus $(\mathrm{img}_\eta(x), \mathrm{img}_\eta(w))) \in R^{\mathrm{comp}}_{\mathrm{honest}}$.

Now we show that if $(\mathrm{img}_\eta(x), \mathrm{img}_\eta(w)) \in R^{\mathrm{comp}}_{\mathrm{adv}}$, then $(x,w) \in R^{\mathrm{sym}}_{\mathrm{adv}}$. By definition $\mathrm{img}_\eta(x) = (m', m'_1)$ and $\mathrm{img}_\eta(w) = d'$ such that $A_{\mathrm{dec}}(d', m'_1) \neq \bot$. Thus x is of the form (m, m_1) and w some d. As mentioned above, it follows that m'_1 is of type ciphertext, since $A_{\mathrm{dec}}(d', m'_1) \neq \bot$. Hence, by consistency of η it follows that m_1 has the form $\mathrm{enc}(\mathrm{ek}(N), t, M)$ for some $N\mathbf{N}_P$,$M \in \mathbf{N}$, $t \in \mathbf{T}$ or it has the form $\mathrm{garbageEnc}(t, N)$ for $t \in \mathbf{T}, N \in \mathbf{N}$. In the latter case, $(x,w) \in R^{\mathrm{sym}}_{\mathrm{adv}}$. In the first case, by consistency of η follows that $A_{\mathrm{dec}}(\mathrm{img}_\eta(\mathrm{dk}(N)), \mathrm{img}_\eta(\mathrm{enc}(\mathrm{ek}(N), t, M))) = \mathrm{img}_\eta(t) \neq \bot$. Thus $\mathrm{img}_\eta(w) = d' = \mathrm{img}_\eta(\mathrm{dk}(N))$ and hence $w = \mathrm{dk}(N)$ by assumption. So $(x,w) \in R^{\mathrm{sym}}_{\mathrm{adv}}$. □

2.7 An Impossibility Result for Computational Soundness of Symbolic ZK Proofs

In this section, we prove an inherent limitation of computational soundness results for ZK: If one permits ZK proofs over statements that themselves involve ZK proofs, then computational soundness cannot be shown to hold in general based on existing cryptographic definitions. Consequently, ZK proofs over statements involving ZK proofs need to be excluded in all computational soundness proofs for symbolic ZK. This is the case in the present paper (formally, Condition 28 ensures this), and it was the case already in [22] for technical reasons. The following theorem shows that this restriction is inevitable.

Theorem 2.7.1. *There is no computational sound symbolic model that has no restrictions on the symbolic relations, even if the underlying zero-knowledge proof system is simulation-extractable*

Proof. If such a model would exist, then it is sound with respect to the computational relation $R^{\mathrm{comp}}_{\mathrm{honest}} := \{(A_{\mathrm{empty}}, w) \mid \exists x : \mathbf{V}(x,w) = 1\}$. Call the proof system $(\mathbf{P}, \mathbf{V})$.

We can assume w.l.o.g. that the extraction-trapdoor contains the simulation-trapdoor, and the simulation-trapdoor contains the crs. We may also assume w.l.o.g. that $\mathbf{V}(x, \mathbf{S}(x)) = 1$ holds always (even for false x). Considering the definition of weakly symbolically-sound ZK-proofs, it is easy to see that, whenever the adversary gets the simulation-trapdoor, it also gets the crs. Analogue, if it gets the extraction-trapdoor, it also gets the crs and the simulation-trapdoor. So we can assume the containment as describe above without changing the properties of our ZK-proof system.

We can achieve the property that $\mathbf{V}(x, \mathbf{S}(x)) = 1$ for all x as follows: Let $\mathrm{simtd}_{\mathrm{vk}}, \mathrm{simtd}_{\mathrm{sk}}$ be a key-pair for a unforgeable signature scheme. The new crs is defined by $(\mathrm{crs}, \mathrm{simtd}_{\mathrm{vk}})$ and

the new simulation-trapdoor by $(\mathrm{simtd}, \mathrm{simtd}_{\mathrm{sk}})$. Then we define $\mathbf{S}'(x, (\mathrm{simtd}, \mathrm{simtd}_{\mathrm{sk}}))$ as follows:

- Compute $p \leftarrow \mathbf{S}(x, \mathrm{simtd})$.
- If $\mathbf{V}(x, p, \mathrm{crs}) = 1$ then output p.
- Else output $\mathrm{sig}(\mathrm{simtd}_{\mathrm{sk}}, p)$.

Define the verification $\mathbf{V}'(x, p, (\mathrm{crs}, \mathrm{simtd}_{\mathrm{vk}}))$ by:

- If $\mathbf{V}(x, p, \mathrm{crs}) = 1$ then output 1.
- Else if $\mathrm{verify}_{\mathrm{sig}}(\mathrm{simtd}_{\mathrm{vk}}, p) \neq \perp$ output 1.
- Else output 0.

The proof system is still zero-knowledge, because if x is a true statement, then the verification of the original simulated proof succeeds (by the zero-knowledge property). The other properties carry over, as well, and it holds $\mathbf{V}'(x, \mathbf{S}'(x)) = 1$.

We construct a proof system $(\mathbf{P}', \mathbf{V}')$ as follows.

- The generation of the CRS and the trapdoors is performed as in $(\mathbf{P}, \mathbf{V})$.
- The prover is the same, too, i.e. $\mathbf{P}' := \mathbf{P}$.
- Define the verifier $\mathbf{V}'(\mathrm{crs}, x, z)$ as follows: It accepts all proofs that $\mathbf{V}$ accepts, but additionally accepts if $z = (\text{"}special\text{"}, x')$ and $x =$"There is a $w = pi$ such that $\mathbf{V}'(x', pi) = 1$".
- The simulator is the same as for $(\mathbf{P}, \mathbf{V})$.
- The extractor $\mathbf{E}'$ is defined as follows: it does the same as for $(\mathbf{P}, \mathbf{V})$, except that in the special case $\mathbf{E}(z, x)$ with $z =$ ("special", x'). In this case, it simply outputs $\mathbf{S}(x')$.

This proof system is still zero-knowledge, since we did not change $\mathbf{P}$ and $\mathbf{S}$ (and only those occur in the definition of ZK). Also, the proof system does not lose simulation-extractability (SE): Assume $\mathbf{P}', \mathbf{V}'$ is not SE. Then there exists an adversary that, given access to a simulation oracle $\mathbf{S}'$, produces a proof x, pi (that was never output by $\mathbf{S}'$) such that $\mathbf{V}'(x, pi) = 1$ and such that $\mathbf{E}'(pi)$ does not output a valid witness for x.

Case 1: The proof pi is not of the form $(\text{"}special\text{"}, x')$. Then $\mathbf{V}'$ and $\mathbf{E}'$ behave as $\mathbf{V}$ and $\mathbf{E}$, and $\mathbf{S} = \mathbf{S}'$ anyway. So we have an attack against SE of $\mathbf{P}$,$\mathbf{V}$.

Case 2: The proof pi is of the form $(\text{"}special\text{"}, x')$. Then $\mathbf{E}'$ outputs a valid witness for x by construction (since $\mathbf{V}(x', S(x')) = 1$).

Given this proof system $\mathbf{P}'$,$\mathbf{V}'$, we can now show its own unsoundness. Namely, let $x :=$"There is a $w = pi$ such that $\mathbf{V}'(\mathrm{false}, pi) = 1$" where false is an arbitrary wrong statement. Then $z :=$ ("special", false) is a valid proof for that statement (i.e.,$\mathbf{V}'(x, z) = 1$).

But the statement false is unsatisfiable, so since z is efficiently computable from false this implies that the NIZK is not sound. We have shown that if there is a proof system satisfying the stated properties, then there is an simulation-extractable NIZK which is unsound. But this is a contradiction because simulation-extractability implies soundness. Thus the initial proof system can not exist. And therefore the symbolic model could not be computationally sound. □

2.8 Conclusions

In this work, we have shown that computational soundness of symbolic ZK proofs can be achieved under realistic cryptographic assumptions for which efficient realizations and generic constructions are known. The computational soundness proof has been conducted in CoSP, and hence it holds independent of the underlying symbolic calculi and comes with mechanized proof support.

We conclude by highlighting two open questions that we consider as future work. First, current abstractions model non-interactive ZK proofs, i.e., a ZK proof constitutes a message that can forwarded, put into other terms, etc. Developing a symbolic abstraction to reflect (the more common) interactive ZK proofs thus requires a conceptually different approach, as such proofs cannot be replayed, put into other terms, etc. We plan to draw ideas from a recently proposed symbolic abstraction for (interactive) secure multi-party computation [18] to reflect this behavior. Second, soundness proofs of individual primitives have typically been proved in isolation, without a guarantee that the soundness result prevails when composed. We plan to build on recent work on composable soundness notions [56] to establish a composable soundness result for ZK proofs.

Chapter 3

Privacy Protocol Implementation: Designing Protocols for Multi-Party Computations (MPC)

Multiparty computation (MPC) among n parties can tolerate up to $t < n/2$ active corruptions in a *synchronous* communication setting; however, in an *asynchronous* communication setting, the resiliency bound decreases to only $t < n/3$ active corruptions. We improve the resiliency bound for asynchronous MPC (AMPC) to match synchronous MPC using *non-equivocation*.

Non-equivocation is a message authentication mechanism to restrict a corrupted sender from making conflicting statements to different (honest) parties. It can be implemented using an increment-only counter and a digital signature oracle, realizable with trusted hardware modules readily available in commodity computers and smartphone devices. A non-equivocation mechanism can also be transferable and allow a receiver to verifiably transfer the authenticated statement to other parties. In this work, using transferable non-equivocation, we present an AMPC protocol tolerating $t < n/2$ faults. From a practical point of view, our AMPC protocol requires fewer setup assumptions than the previous AMPC protocol with $t < n/2$ by Beerliová-Trubíniová, Hirt and Nielsen [PODC 2010]: unlike their AMPC protocol, it does not require *any* synchronous broadcast round at the beginning of the protocol and avoids the threshold homomorphic encryption setup assumption. Moreover, our AMPC protocol is also efficient and provides a gain of $\Theta(n)$ in the communication complexity per multiplication gate, over the AMPC protocol of Beerliová-Trubíniová et al. In the process, using non-equivocation, we also define the first asynchronous verifiable secret sharing (AVSS) scheme with $t < n/2$, which is of independent interest to threshold cryptography. The results described in this chapter have been published in [**BaBeChKa:14**].

3.1 Multi-Party Computations & Related Work

Multi-party computation (MPC) is an important primitives in distributed systems. Informally, in a system of n mutually distrusting parties, an MPC protocol allows the parties to "securely" evaluate any agreed-on function f of their private inputs, in the presence of a centralized active adversary $\mathcal{A}$, controlling at most any t out of the n parties. In the synchronous communication model, where the message transfer delays are bounded by a known constant, the MPC problem has been studied extensively (e.g.,[120, 72, 35, 49, 106, 58, 29, 30, 37]). In practice, there is growing interest in generalizing MPC to an asynchronous communication model [34, 46, 42] that does not place any bound on the communication delays. The weaker restrictions on the adversary in the asynchronous model not only worsen the required resiliency conditions and communication complexities, but also make designing protocols a more challenging task; intuitively this is because in a completely asynchronous setting, it is not possible to distinguish between a slow (but honest) sender and a crashed sender. Due to this, at any "stage" of an asynchronous protocol, no party can afford to wait to hear from all the parties and so the communication from t (potentially honest) parties may be ignored [46]. Due to their complexity, only a few asynchronous MPC (AMPC) protocols are available [34, 36, 114, 81, 80, 31, 104, 51].

In this work, we focus on an asynchronous model with a *computationally bounded* adversary $\mathcal{A}$, where the parties are connected by pairwise authenticated links. In this setting, AMPC protocols are possible if and only if $t < n/3$ [81, 80]. This is in contrast to the synchronous world, where we can tolerate up to $t < n/2$ corruptions [79]. Interested in bridging this gap between the resilience of synchronous and asynchronous MPC protocols, Beerliová-Trubíniová, Hirt and Nielsen [32] observed that it is possible to design an AMPC protocol tolerating $t < n/2$ corruptions in a "partial" synchronous network. More specifically, assuming one *synchronous broadcast* round at the beginning of the protocol, where each party can synchronously broadcast to every other party, they designed an AMPC protocol tolerating $t < n/2$ corruptions. Due to the availability of the synchronous broadcast round, their protocol could also ensure "input provision", i.e., the inputs of all the (honest) parties are considered for the computation, which otherwise is impossible to achieve in an asynchronous protocol [46]. Nevertheless, their requirement of one synchronous broadcast round per MPC instance may not always be realizable: deterministic broadcast protocols [63] require $\Theta(t)$ rounds of communication over the pairwise channels or randomized broadcast protocols [116, 68] require $\mathcal{O}(1)$ (with a large constant) expected rounds of communication. It was left as an open problem in [32] to see whether one can design an AMPC protocol with $t < n/2$ under other simplified assumptions.

In distributed computing research, a similar problem with asynchronous protocols has recently been addressed by introducing a small trusted hardware assumption [52, 92, 55, 54, 82, 84]. In particular, it was shown that, the resilience of asynchronous distributed computing tasks such as reliable broadcast, Byzantine agreement, and state machine replication (SMR) can be improved using a small *trusted hardware module* at each party. The hardware module utilized is just a trusted, increment-only local counter and a signature oracle, which can be realized

with pervasively available trusted hardware-enabled devices. Using such trusted hardware with each party, one can design asynchronous reliable broadcast tolerating up to $t < n$ active faults [54, 55], and asynchronous Byzantine agreement (ABA) and SMR protocols tolerating up to $t < n/2$ [52, 92, 84] active faults, all of which otherwise require $t < n/3$ [116].

At a conceptual level, such a trusted module makes it impossible for a corrupted party to perform *equivocation*, which essentially means making conflicting statements to different (honest) parties. The use of signatures (or transferable authentication) complements *non-equivocation* (i.e., making equivocation impossible) by making it transferable as required in the asynchronous environment with unknown delays. Clement et al. [54] generalized the results [52, 92, 55] and proved that non-equivocation with signatures (i.e., *transferable* non-equivocation) allows treating active (or Byzantine) faults as crash failure for many distributed computing primitives. In particular, they present a generic transformation that enables any crash-fault tolerant distributed protocol to tolerate the same number of Byzantine faults using transferable non-equivocation. Nevertheless, their generic transformation considers only the basic distributed computing requirements of safety and liveness. It does not apply to cryptographic tasks such as AMPC where *confidentiality (or privacy)* of inputs is also required. This presents an interesting challenge to assess the utility of transferable non-equivocation for the secure distributed computing task of AMPC.

3.1.1 Contribution and Comparison

We study the power of transferable non-equivocation in the context of AMPC and demonstrate how to improve the resilience of AMPC from $t < n/3$ to $t < n/2$, without any synchrony assumption. In particular, we present a general MPC protocol in a completely asynchronous communication model with $n \geq 2t + 1$. Our protocol, called NeqAMPC , improves upon the previous AMPC protocol [32] with $n \geq 2t + 1$ in the following ways:

(a) Simplified assumptions. The NeqAMPC protocol needs a transferable non-equivocation mechanism, but unlike [32] neither makes a synchronous broadcast round assumption nor requires a threshold homomorphic encryption setup. Given the feasibility of realizing transferable non-equivocation over prevalent computing devices, we argue that transferable non-equivocation is a more practical assumption than the synchronous broadcast round assumption.

(b) Efficiency. For a security parameter κ, our AMPC protocol requires an amortized communication complexity of $\mathcal{O}(n^3\kappa)$ bits per multiplication gate, which improves upon the AMPC protocol of [32] by a factor of $\Theta(n)$.

To reduce the setup assumptions for the NeqAMPC protocol, we avoid the traditional threshold additive homomorphic encryption based circuit evaluation approach as used in [81, 80, 32]. Instead, we employ a secret-sharing based circuit evaluation approach [35, 49, 106], where privacy of the computation is maintained via secret sharing. Nevertheless, as detailed in our protocol overview (Section 3.2), secret-sharing based AMPC with $n = 2t + 1$ and $\mathcal{O}(n^3\kappa)$ communication complexity (per multiplication) presents several interesting challenges. As a

result the NeqAMPC protocol is significantly different than those in the literature [81, 80, 31, 32].

In the process, we also present the first computationally secure asynchronous verifiable secret sharing (AVSS) [46, 42, 12, 11] scheme for $n \geq 2t + 1$ with $\mathcal{O}(n^2\kappa)$ communication complexity (using transferable non-equivocation), which otherwise requires $t < n/3$ [46]. Our AVSS scheme has an additional useful feature—it is the first publicly verifiable [115] AVSS scheme, as it allows any third party to publicly verify the "consistency" of the shares. With its efficiency and public verifiability, our AVSS scheme may be of independent interest to other cryptographic protocols.

Comparison with Existing Work. The best known computationally secure AMPC protocols are reported in [80, 32]. The protocol in [80] considers a *fully* asynchronous setting with $t < n/3$, whereas [32] assumes one synchronous broadcast round and can tolerate up to $t < n/2$ corruptions. Both the protocols require a threshold additive homomorphic encryption instantiation, and incur an (amortized) communication complexity of $\mathcal{O}(n^2\kappa)$ and $\mathcal{O}(n^4\kappa)$ bits per multiplication gate respectively.[1]

We do not employ a threshold encryption setup, but rather prefer a more standard public key encryption setup with the addition of transferable non-equivocation. Our NeqAMPC protocol with $t < n/2$ performs circuit evaluation by secret-sharing the inputs and incurs a communication complexity of $\mathcal{O}(n^3\kappa)$ bits per multiplication gate. Nevertheless, by modifying our protocol and employing a threshold encryption setup (coupled with transferable non-equivocation), we can tolerate $t < n/2$ faults with communication complexity $\mathcal{O}(n^2\kappa)$ bits per multiplication gate. However, we prefer the secret-sharing based AMPC, as we aim to reduce the assumptions relied upon.

We note that unlike [32], our AMPC protocol could not enforce *input provision*: the input from t potentially honest parties may be ignored for computation. As discussed earlier, this is inherent to asynchronous systems and presents a trade-off between our protocol and that of [32] based on what is more important: input provision or avoiding the synchrony assumption. Finally, we note that using a transferable non-equivocation mechanism, one can realize asynchronous reliable broadcast (see Section 3.1.3) with $t < n$ and consequently get rid of the synchronous broadcast round required in [32]. Nevertheless, the resultant protocol will still require the threshold homomorphic encryption setup and $\mathcal{O}(n^4\kappa)$ communication complexity per multiplication, and it will no longer support input provision.

3.1.2 Preliminaries

In this subsection, we discuss our adversary and communication model, define the AMPC protocol and the non-equivocation mechanism, and describe the required primitives.

[1] Beerliová-Trubíniová et al. [32] focused on designing a protocol with $t < n/2$, and the communication complexity of $\mathcal{O}(n^4\kappa)$ of their protocol (measured by us in section 3.9) can possibly be improved.

Model

We consider a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of n parties connected by pairwise authenticated channels, where $n = 2t + 1$. These communication channels are asynchronous with arbitrary but finite delay (i.e. the messages reach their destinations eventually). A centralized static adversary $\mathcal{A}$ can actively corrupt any t out of the n parties and force them to deviate in any arbitrary manner. A party not under the control of $\mathcal{A}$ is called *honest*. The adversary $\mathcal{A}$ is modeled as a probabilistic polynomial time (PPT) algorithm, with respect to a security parameter κ. During a protocol execution, the message delivery order is decided by a *scheduler* controlled by $\mathcal{A}$. Nevertheless, the scheduler cannot modify the messages exchanged between honest parties. A protocol execution is considered as a sequence of *atomic* steps, where a single party is active in each such step. A party is activated upon receiving a message, after which it performs some computation and possibly outputs messages on its outgoing links. The scheduler controls the order of these atomic steps. At the beginning of the execution, each party will be in a special *start* state. A party is said to *terminate/complete* the execution if it reaches a *halt* state. A protocol execution is said to *be complete* when all honest parties complete it. We assume that every message sent by a party during an execution has a publicly known unique *identifier (key)* associated with it. By $[y, z]$ we denote the set $\{y, y + 1, \ldots, z\} \subset \mathbb{N}$.

Definitions

Computationally Secure AVSS. Informally an AVSS scheme consists of two phases, a sharing phase, where a special party called *dealer* shares a secret and a reconstruction phase, where the parties reveal their shares to reconstruct the secret.

More formally, let $(\mathsf{Sh}, \mathsf{Rec})$ be a pair of protocols for parties in $\mathcal{P}$, where a dealer $\mathsf{D} \in \mathcal{P}$ has a private input $s \in \mathbb{Z}_p$ for Sh. Then $(\mathsf{Sh}, \mathsf{Rec})$ is an computationally secure AVSS scheme, if the following requirements hold for every possible adversary $\mathcal{A}$, except with a negligible probability in κ:

- TERMINATION:

(a) If D is *honest* and all the honest parties participate in the protocol Sh, then each *honest* party eventually terminates the protocol Sh;

(b) If some *honest* party terminates Sh, then every *honest* party eventually terminates Sh;

(c) If all the honest parties invoked Rec, then each *honest* party eventually terminates Rec.

- CORRECTNESS: If some honest party terminates Sh, then there exists a fixed value $\overline{s} \in \mathbb{Z}_p$ such that the following requirements hold except with a negligible probability in κ:

(a) If D is *honest*, then $\overline{s} = s$, and all the honest parties output $\overline{s}$ upon terminating Rec;

(b) Even if D is *corrupted*, all the honest parties output $\overline{s}$ upon terminating Rec.

- PRIVACY: If D is *honest* during the protocol Sh and no honest party has started to execute the protocol Rec, then the adversary $\mathcal{A}$ has no information about the secret s.

Computationally Secure AMPC. We briefly review computationally secure AMPC here, and refer the readers to [81, 80] for a formal definition. Informally, in an AMPC protocol Π_{MPC}, every party first provides its input in $\mathbb{Z}_p$ to the computation (in a secure fashion). Due to the asynchronous nature of communication, the parties cannot wait to consider the inputs of all n parties, and instead they agree on inputs from a set $\mathcal{CORE}$ of $n - t$ parties. The parties then compute an "approximation" of f on the inputs from the $\mathcal{CORE}$ set and assuming a default value (say 0) as the remaining t inputs. For every possible $\mathcal{A}$ and for all possible inputs and random coins of the (honest) parties, we expect the following properties for a Π_{MPC} instance, except with a negligible probability (in κ):

- *Termination:* all the honest parties eventually terminate Π_{MPC};
- *Correctness:* the honest parties obtain the correct output of the function f;
- *Privacy:* the adversary $\mathcal{A}$ obtains no additional information about the inputs of the honest parties other than what may be inferred from the inputs and outputs of the corrupted parties.

The above properties are formalized to the standard simulation-based definition following the real-world/ideal-world paradigm [46, 34, 81, 80].

(Transferable) Non-equivocation. Non-equivocation restricts a corrupted party from making conflicting statements to different parties, and it has been used in several asynchronous distributed systems [52, 92, 55, 54, 84] to improve their resiliency. In particular, these systems employ transferable non-equivocation, which (similar to digital signatures) allows a party to verifiably transfer a non-equivocation tag (or signature) provided by a sender to other parties. Clement et al. [54] justify the necessity of transferability of non-equivocation by proving that non-equivocation or signature alone are powerless in *asynchronous* distributed systems. Nevertheless, (transferable) non-equivocation has not been formalized so far, and we present a simplified, idealized definition for transferable non-equivocation.

In Figure 3.1, we define a simplified mechanism (Neq) which is intended to model the event for a transferable non-equivocation instantiation:

(1) during the setup phase, every party P_i gets associated with a unique non-equivocation list L_i characterized by its index i, and all parties are informed about this association.

(1) The list owner party can create a non-equivocation signature for any key-message pair except that she cannot equivocate and obtain a signature for the *same* key twice.

(1) Given a key-message-signature triplet associated with a sender, any party successfully verifies only correctly generated signatures except with a negligible probability.

The Neq mechanism

Neq is parameterized by a polynomial $p(\cdot)$, and an implicit security parameter κ.

SETUP: Upon receiving a (Setup) message from party $P_i \in \mathcal{P}$, do:

1. If a list L_i exists, return $\perp$.
2. Otherwise, create an empty list L_i of key-message-signature triplets of type $\{0,1\}^* \times \{0,1\}^* \times \{0,1\}^{p(\kappa)}$.
   Send a message (Registered, P_i) to all parties in $\mathcal{P}$.

SIGNING: Upon receiving an (Neq-Sign, P_i, ℓ, m) message from party P_i, do:

1. If the list L_i does not exist *or* $\langle \ell, \_, \_ \rangle \in L_i$, return $\perp$.
2. Otherwise, choose an arbitrary (signature) tag $\sigma_i^{\ell,m} \in \{0,1\}^{p(\kappa)}$, update $L_i \leftarrow L_i \cup \langle \ell, m, \sigma_i^{\ell,m} \rangle$ and return $\sigma_i^{\ell,m}$.

VERIFICATION: Upon receiving an (Neq-Verify, P_i, ℓ, m, σ) message from party $P_j \in \mathcal{P}$, do:

1. If the list L_i does not exist *or* $\langle \ell, m, \sigma \rangle \notin L_i$, then return 0, else return 1.

Figure 3.1: A simplified transferable non-equivocation mechanism Neq

We survey existing transferable non-equivocation instantiations and analyze their relations to Neq in section 3.7. In most instantiations, the transferable non-equivocation is implemented using an increment-only counter for keys and signatures with public key infrastructure (PKI) [52, 92, 55, 54] or message authentication codes (MACs) generated with a replicated secret key [84]. In Neq, we generalize these using the list L_i of key-message-signature triplets associated with party P_i indexed by *party-defined ordered keys* ℓ. Similar to signatures, only P_i can use Neq-Sign to add triplets (one per each key) to L_i. Similar to PKI, anybody can Neq-Verify if a triplet $\langle \ell, m, \sigma \rangle$ belongs to L_i of P_i, and verifiably transfer authentication to others. Note that the increment-only counter provides a space-efficient way to implement a list L_i as only the counter value has to be maintained and not the whole list.

For ease of exposition, we use a phrase P_i *sends* m_{σ_i} *to* P_j and P_j *receives* m_{σ_i} *from* P_i to suggest that P_i sends a triplet $\langle \ell, m, \sigma_i^{\ell,m} \rangle$ for a key ℓ to P_j and P_j delivers it only after applying Neq-Verify to check if $\sigma_i^{\ell,m}$ is obtained by P_i using Neq-Sign on ℓ and m. Similarly, we use a phrase P_j *forwards* m_{σ_i} *to* P_k to suggest that P_j received (in the above sense) message m_{σ_i} (of P_i) from some party, and then forwards it to P_k who should also (non-equivocally) receive it. Note that we avoid the keys ℓ in the above phrases as they can be pre-assigned to protocol instance-step combinations in an unambiguous manner.

3.1.3 Employed Primitives

We now discuss the existing primitives used in our protocols.

Homomorphic Encryptions and Commitments. We assume an IND-CPA secure *linear homomorphic encryption* scheme $(\mathsf{Enc}, \mathsf{Dec})$. Every party P_i has its own key-pair $(\mathsf{pk}_i, \mathsf{sk}_i)$, for which the public key pk_i is known to all parties. Given two ciphertexts $\mathbf{c}_{m_1} = \mathsf{Enc}_{\mathsf{pk}_i}(m_1, \cdot)$ and $\mathbf{c}_{m_2} = \mathsf{Enc}_{\mathsf{pk}_i}(m_2, \cdot)$, we require that there exist operations $\boxplus$ and $\boxdot$ on ciphertexts such that $\mathbf{c}_{m_1} \boxplus \mathbf{c}_{m_2} = \mathsf{Enc}_{\mathsf{pk}_i}(m_1 + m_2, \cdot)$ and $a \boxdot \mathbf{c}_{m_i} = \mathsf{Enc}_{\mathsf{pk}_i}(a \cdot m_i, \cdot)$ holds. We also assume an unconditional hiding and computational binding *linear homomorphic commitment* scheme

$(\mathsf{Commit}, \mathsf{Open})$ with the analogous homomorphic operations; these are denoted by $\oplus$ and $\odot$. For the sake of readability, we sometimes leave the randomness of encryptions and commitments implicit.

For instantiating encryptions and commitments over messages in $\mathbb{Z}_p$, we use the encoding-free additive El-Gamal encryption scheme [50] and Pedersen commitment scheme [105] respectively. In particular, we only require that the scheme is CPA secure (Theorem 3 in [50]) and it is not necessary that applying homomorphism twice to the same encrypted values leads to different ciphertexts.

Zero-knowledge (ZK) Proofs. We assume the presence of the following two-party ZK protocols.

1) Zero knowledge proof of equality of encrypted and committed values (PoE)**.** There exists a prover $\mathsf{P} \in \mathcal{P}$ who computes and publishes a commitment $\mathsf{Com}_m = \mathsf{Commit}(m, r)$, and ciphertexts $\mathbf{c}_m = \mathsf{Enc}_{\mathsf{pk}_i}(m, \cdot)$ and $\mathbf{c}_r = \mathsf{Enc}_{\mathsf{pk}_i}(r, \cdot)$. Then using PoE, the prover P can prove to *any* verifier $\mathsf{V} \in \mathcal{P}$ (knowing Com_m, $\mathbf{c}_m$, $\mathbf{c}_r$ and pk_i) that the message encrypted in $\mathbf{c}_m$ is also committed in Com_m, under the randomness encrypted in $\mathbf{c}_r$; i.e.,

$$\exists m, r, r_1, r_2 : \mathsf{Com}_m = \mathsf{Commit}(m, r) \quad \wedge \quad \mathbf{c}_m = \mathsf{Enc}_{\mathsf{pk}_i}(m, r_1) \quad \wedge \quad \mathbf{c}_r = \mathsf{Enc}_{\mathsf{pk}_i}(r, r_2).$$

2) Zero knowledge proof of correct pre-multiplication (PoCM)**.** Given publicly known commitments $\mathsf{Com}_{v_j} = \mathsf{Commit}(v_j, r_j)$ and corresponding ciphertexts $\mathbf{c}_{v_j} = \mathsf{Enc}_{\mathsf{pk}_j}(v_j, \cdot), \mathbf{c}_{r_j} = \mathsf{Enc}_{\mathsf{pk}_j}(r_j, \cdot)$ for $j \in [1, n]$, there exists a prover $\mathsf{P} \in \mathcal{P}$ who selects a random $u \in Z_p$ and t-degree random polynomials $m(\cdot)$ and $\hat{m}(\cdot)$ in $Z_p[x]$ with $m(0) = \hat{m}(0) = 0$. Let $m_j = m(j)$ and $\hat{m}_j = \hat{m}(j)$ for $j \in [0, n]$. In addition, P publishes $\mathsf{Com}_u = \mathsf{Commit}(u, \cdot)$ and $\mathsf{Com}_{m_j} = \mathsf{Commit}(m_j, \hat{m}_j)$. Using the homomorphic property of commitments and encryptions, P computes and publishes the commitment $\mathsf{Com}_{u \cdot v_j + m_j} = \mathsf{Commit}(u \cdot v_j + m_j, u \cdot r_j + \hat{m}_j)$ and encryptions $\mathbf{c}_{u \cdot v_j + m_j}$ and $\mathbf{c}_{u \cdot r_j + \hat{m}_j}$ of $u \cdot v_j + m_j$ and $u \cdot r_j + \hat{m}_j$ respectively. Then using PoCM, P can prove to *any* verifier $\mathsf{V} \in \mathcal{P}$ that all values $\mathsf{Com}_{u \cdot v_j + m_j}$ were generated by multiplying Com_{v_j} with the same u followed by re-randomization using the same $m(\cdot)$ and $\hat{m}(\cdot)$ polynomials, and that all $\mathbf{c}_{u \cdot v_j + m_j}$ and $\mathbf{c}_{u \cdot r_j + \hat{m}_j}$ values were generated by multiplying $\mathbf{c}_{v_j}$ and $\mathbf{c}_{r_j}$ respectively with the same u, followed by re-randomization using the same $m(\cdot)$ and $\hat{m}(\cdot)$ respectively. i.e.,

$$\begin{aligned}\exists u, \rho, m(\cdot), \hat{m}(\cdot), \{k_j, \hat{k}_j\}_{j \in [1,n]} : \mathsf{Com}_u = \mathsf{Commit}(u, \rho) \ \wedge \ \deg(m(\cdot)) \leq t \ \wedge \ \deg(\hat{m}(\cdot)) \leq t \ \wedge \\ m(0) = 0 = \hat{m}(0) \ \wedge \ \mathsf{Com}_{u \cdot v_j + m_j} = u \odot \mathsf{Com}_{v_j} \oplus \mathsf{Commit}(m_j, \hat{m}_j) \ \wedge \\ \mathbf{c}_{u \cdot v_j + m_j} = u \boxdot \mathbf{c}_{v_j} \boxplus \mathsf{Enc}_{\mathsf{pk}_j}(m_j, k_j) \ \wedge \mathbf{c}_{u \cdot r_j + \hat{m}_j} = u \boxdot \mathbf{c}_{r_j} \boxplus \mathsf{Enc}_{\mathsf{pk}_j}(\hat{m}_j, \hat{k}_j).\end{aligned}$$

Both the ZK protocols are based on standard Σ-protocols [33] and have communication complexity $\mathcal{O}(\kappa)$ bits and $\mathcal{O}(n\kappa)$ bits respectively. See Appendix 3.8 for their instantiations based on the ZK protocols in [45].

Certificates of Claims. Hirt, Nielsen, and Przydatek [80] introduced this concept to allow a prover $\mathsf{P} \in \mathcal{P}$ to *publicly* prove correctness of a certain claim (like real-life certificates), without

revealing any additional information. Here, to certify validity of a statement m, the prover P proves m to every *verifier* $P_i \in \mathcal{P}$ using an appropriate zero-knowledge (ZK) protocol. A verifier P_i, upon successful verification, sends a signature to P on an "appropriate" message (known publicly), corresponding to m. P cannot wait for all n signatures in the asynchronous environment; thus, upon receiving $(n-t) = t+1$ signatures, the prover P concatenates them to construct a certificate α for the claim m. These $t+1$ signatures ensure that at least one honest party has verified the claim, and that m is true with an overwhelming probability. Assuming each signature to be of size $\mathcal{O}(\kappa)$ bits, the size of α will be $\mathcal{O}(n\kappa)$ bits; this can be reduced to $\mathcal{O}(\kappa)$ bits using a *threshold signature scheme* with threshold t [32]. Here, the verifiers send signature *shares* and the prover P, instead of concatenating, combines $(n-t)$ shares to a single signature.[2]

Let zkp be the ZK protocol corresponding to the claim m. We denote the task of constructing a certificate α for m as $\alpha = \mathsf{certify}_{\mathsf{zkp}}(m)$. Similarly we say that "P_i verifies the certificate α for the claim m" to mean that that P_i verifies that α is a valid (threshold) signature on the appropriate message corresponding to m. The communication cost of constructing α is the same as that of executing n instances of the corresponding ZK protocol zkp.

Reliable Broadcast ($\mathsf{r\text{-}broadcast}$). This asynchronous primitive [39, 116, 78] allows a sender S to send a message m identically to *all* the parties: For a given instance τ_b of $\mathsf{r\text{-}broadcast}$, when S is *honest*, all honest parties eventually terminate with output (τ_b, m); if S is *corrupted* and some honest party terminates with (τ_b, m'), then every honest party also eventually terminates with (τ_b, m'); for any instance, at most one message can be delivered by an honest party.

The resiliency bound for $\mathsf{r\text{-}broadcast}$ is $n \geq 3t+1$ [39, 116, 78]; however, assuming transferable non-equivocation, an $\mathsf{r\text{-}broadcast}$ protocol with $n \geq t+1$ and $\mathcal{O}(n^2(\ell+\kappa))$ bits of communication for broadcasting an ℓ-bit message is available [54, 55]. The high-level idea of the protocol is as follows: S first (non-equivocally) sends $m_{\sigma_{\mathsf{S}}}$ to all the parties; this prevents a corrupted S from sending different messages to different honest parties. However, a corrupted S can avoid sending $m_{\sigma_{\mathsf{S}}}$ to some honest parties. Thus, to ensure that all the honest parties eventually receive $m_{\sigma_{\mathsf{S}}}$, whenever an honest party (non-equivocally) receives some message $m_{\sigma_{\mathsf{S}}}$, before delivering the message, it non-equivocally forwards it to every other party. This ensures that whenever an honest party receives $m_{\sigma_{\mathsf{S}}}$ then it will be eventually received by every other honest party.

In the rest of the paper, the term "P_i *broadcasts* m" means that P_i as a sender invokes an $\mathsf{r\text{-}broadcast}$ instance for m. Similarly, "P_j *receives* m *from the broadcast of* P_i" means that P_j terminates the $\mathsf{r\text{-}broadcast}$ instance τ_b invoked by P_i with the output (τ_b, m).

Agreement on a Common Subset (ACS). This primitive allows the parties to agree on a common subset of $(n-t)$ parties, who correctly invoked some protocol, say Π, satisfying the following requirements: (a) If an honest party invokes an instance of Π then all the (honest) parties eventually terminate the instance; (b) If some honest party terminates an instance of Π

[2] Note that the AMPC protocols of [80, 32] also assume a threshold signature setup.

invoked by a corrupted party, then every honest party eventually does the same. ACS can be realized by executing n instances (one for each party) of an asynchronous Byzantine agreement (ABA) protocol to decide if it should be included in the common subset. Assuming transferable non-equivocation, ABA, and hence ACS, can be implemented with $n \geq 2t + 1$ [54, 55, 84]. An efficient ACS protocol with expected communication complexity of $\mathcal{O}(n^3\kappa)$ bits can be obtained by using the Neq mechanism in the multi-valued ABA of [43].

3.1.4 Secret Sharing Notations

Given a secret $s \in \mathbb{Z}_p$, let $\phi(\cdot), \psi(\cdot) \in \mathbb{Z}_p[x]$ be, respectively, a degree t *sharing polynomial* with $\phi(0) = s$ and a t-degree *randomness polynomial* required for commitments; here, p is a κ-bit prime. For party P_j, $s_j = \phi(j)$ and $r_j = \psi(j)$ are respectively her shares of s and the randomness polynomial. Let $\mathbf{c}_{s_j} = \mathsf{Enc}_{\mathsf{pk}_j}(s_j, \cdot)$, $\mathbf{c}_{r_j} = \mathsf{Enc}_{\mathsf{pk}_j}(r_j, \cdot)$ and $\mathsf{Com}_{s_j} = \mathsf{Commit}(s_j, r_j)$ and let $\mathsf{Com}_s = \mathsf{Commit}(s, \psi(0))$. We call $\{\mathbf{c}_{s_j}, \mathbf{c}_{r_j}\}_{j\in[1,n]}$ the encrypted shares and $\{\mathsf{Com}_{s_j}\}_{j\in[1,n]}$ the committed shares of s.

$[\cdot]$-sharing: A secret s is said to be $[\cdot]$-shared, if every (honest) party $P_i \in \mathcal{P}$ holds s_i, r_i, $\{\mathsf{Com}_{s_j}\}_{j\in[1,n]}$ and Com_s. The information held by the (honest) parties corresponding to $[\cdot]$-sharing of s is denoted as $[s]$.

Privileged party: P_i is called a privileged party of $[\cdot]$-sharing of s if it holds the encrypted shares $\{\mathbf{c}_{s_j}, \mathbf{c}_{r_j}\}_{j\in[1,n]}$.

Due to linearity of sharing and commitments, $[\cdot]$-sharing is also *linear*: given $[a]$, $[b]$ and a public constant c, every party can locally compute its information corresponding to $[a + b]$ and $[c \cdot a]$, as $[a + b] = [a] + [b]$ and $[c \cdot a] = c \cdot [a]$ respectively.

3.2 Overview of Our NeqAMPC Protocol

Without loss of generality, we assume $n = 2t + 1$; thus, $t = \Theta(n)$. We assume that the function f to be computed is expressed as an arithmetic circuit over the field $\mathbb{Z}_p$, where $p > n$ is a κ bit prime and κ is the security parameter. The circuit consists of two-input addition (linear) and multiplication (non-linear) gates, apart from random gates. The AMPC protocol consists of two phases: an input phase and a computation phase. During the input phase, the parties share their inputs, while during the computation phase, the parties jointly evaluate f on the shared inputs and publicly reconstruct the output. Linear gates can be evaluated locally if the underlying secret-sharing scheme is linear; thus, we use the polynomial-based (Shamir) secret-sharing scheme with threshold t [113]. We denote a sharing of a value s by $[s]$. It follows that locally adding the shares of $[x]$ and $[y]$ provides the shares for $[x + y]$.

Multiplication gates cannot be evaluated locally since multiplying the individual shares results in the underlying sharing polynomial having degree $2t$ instead of t. Therefore we evaluate multiplication gates using the standard Beaver's circuit-randomization technique [28]. This technique requires three "pre-processed" secret-shared values, say $([u], [v], [w])$, unknown to the adversary $\mathcal{A}$, such that $w = u \cdot v$. Given such a shared *multiplication triple*, and shared

inputs of a multiplication gate, say $[x]$ and $[y]$, the multiplication gate is securely evaluated using the equation $[x \cdot y] = (x-u) \cdot (y-v) + [v] \cdot (x-u) + [u] \cdot (y-v) + [u \cdot v]$. In particular, the parties compute the sharings of $(x-u)$ and $(y-v)$, and publicly reconstruct the same. Once $(x-u)$ and $(y-v)$ are public, the parties can compute their shares of $x \cdot y$, using the above equation and employing linearity of the secret sharing. As u and v are random and unknown to $\mathcal{A}$, the public knowledge of $(x-u)$ and $(y-v)$ does not violate the privacy of x and y.

3.2.1 Pre-processing Phase

Although our above AMPC protocol idea is the same as the existing information-theoretically secure MPC and AMPC protocols [60, 31, 30, 51], our major challenge lies in generating the required shared multiplication triples with $n = 2t+1$ parties; the existing protocols [60, 31, 30, 51] employ at least $n > 3t$ parties for this purpose[3]. These triplets are independent of the circuit and the inputs of the parties, and generated in an additional *pre-processing* phase. Generating these triplets efficiently is the important problem we solve in our protocol. In the rest of the section, we give an overview of how $(c_M + c_R)$ shared random triples are generated, where c_M and c_R are the number of multiplication gates and random gates in the circuit. As a first step, we describe how a single triple is generated (see Figure 3.2 for a pictorial representation of the protocols involved) and then extend this to $c_M + c_R$ triples.

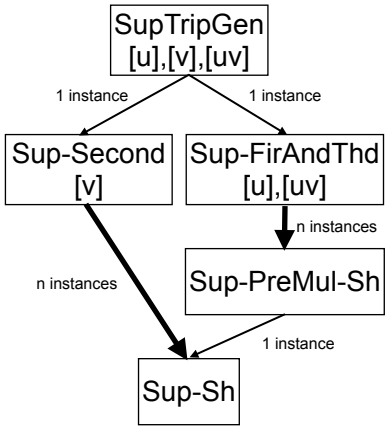

Figure 3.2: Multiplication Triple Generation under Supervision of a P_{king}

Supervised Triple Generation (Section 3.5). The idea for generating a random shared multiplication triple $[u], [v], [w]$ is to compute a random sharing $[v]$ and then combining "several" $[u^{(i)}]$s and $[u^{(i)} \cdot v]$s to get $[u]$ and $[w]$. The triple generation protocol uses two sub-protocols: Sup-Sh and Sup-PreMul-Sh. Protocol Sup-Sh allows a *dealer* D to "verifiably" generate the sharing $[u]$ of his value u, while Sup-PreMul-Sh allows a dealer D to "verifiably" generate a sharing $[u]$ and $[u \cdot v]$, given u and $[v]$. To generate $([u], [v], [w])$, we use Sup-Sh and

[3]Shared multiplication triples with $n = 2t+1$ have been generated in the synchronous setting [29, 37]; however, their adaptability to the asynchronous setting is unclear.

Sup-PreMul-Sh in the following way: first, we ask each party P_i to act as a dealer D and invoke an instance of Sup-Sh to share a uniformly random value, say $v^{(i)}$. The parties then agree on a common subset (say $\mathcal{T}_v$) of $t+1$ dealers whose Sup-Sh instances will be eventually terminated by all the parties. We set $v = \sum_{P_i \in \mathcal{T}_v} v^{(i)}$. The shared value v will be random and unknown to $\mathcal{A}$, as $\mathcal{T}_v$ has at least one honest party. Next, each party P_i is asked to act as a D and invoke a Sup-PreMul-Sh instance to share a uniformly random value $u^{(i)}$ as well as $u^{(i)} \cdot v$. The parties then agree on a common subset of $t+1$ dealers, say $\mathcal{T}_u$, whose Sup-PreMul-Sh instances will be eventually terminated by all the parties. For $u = \sum_{P_i \in \mathcal{T}_u} u^{(i)}$ and $w = \sum_{P_i \in \mathcal{T}_u} u^{(i)} \cdot v$, the triple (u, v, w) is a random multiplication triple.

There is, however, an important subtlety: As a *precondition*, the Sup-PreMul-Sh protocol expects its dealer D to also have *encryptions* of all n shares of $[v]$, encrypted under the individual keys of the respective share-holders; here, the encryption scheme is additively homomorphic (and not threshold additively homomorphic). For any sharing, we call a party having such n encrypted shares to be *privileged*. Due to asynchronicity, the Sup-Sh protocol cannot guarantee that *all* the $n-t$ honest parties are privileged with respect to every $[v^{(i)}]$ sharing of $P_i \in \mathcal{T}_v$. We solve the problem by ensuring in Sup-Sh that there exists a designated (possibly corrupted) *supervisor* P_{king} (called *king*), who is a privileged party with respect to *each* $[v^{(i)}]$. An honest P_{king} then computes all the n encrypted shares of $[v]$ using the homomorphic properties of encryption, and reliably broadcasts those encrypted shares. Using non-equivocation, the required asynchronous reliable broadcast is possible for $n > t$ (Section 3.1.3). Once P_{king} (correctly) broadcasts the n encrypted shares of v, then each P_i can invoke its Sup-PreMul-Sh instance.

The resultant wrapper protocols are called Sup-Second and Sup-FirAndThd, where Sup-Second generates $[v]$ under the supervision of P_{king} and Sup-FirAndThd generates $[u]$ and $[w = u \cdot v]$ under the supervision of P_{king}. A combination of Sup-Second and Sup-FirAndThd leads to the protocol SupTripGen under the supervision of a designated P_{king}, which outputs a uniformly random and private multiplication triple $([u], [v], [w])$.

Preprocessing Phase Protocol. Protocol SupTripGen may not terminate for a *corrupted* P_{king}. Therefore, we ask each party P_i to act as a king and generate shared random multiplication triples under its supervision by invoking an instance of SupTripGen. As the instances of honest kings will eventually terminate, we distribute the load of generating $c_M + c_R$ shared random multiplication triples among n parties. Each party P_i is asked to act as a king and generate $\frac{c_M + c_R}{t+1}$ shared multiplication triples in its SupTripGen instance. The parties then agree on a common subset $\mathcal{T}_{\mathsf{king}}$ of $t+1$ kings whose SupTripGen instances will be eventually completed by everyone and the $|\mathcal{T}_{\mathsf{king}}| \cdot \frac{c_M + c_R}{t+1} = c_M + c_R$ shared triples obtained in these instances are considered as the final output.

3.2.2 Important Sub-protocols for the Preprocessing Phase

We now discuss the realization of the main sub-protocols Sup-Sh and Sup-PreMul-Sh for the preprocessing phase.

Protocol Sup-Sh **(Section 3.4.1).** Our Sup-Sh protocol is almost equivalent to the AVSS primitive[42, 46, 12]: it allows a *dealer* D to "verifiably" share a secret s, thus generating $[s]$, and ensures that at least *one* honest party is privileged to obtain all the n shares encrypted for the respective share holders. The existing computational AVSS protocols (e.g., [42, 12]) are designed with $n = 3t + 1$ and are based on sharing a secret using a bivariate polynomial of degree t in each variable and (homomorphic) commitments. In this paradigm, it is ensured that D has distributed "consistent" shares to $n - t = 2t + 1$ parties such that (at least) $t + 1$ *honest* parties among them can "enable" the remaining parties to get their shares. Unfortunately, this approach cannot be used with $n = 2t + 1$, as here we can only ensure that D has distributed consistent shares to $n - t = t + 1$ parties. In the worst case, there will be *only one* honest party in this set, who does not have sufficient information to help the other honest parties to complete a sharing.

We solve this problem by introducing encryptions of the shares[4], and by employing univariate polynomials instead of bivariate polynomials. Here, D provides a vector of n encrypted shares as well as homomorphic commitments of those shares to each party. The non-equivocation mechanism is used to ensure that a *corrupted* D does not distribute different sets of encrypted and committed shares to the different parties. Once $n - t = t + 1$ parties confirm that they have received "consistent" n encrypted and committed shares, there must exist at least one honest privileged party with all n encrypted shares, who can transfer the individual encrypted shares to the individual parties. Transferability of non-equivocation ensures that corrupted privileged parties do not transfer incorrect encryptions.

Protocol Sup-PreMul-Sh **(Section 3.4.2).** The protocol takes as input an existing sharing $[v]$ of a value v unknown to everybody including $\mathcal{A}$, such that *all* the parties are privileged, i.e., all the parties hold encryptions of all shares. The protocol then allows a dealer D to verifiably share its value u as well as $u \cdot v$ (i.e. $[u]$ and $[u \cdot v]$). The protocol ensures that $u \cdot v$ remains secure in general and u is secure for an *honest* D. The idea behind the protocol is that knowing the encrypted and committed shares of v and employing the homomorphic properties of encryptions and commitments, D can compute the encrypted and committed shares corresponding to $u \cdot v$ for his choice of u, even without knowing v. The dealer can then (non-equivocally) distribute the encrypted and committed shares to the parties. Once it is confirmed that $t + 1$ parties have received all the n encrypted and committed shares of $u \cdot v$, it is ensured that there exists a honest privileged party, who can relay the individual encrypted shares of $u \cdot v$ to the respective parties.

We take a more bottom-up approach in the rest of the paper. We describe our model, and define non-equivocation and other primitives in Section 3.1.2. We present our AVSS protocol in Section 3.3. We start our AMPC construction with subprotocols Sup-Sh and Sup-PreMul-Sh in Section 3.4. We then present our supervised multiplication triple generation in Section 3.5 and finally describe the complete AMPC protocol in Section 3.6.

[4]We argue that the problem is inherently not solvable for $n = 2t + 1$ with only commitments usually employed in computationally secure VSS protocols [42, 12], and that we have to employ encryptions which allow a *single honest* party to procure encrypted shares of all the parties. Interestingly, the problem persists even when we assume the adversary $\mathcal{A}$ is only passive (but crashable), not Byzantine.

3.3 Employed AVSS Protocol

Protocols Sh and Rec presented in Figure 3.3 constitutes an AVSS scheme with $n = 2t + 1$. Protocol Sh allows a *dealer* D to "verifiably" generate $[s]$ for a secret $s \in \mathbb{Z}_p$. It ensures that if the protocol terminates then there exists a value (say $\overline{s}$) which will be $[\cdot]$-shared among the parties; if D is *honest* then $\overline{s} = s$, and $\mathcal{A}$ learns no new information on s. The protocol always terminates for an *honest* D and has communication complexity $\mathcal{O}(n^2\kappa)$. Protocol Rec allows the parties to reconstruct s, given that s is $[\cdot]$-shared.

In the Sh protocol, D polynomial-shares the secret s with threshold t to generate shares $\{s_j\}_{j\in[1,n]}$ and computes the commitments $\mathsf{Com}_{s_j} = \mathsf{Commit}(s_j, r_j)$, where r_j is a share of a random t-degree randomness polynomial $\psi(\cdot)$. It also computes the encryptions $\mathbf{c}_{s_j} = \mathsf{Enc}_{\mathsf{pk}_j}(s_j)$ and $\mathbf{c}_{r_j} = \mathsf{Enc}_{\mathsf{pk}_j}(r_j)$ of each share-pair s_j, r_j, and the commitment $\mathsf{Com}_s = \mathsf{Commit}(s, \psi(0))$. D then (non-equivocally) *sends* $\{\mathbf{c}_{s_j}\}_{\sigma_\mathsf{D}}$, $\{\mathbf{c}_{r_j}\}_{\sigma_\mathsf{D}}$ and $\{\mathsf{Com}_{s_j}\}_{\sigma_\mathsf{D}}$ for all $j \in [1, n]$, and $\{\mathsf{Com}_s\}_{\sigma_\mathsf{D}}$ to *every* party and claims that it has correctly $[\cdot]$-shared a secret: the claim involves proving that the plaintexts in $\mathbf{c}_{s_j}$ and $\mathbf{c}_{r_j}$ are committed in Com_{s_j} and that the values committed in $\{\mathsf{Com}_{s_j}\}_{j\in[1,n]}$ constitute shares of the secret committed in Com_s with threshold t. Note that although sending the full vector $\{\mathbf{c}_{s_j}\}_{\sigma_\mathsf{D}}$, $\{\mathbf{c}_{r_j}\}_{\sigma_\mathsf{D}}$ and $\{\mathsf{Com}_{s_j}\}_{\sigma_\mathsf{D}}$ to each party looks a bit non-intuitive, it is the *crux* of our Sh protocol to ensure that every party eventually receives its shares for $[s]$.

To verify D's claim, upon (non-equivocally) receiving information from D, every P_i verifies if the committed shares $\{\mathsf{Com}_{s_j}\}_{j\in[1,n]}$ constitute Shamir sharing of the secret committed in Com_s with threshold t. For this, the parties use the fact that given the commitments to at least $t + 1$ distinct points on a t degree polynomial, it is possible to (homomorphically) compute the commitments of the coefficients of the polynomial [71, 42]. In particular, party P_i takes Com_s along with $\{\mathsf{Com}_{s_j}\}_{j\in[1,t]}$ and homomorphically computes the commitments of the sharing and randomness polynomial. Using these commitments, party P_i then computes the commitments of the remaining $n - t$ points and matches them with $\{\mathsf{Com}_{s_j}\}_{j\in[t+1,n]}$. Additionally P_i engages in n instances of PoE (one per triplet $\{\mathbf{c}_{s_j}, \mathbf{c}_{r_j}, \mathsf{Com}_{s_j}\}$) with D. By non-equivocally sending messages to the parties, it is ensured that the parties who receive the messages from D, receive the *same* messages. D then constructs a certificate $\alpha^{\mathsf{D},\tau}$ (for session id τ) to support its claim of correct sharing and broadcasts it. A party proceeds further only upon receiving a valid certificate. A valid $\alpha^{\mathsf{D},\tau}$ ensures that at least one honest party, say P_h, has verified D's claim; P_h will be an *honest privileged party*.

A valid certificate from D does not ensure that every (honest) P_i will eventually hold its information corresponding to $[s]$: due to asynchrony or possible corrupted behavior of D, t honest parties may not receive their shares corresponding to $[s]$. We solve this problem by using *two* additional "rounds" of communication, which we call the D-*independent phase*. Each *privileged* party non-equivocally *forwards only* $\{\mathbf{c}_{s_j}\}_{\sigma_\mathsf{D}}$, $\{\mathbf{c}_{r_j}\}_{\sigma_\mathsf{D}}$, and $\{\mathsf{Com}_{s_j}\}_{\sigma_\mathsf{D}}$ to every party P_j, who can decrypt $\mathbf{c}_{s_j}$ and $\mathbf{c}_{r_j}$ to obtain s_j, r_j. Existence of at least one honest privileged party ensures that every P_j eventually receives s_j, r_j and Com_{s_j}. Next, every P_i forwards $\{\mathsf{Com}_{s_i}\}_{\sigma_\mathsf{D}}$ to all parties. As all $t + 1$ honest parties would eventually receive their respective

$\{\mathbf{c}_{s_i}\}_{\sigma_\mathsf{D}}$, $\{\mathbf{c}_{r_i}\}_{\sigma_\mathsf{D}}$, and $\{\mathsf{Com}_{s_i}\}_{\sigma_\mathsf{D}}$ messages at the end of first "round" of the D-independent phase, eventually every honest party will receive $t+1$ forwarded committed shares. Now using the homomorphic property of commitments every party can compute the remaining committed shares and Com_s, thus possessing all the necessary information of $[s]$.

Given $[s]$ generated using Sh, protocol Rec is based on the standard reconstruction protocol used in the existing computationally secure AVSS [12, 42], which allows the parties to *robustly* reconstruct s. In the protocol, each party sends its share-pair to all the parties, which are verified with the corresponding commitment, available with the parties (as part of $[s]$). Once $t+1$ "correct" share pairs are received, the sharing polynomial, and hence s, is reconstructed. As there exist at least $t+1$ honest parties whose shares will eventually be communicated among themselves, the Rec protocol eventually terminates. As stated in Lemma 3.3.1. the pair of protocols $(\mathsf{Sh}, \mathsf{Rec})$ constitutes an AVSS scheme with $n = 2t+1$ and communication complexity $\mathcal{O}(n^2\kappa)$ bits.

Lemma 3.3.1. *Protocols* $(\mathsf{Sh}, \mathsf{Rec})$ *constitute a computational secure AVSS scheme tolerating* $t < n/2$ *corruptions, where both* Sh *and* Rec *incur communication cost of* $\mathcal{O}(n^2\kappa)$ *bits.*

This lemma follows immediately from lemma 3.4.1 comparing the protocols Sup-Sh and Sh. Notice that the above AVSS scheme is also publicly verifiable [115] as any third party can verify the consistency of the shares using the valid certificate broadcasted by D.

Commitment to Shares instead of Polynomial Coefficients during Sh: In most existing computational secure VSS schemes [105, 12], D commits to the polynomial coefficients of sharing and randomness polynomials during the sharing phase, and the parties homomorphically generates the committed shares from those. This approach makes VSS simpler as the parties are not required to verify whether the committed shares lie on degree t polynomial.

If we follow this approach in our Sh protocol, then D has to non-equivocally distribute the commitment to polynomial coefficients of the sharing and randomness polynomials. In that case, however, the homomorphically generated committed shares will not have the necessary non-equivocation tag as the non-equivocation mechanism is not required to be homomorphic in nature. As a result, a corrupted privileged party can forward some incorrect committed shares to the respective parties during the D-independent phase, and the correctness of the protocol. Although the privileged parties can be asked to non-equivocally forward the polynomial coefficients during the D-independent phase, and it will result in an additional $\Theta(n)$ communication overhead.

3.4 Supervised Sharing Protocols

In this section, we present two protocols for generating $[\cdot]$-sharings with different properties under the supervision of a king P_{king}. Here, if the protocols terminate, then an *honest* P_{king} will be a *privileged* party with respect to the generated sharings.

3.4.1 Protocol Sup-Sh: Supervised $[\cdot]$-sharing

In our first supervised sharing protocol Sup-Sh, a *dealer* D verifiably generates $[s]$ for a secret s under the supervision of $P_{\mathsf{king}} \in \mathcal{P}$. If the protocol terminates then there exists a value (say $\overline{s}$) which will be $[\cdot]$-shared among the parties; if D is *honest* then $\overline{s} = s$, and $\mathcal{A}$ learns no new information on s from the protocol execution. Moreover, if P_{king} is *honest* then it will be a privileged party. The protocol always terminates for an *honest* D and P_{king} and has communication complexity $\mathcal{O}(n^2\kappa)$.

We obtain protocol Sup-Sh by adding two small verification steps during the D-dependent phase in the protocol Sh in Figure 3.3. Specifically, in the term of properties Sup-Sh is the same as Sh, with the additional requirement that P_{king} is a privileged party. To ensure the same, in protocol Sh, during the D-dependent phase, we ask P_{king} to broadcast an "acknowledgement" (a special message) after verifying the claim of D during the D-dependent phase. Additionally, we enforce that every party should this receive acknowledgement from P_{king}, before proceeding further. As an honest P_{king} will broadcast the acknowledgement only after verifying the claim of D, these additional steps ensure that indeed P_{king} will be a privileged party. We summarize the key properties of Sup-Sh in the following lemma and conclude the subsection with its proof.

Lemma 3.4.1. *Let s be the* D*'s secret. Then for every possible $\mathcal{A}$ and scheduler, protocol* Sup-Sh *achieves the following properties up to a negligible probability in κ:*

(1) TERMINATION*: if* D *and P_{king} are honest then all the honest parties eventually terminate the protocol. Moreover, if some honest party terminates the protocol, then every other honest party eventually does the same.*

(1) CORRECTNESS*: if some honest party terminates the protocol, then there exists a value $\overline{s}$ which will eventually be $[\cdot]$-shared among the parties. Moreover, if* D *is honest then $\overline{s} = s$. Furthermore if P_{king} is honest then P_{king} will be a privileged party.*

(1) PRIVACY*: if* D *is honest then s remains private.*

(1) COMMUNICATION COMPLEXITY*: the protocol has communication complexity of $\mathcal{O}(n^2\kappa)$ bits.*

Proof. For TERMINATION, we first consider an honest D and P_{king}. In this case, D will (non-equivocally) send $\{\mathbf{c}_{s_j}\}_{\sigma_D}, \{\mathbf{c}_{r_j}\}_{\sigma_D}, \{\mathsf{Com}_{s_j}\}_{\sigma_D}, \{\mathsf{Com}_s\}_{\sigma_D}$ to all parties. In particular all honest parties will eventually receive them and start participating in the instances of PoE, where all the verifications will pass. So P_{king} will eventually broadcast the $(\mathtt{OK}, \mathsf{D})$ message, which from the properties of r-broadcast will eventually reach every honest party with high probability. Moreover, since there exist at least $n - t = t + 1$ honest parties, D will be able to construct the certificate $\alpha^{\mathsf{D},\tau}$ and eventually broadcast the same. Therefore every honest party eventually receives $\alpha^{\mathsf{D},\tau}$ as well as the $(\mathtt{OK}, \mathsf{D})$ message. Moreover, P_{king} will be a privileged party and forwards $\{\mathbf{c}_{s_j}\}_{\sigma_D}, \{\mathbf{c}_{r_j}\}_{\sigma_D}$ and $\{\mathsf{Com}_{s_j}\}_{\sigma_D}$ to every P_j. It now follows easily that every honest P_j will eventually receive $\{\mathbf{c}_{s_j}\}_{\sigma_D}, \{\mathbf{c}_{r_j}\}_{\sigma_D}$ and $\{\mathsf{Com}_{s_j}\}_{\sigma_D}$ from P_{king} and obtain its share

pair s_j, r_j by decrypting $\mathbf{c}_{s_j}$ and $\mathbf{c}_{r_j}$. Moreover, since every such P_j forwards its $\{\mathsf{Com}_{s_j}\}_{\sigma_D}$ to every other party and there are at least $t+1$ such P_js, it follows that every honest party will eventually have at least $t+1$ committed shares with high probability, using which it will homomorphically obtain the remaining committed shares and terminate.

Now consider a corrupted D (and possibly a corrupted P_{king}) and let P_i be an honest party that terminates the protocol. We show that all other honest parties will eventually do the same. Since P_i terminated the protocol, it implies that P_i received $\alpha^{\mathsf{D},\tau}$ from the broadcast of D as well as $(\texttt{OK}, \mathsf{D})$ from P_{king}'s broadcast. From the properties of broadcast, it follows that with high probability, every other honest party will eventually receive them. In addition, since α^{D} was constructed, at least $t+1$ and hence at least one honest party, say P_h, must have received $\{\mathbf{c}_{s_j}\}_{\sigma_D}, \{\mathbf{c}_{r_j}\}_{\sigma_D}, \{\mathsf{Com}_{s_j}\}_{\sigma_D}, \{\mathsf{Com}_s\}_{\sigma_D}$ from D and successfully performed all the verifications. Since P_h is a privileged party, the rest of the proof follows using the same arguments as above, except that P_h plays the role of P_{king}.

CORRECTNESS: If some honest party, say P_i, has terminated the protocol, then it follows that it has received a valid certificate $\alpha^{\mathsf{D},\tau}$ from the broadcast of D, which implies that with overwhelming probability, there exists at least one honest party, say P_h, who would have participated in the construction of α^{D}. This further implies that P_h must have received $\{\mathbf{c}_{s_j}\}_{\sigma_D}, \{\mathbf{c}_{r_j}\}_{\sigma_D}, \{\{\mathsf{Com}_{s_j}\}_{\sigma_D}\}_{j\in[1,n]}$ and $\{\mathsf{Com}_s\}_{\sigma_D}$ from D and successfully performed the required verifications. Particularly, P_h would have verified that there exist polynomials of degree at most t, say $\overline{\phi}(\cdot)$ and $\overline{\psi}(\cdot)$, such that $\mathsf{Com}_s = \mathsf{Commit}(\overline{\phi}(0), \overline{\psi}(0))$ and $\mathsf{Com}_{s_j} = \mathsf{Commit}(\overline{\phi}(j), \overline{\psi}(j))$. We define $\overline{s}$ to be $\overline{\phi}(0)$ and show that eventually $\overline{s}$ will be $[\cdot]$-shared. Since P_i has terminated the protocol, from the termination property of the protocol, it follows that each honest party will eventually terminate with its shares s_j and r_j and a vector of committed shares, so what remains is to show that they correspond to $[\overline{\phi}(0)]$. However, this follows from the properties of non-equivocation. Specifically, neither a corrupted D nor any corrupted party can send or forward any other encrypted and committed shares, different from $\{\mathbf{c}_{s_j}\}_{\sigma_D}, \{\mathbf{c}_{r_j}\}_{\sigma_D}$ and $\{\mathsf{Com}_{s_j}\}_{\sigma_D}$ respectively, to any honest P_j. Similarly, no corrupted party P_k can forward its committed share, different from $\{\mathsf{Com}_{s_k}\}_{\sigma_D}$, to any honest party. Thus with high probability, $\overline{s}$ will be $[\cdot]$-shared.

It follows easily that if D is honest then $\overline{s} = s$, as in this case the polynomials $\overline{\phi}(\cdot)$ and $\overline{\psi}(\cdot)$ are the same as $\phi(\cdot)$ and $\psi(\cdot)$, as selected by D. Moreover it follows easily that if P_{king} is honest then it will be a privileged party, since an honest P_{king} will broadcast the $(\texttt{OK}, \mathsf{D})$ message only after receiving $\{\mathbf{c}_{s_j}\}_{\sigma_D}, \{\mathbf{c}_{r_j}\}_{\sigma_D}, \{\mathsf{Com}_{s_j}\}_{\sigma_D}, \{\mathsf{Com}_s\}_{\sigma_D}$ from D and successfully verifying it.

PRIVACY: We show that for an honest dealer D, and any $s, \overline{s}$ the adversary can not distinguish whether D shared s or $\overline{s}$. Since the non-equivocation signature $\{x\}_{\sigma_D}$ does not provide any information additional to the signed value x, we drop the tag for this part of the proof for the sake of readability. So let $\mathcal{T}_{cor}$ be the set of corrupted parties. Hence define $\mathcal{K}_{corr} := \{s_i, r_i, k_i \mid P_i \in \mathcal{T}_{corr},$ where s_i, r_i are the shares of party P_i and k_i its encryption and decryption keys$\}$ $\cup\{sk_i \mid sk_i$ is the signing key of $P_i\}$. Note that we only assumed authentic channels; there-

fore, it is easy to see that the view of the adversary $\mathsf{view}_A(x)$ during the execution of the protocol with secret x consists of $\mathsf{Com}_x, \{\mathsf{Com}_{x_j} = \mathsf{Commit}(x_j, r_{x,j}), \mathbf{c}_{x_j} = \mathsf{Enc}_{\mathsf{pk}_j}(x_j, \cdot), \mathbf{c}_{r_{x,j}} = \mathsf{Enc}_{\mathsf{pk}_j}(r_{x,j}, \cdot)\}_{j\in[1,n]}$ as well as $\alpha^{\mathsf{D},\tau}, (\mathtt{OK}, \mathsf{D})$ and the messages during the protocol executions of PoE.

Assume there is an adversary A that can distinguish whether $x = s$ or $x = \overline{s}$ is shared with non-negligible probability. We then show that there is an adversary that can distinguish Com_s from $\mathsf{Com}_{\overline{s}}$ with non-negligible probability. We do this in several steps; in particular, we define the following views and show that these are indistinguishable for s and $\overline{s}$.

- $\mathsf{view}^1_A(x) := \mathsf{view}_A(x) \cup \mathcal{K}_{corr}$
- $\mathsf{view}^2_A(x) := \mathsf{Com}_x, \{\mathsf{Com}_{x_j}, \mathbf{c}_{x_j}, \mathbf{c}_{r_{x,j}}\}_{j\in[1,n]}$ and $\mathcal{K}_{corr}$
- $\mathsf{view}^3_A(x) := \mathsf{Com}_x, \{\mathsf{Com}_{x_j}\}_{j\in[1,n]}$ and $\mathcal{K}_{corr}$
- $\mathsf{view}^4_A(x) := \mathsf{Com}_x$ and $\mathcal{K}_{corr}$
- $\mathsf{view}^5_A(x) := \mathcal{K}_{corr}$

Next, we show for $i \in [1, 4]$ that $\mathsf{view}^i_A(x) \sim \mathsf{view}^{i+1}_A(x)$. For each step we need to show that if for each adversary A^i having input $\mathsf{view}^i_A(x)$, there is an adversary A^{i+1} which has an indistinguishable output on input $\mathsf{view}^{i+1}_A(x)$.[5]

1. $\mathsf{view}^1_A(x) \sim \mathsf{view}^2_A(x)$

   Let A^1 be given, A^2 internally uses A^1 by computing $(\mathtt{OK}, \mathsf{D})$, computing $\alpha^{\mathsf{D},\tau}$ and simulating the zero-knowledge proofs PoE. The first part is trivial, the second part can be done since the message signed in α can be deduced from $\mathsf{view}^2_A(x)$ and A^2 has access to all signing key shares. In order to prove the last part we need to distinguish two cases: PoE executions with honest parties and PoE executions with corrupted parties. The executions with honest parties can be computed by A^2 using the simulator of the honest-verifier zero-knowledge property, since the honest parties choose their challenge randomly. For the corrupted parties, the adversary A^2 knows their s_i. Consequently, he can act as the dealer in PoE and use the adversary of the protocol execution in order to generate these proofs.

   Finally, we need to show that the output of A^1 is indistinguishable from the output of A^2. By construction of A^2 it is sufficient to show that the input given to A^1 inside A^2 is indistinguishable from the input that A^1 gets. For $(\mathtt{OK}, \mathsf{D})$ and $\alpha^{\mathsf{D},\tau}$ this is obvious. For the ZK proofs of honest parties, this is implied by the honest-verifier zero-knowledge. The zero-knowledge proofs of the corrupted parties consist of messages (a, c, e). Here a has the same distribution as in $\mathsf{view}^1_A(x)$, i.e., uniformly at random. The part c has the same distribution since A^2 internally invokes the adversary of the protocol execution. Finally, e is completely determined by a and c. Therefore e has the same distribution as well.

[5] Note that the adversary even knows the signing keys of the parties.

2. $\mathsf{view}^2_A(x) \sim \mathsf{view}^3_A(x)$

   In this step we basically remove the ciphertexts from the input of A^2. We construct A^3 by internally running A^2 on $\mathsf{view}^3(x)$ and the ciphertexts computed by A^3. In order to compute the ciphertexts A^3 needs to distinguish two cases, ciphertexts of corrupted and ciphertexts of honest parties. For corrupted parties, A^3 can simply access the plaintexts using $\mathcal{K}_{corr}$ and encrypt them as D does, hence having indistinguishability. The ciphertexts of honest parties are replaced by encryptions of 0s. By the IND-CPA property, it follows that this ciphertext is indistinguishable from the original message's ciphertext. Therefore the input to A^2 is indistinguishable to $\mathsf{view}^2(x)$ and consequently its output as well.

3. $\mathsf{view}^3_A(x) \sim \mathsf{view}^4_A(x)$

   In this step we remove all commitments except the commitment to x. The adversary $A^4(x)$ can compute the set $\{\mathsf{Com}_{x_j}\}$ for the corrupted parties P_j by recomputing them. For the other commitments, the adversary A^4 can interpolate the polynomial inside the commitments; since he has t values Com_{x_j} and the value Com_x this leads to a unique polynomial inside the commitments, i.e., $\{\mathsf{Com}_{x_j}\}_{j\in[1,n]}$. Then A^4 invokes A^3 on the computed input.

4. $\mathsf{view}^4_A(x) \sim \mathsf{view}^5_A(x)$

   Finally we want to remove the commitment Com_x. Since there are exactly t shares s_i in the adversaries' knowledge, any x can be used in order to determine a unique polynomial. By the computational hiding property, committing to any random value using uniform randomness cannot be distinguished from Com_x. Therefore $A^5(x)$ computes such a commitment and runs A^4 on this input.

We can conclude that $\mathsf{view}_A(s) \sim \mathsf{view}^5_A(s)$ and $\mathsf{view}_A(\overline{s}) \sim \mathsf{view}^5_A(\overline{s})$. Since we assume that $\mathsf{view}_A(s)$ is distinguishable from $\mathsf{view}_A(\overline{s})$, we can conclude that $\mathcal{K}_{corr}(s) = \mathsf{view}^4_A(s)$ is distinguishable from $\mathcal{K}_{corr}(\overline{t}) = \mathsf{view}^4_A(\overline{t})$. However, both $\mathcal{K}_{corr}(s)$ and $\mathcal{K}_{corr}(\overline{s})$ consists of the adversaries keys and — since D is honest — t values which are uniformly random. Therefore they cannot be distinguished (a contradiction). Hence the assumption has to be wrong and *privacy* follows by the contraposition.

COMMUNICATION COMPLEXITY: During the D-DEPENDENT PHASE, D has to non-equivocally distribute $\mathcal{O}(n)$ encrypted shares and committed shares to every party, which costs $\mathcal{O}(n^2\kappa)$ bits. Construction of the certificate α^{D} requires $\mathcal{O}(n^2\kappa)$ bits of communication, as there are n encrypted and committed shares and so D needs to execute in total n^2 instances of PoE. Broadcasting α^{D} costs $\mathcal{O}(n^2\kappa)$ bits of communication, as the certificate is of size $\mathcal{O}(\kappa)$ bits. During the D-INDEPENDENT PHASE, each party just needs to send one encrypted share and one committed share to every other party, incurring a communication of $\mathcal{O}(n^2\kappa)$ bits. □

3.4.2 Supervised Pre-multiplication Protocol

The Sup-PreMul-Sh protocol (Figure 3.4) takes input a $[\cdot]$-shared uniformally random and private value v and allows a dealer $\mathsf{D} \in \mathcal{P}$ to verifiably generate $[u]$ as well as $[u \cdot v]$ for a value u of his choice, under the supervision of a designated $P_{\mathsf{king}} \in \mathcal{P}$. As a *pre-condition*, the protocol assumes that every (honest) party is a privileged party with respect to the input $[v]$. The protocol ensures that v and $u \cdot v$ remains private, and when D is *honest* then $\mathcal{A}$ learns no new information on u. Finally if P_{king} is *honest* then it will be a privileged party with respect to $[u]$ as well as $[u \cdot v]$. The protocol always terminates for an *honest* D and P_{king} and has communication complexity $\mathcal{O}(n^2\kappa)$ bits.

Let $\{\mathbf{c}_{v_j}, \mathbf{c}_{r_j}, \mathsf{Com}_{v_j}\}_{j\in[1,n]}$ and Com_v be the encrypted shares and the committed shares corresponding to $[v]$ that is available to all the parties. Let $\phi(\cdot)$ and $\psi(\cdot)$ be the sharing and randomness polynomial corresponding to $[v]$. Thus $v_i = \phi(i)$, $r_i = \psi(i)$ is the share-pair available with P_i and $\mathbf{c}_{v_j} = \mathsf{Enc}_{\mathsf{pk}_j}(v_j), \mathbf{c}_{r_j} = \mathsf{Enc}_{\mathsf{pk}_j}(r_j), \mathsf{Com}_{v_j} = \mathsf{Commit}(v_j, r_j)$ and $\mathsf{Com}_v = \mathsf{Commit}(\phi(0), \psi(0))$. To generate $[u]$, D first invokes an instance of Sup-Sh. The next task for D would be to generate $[u \cdot v]$ and that too without knowing v. To do this, we observe that $u \cdot \phi(\cdot) + m(\cdot)$ and $u \cdot \psi(\cdot) + \hat{m}(\cdot)$ constitute correct sharing and randomness polynomial respectively for $[u \cdot v]$, where $m(\cdot)$ and $\hat{m}(\cdot)$ are random *masking polynomials* of degree at most t selected by D with the constraint $m(0) = \hat{m}(0) = 0$. This is because $u \cdot \phi(\cdot) + m(\cdot)$ and $u \cdot \psi(\cdot) + \hat{m}(\cdot)$ will have degree at most t, with the constant term of $u \cdot \phi(\cdot) + m(\cdot)$ being $u \cdot v$. Thus $w_j = u \cdot v_j + m(j)$ and $\hat{r}_j = u \cdot r_j + \hat{m}(j)$ constitute valid share-pair for $[u \cdot v]$ and so by using the homomorphic property of encryption and commitment, D can compute the encrypted shares and committed shares of $[u \cdot v]$ and distribute the same to the parties; the presence of masking polynomials preserves the privacy of u and $u \cdot v$.

The rest of the protocol is now similar to Sup-Sh, except that PoCM is used instead of PoE by D to construct the certificate that it has done "correct pre-multiplication". As nothing about the masking polynomials is revealed, u remains private. The properties of the protocol are stated in Lemma 3.4.2 followed by its proof.

Lemma 3.4.2. *Let v be a completely random and unknown value which is $[\cdot]$-shared among $\mathcal{P}$ and let u be a value selected by D. Then for every possible $\mathcal{A}$ and scheduler, protocol* Sup-PreMul-Sh *achieves the following properties up to a negligible probability in κ:*

(1) TERMINATION*: if D and P_{king} are honest then all the honest parties eventually terminate the protocol. Moreover, if some honest party terminates the protocol, then every other honest party eventually does the same.*

(1) CORRECTNESS*: if some honest party terminates the protocol, then there exists a value $\overline{u}$, such that $\overline{u}$ and $\overline{u} \cdot v$ will eventually be $[\cdot]$-shared among the parties. If D is honest then $\overline{u} = u$. Moreover if P_{king} is honest then P_{king} will be a privileged party with respect to $[\overline{u}]$ as well as $[\overline{u} \cdot v]$.*

(1) PRIVACY*: v and $u \cdot v$ remains private at the end of the protocol. Additionally, if D is honest then u also remains private.*

(1) COMMUNICATION COMPLEXITY*: the protocol has communication complexity of* $\mathcal{O}(n^2\kappa)$ *bits.*

Proof. 1. COMMUNICATION COMPLEXITY: The *generating* phase of the protocol consists of one instance of the Sup-Sh protocol which has communication complexity $\mathcal{O}(n^2\kappa)$. The D independent phase of the protocol is similar as in the Sup-Sh protocol and hence has communication complexity $\mathcal{O}(n^2\kappa)$. The same holds for the *share verification and certification* part of the protocol, a broadcast from the king with complexity $\mathcal{O}(n^2\kappa)$ complexity and n^2 instances of PoCM (n parties running n instances each). Hence this part has communication complexity $\mathcal{O}(n^2\kappa)$ as well. Finally, in the *share communication and certificate generation* part, the certificate β is broadcasted by the dealer and the dealer takes part in all executions of PoCM. In addition the dealer sends $3+2n$ commitments and n ciphertexts non-equivocally to every party. The broadcast and PoCM executions have communication complexity $\mathcal{O}(n^2\kappa)$ and the same holds for sending $\mathcal{O}(n)$ values of size $\mathcal{O}(\kappa)$ non-equivocally to every party. Consequently the overall protocol has complexity $\mathcal{O}(n^2\kappa)$.

2. TERMINATION: Termination of the *generating* $\langle u\rangle$ phase follows by the corresponding properties of the Sup-Sh protocol. For the remaining phases of the protocol the termination properties follow completely analogously to the corresponding properties of Sup-Sh since the protocol structure is essentially the same.

3. CORRECTNESS: For correctness we have to show that in the end there is an t-sharing $\bar{u}$ and $\bar{u}v$. We have to show that for an honest D it holds that $\bar{u} = u$ and that if the king is honest, he is privileged with respect to $[u]$ and $[u \cdot v]$.

   The correctness of the protocol Sup-Sh already implies that there is a t-sharing of $\bar{u}$ and that an honest D will share $u = \bar{u}$. In addition, the correctness of Sup-Sh implies that P_{king} is privileged with respect to $[u]$.

   Therefore we only need to show that upon termination, there is a t-sharing of $\bar{u} \cdot v$ and that an honest P_{king} is privileged with respect to this sharing. Since an honest P_{king} only broadcasts $(\texttt{approve}, \mathsf{D})$ when he has received $\{\mathsf{Com}_u\}_{\sigma_D}, \{\{\mathbf{c}_{u\cdot v_j+m_j}\}_{\sigma_D}, \{\mathbf{c}_{u\cdot r_j+\hat{m}_j}\}_{\sigma_D}\}_{j\in[1,n]}, \{\{\mathsf{Com}_{u\cdot v_j+m_j}\}_{\sigma_D}\}_{j\in[1,n]}$ and $\{\mathsf{Com}_{u\cdot v}\}_{\sigma_D}$, as a subset of this message received all necessary information to be privileged with respect to $\bar{u} \cdot v$. Finally, upon termination, every party P_i received $\{\mathbf{c}_{\bar{u}\cdot v_i+m_i}\}_{\sigma_D}$ and $\{\mathbf{c}_{\bar{u}\cdot r_i+\hat{m}_i}\}_{\sigma_D}$ as well as the corresponding commitment. For the same reason as in the proof of protocol Sup-Sh, these values correspond to the verified shares, i.e., belong to a degree t polynomial, and by the non-equivocation property, $\bar{u} = u$ is ensured.

4. PRIVACY: The privacy proof is threefold. First, we have to show that v remains private, i.e., communication during the protocol does not help in distinguishing the value of two different v. Second, we have to show that $u \cdot v$ remains private in the same sense and third,

we have to show that u is private if the dealer D is honest. As in the proof for Sup-Sh we drop the σ_D annotation here, since there is no additional information in the tag than the tagged value already provides regarding the privacy.

(a) v *remains private:* Assume an adversary A can distinguish v from v' after seeing the additional information of the Sup-PreMul-Sh execution for some u. Since the execution requires that $[v]$ (or $[v']$) is already shared, it follows that an adversary $\mathcal{B}_A$ can internally simulate an execution of Sup-PreMul-Sh using u and then invoke A in order to distinguish v from v'. Hence v an adversary does not gain any additional information about v by the execution of Sup-PreMul-Sh.

(b) $u \cdot v$ *remains private:* Assume an adversary A can distinguish $w = u \cdot v$ from $w' = u' \cdot v'$. Since D may be corrupted, D can choose $u = u' = 1$. As a consequence A can now distinguish the cases $v = w$ from $v' = w'$ contradicting the privacy of v. Hence $u \cdot v$ remains private as well.

(c) u *remains private if* D *is honest:* This case is more difficult than the other two cases since the protocol leaks more information about u than the protocol Sup-Sh executed on u. However, we can follow the privacy proof of Sup-Sh.

Assume there is an adversary A that distinguishes the protocol execution for some u and u'. In addition to the execution of Sup-Sh on u, A gets $\{\mathbf{c}_{u \cdot v_j + m_j}, \mathbf{c}_{u \cdot r_j + \hat{m}_j}, \mathsf{Com}_{u \cdot v_j + m_j}\}_{j \in [1,n]}$, $\mathsf{Com}_{u \cdot v}$, the executions of PoCM and $(\texttt{approve}, \mathsf{D})$ as well as $\beta^{\mathsf{D},\tau}$. As in the proof of privacy with respect to the protocol Sup-Sh, the information $(\texttt{approve}, \mathsf{D})$ and $\beta^{\mathsf{D},\tau}$ does not help the adversary in distinguishing u from u'.

Since $\mathsf{Com}_{u \cdot v}$ can be computed from $\mathsf{Com}_{u \cdot v_j + m_j}$, we know there is an adversary that distinguishes u from u' without the input $\mathsf{Com}_{u \cdot v}$. Using the zero-knowledge property we can also simulate the proofs leading to an indistinguishable outcome of A (by definition of zero-knowledge). As a consequence there is an adversary that distinguishes u from u' by seeing the output of Sup-Sh, $\mathbf{c}_{u \cdot v_j + m_j}, \mathbf{c}_{u \cdot r_j + \hat{m}_j}, \mathsf{Com}_{u \cdot v_j + m_j}$. Since the dealer D is honest, we can use the IND-CPA property to remove the ciphertexts $\mathbf{c}_{u \cdot v_j + m_j}, \mathbf{c}_{u \cdot r_j + \hat{m}_j}$ as we did in the privacy proof for Sup-Sh. Also following the privacy proof for Sup-Sh, we can reduce the input of the adversary to $\mathsf{Com}_{u \cdot v}$, which finally contradicts the computational hiding property of our commitment scheme. Consequently, there is no such adversary A.

□

3.5 Supervised Triple Generation

Protocol SupTripGen generates $[u], [v], [w]$ for a uniformly random and private multiplication triple (u, v, w) under the supervision of a king P_{king} with $\mathcal{O}(n^3\kappa)$ communication complexity.

It employs two sub-protocols, Sup-Second and Sup-FirAndThd.

3.5.1 Generating the Second Component of the Triple

Protocol Sup-Second generates $[v]$ for a uniformly random value v, unknown to $\mathcal{A}$, under the supervision of P_{king}. For a requirement clarified in the sequel, Sup-Second also ensures that each (honest) party becomes a privileged party with respect to $[v]$. In the protocol, each P_i invokes an instance $\mathsf{Sup\text{-}Sh}_i$ of Sup-Sh (as a D) to generate $[v^{(i)}]$ for a uniformly random $v^{(i)}$. Let $\mathcal{T}_{\mathsf{king}}$ be the set of first $t+1$ parties whose instance of Sup-Sh is terminated by P_{king}, and let $v = \sum_{P_k \in \mathcal{T}_{\mathsf{king}}} v^{(k)}$. As at least one party in $\mathcal{T}_{\mathsf{king}}$ is honest, v is uniformly random and private. Next, P_{king} broadcasts $\mathcal{T}_{\mathsf{king}}$ and every party waits until it terminates all $\mathsf{Sup\text{-}Sh}_k$ instances of $P_k \in \mathcal{T}_{\mathsf{king}}$; this ensures that every party obtains its share-pair and all committed shares of v.

As P_{king} is a privileged party for *every* $[v^{(k)}]$, it computes all encrypted shares of $[v]$, using the homomorphic property of encryptions. However, unlike P_{king}, other honest parties may be non-privileged for one or more $[v^{(k)}]$s, and thus may not compute the encrypted shares of $[v]$. The way out is that P_{king} "helps" other parties by broadcasting the encrypted shares of v, which costs $\mathcal{O}(n^3\kappa)$ bits. To confirm whether P_{king} indeed broadcasted the correct encrypted shares of v, P_{king} is also asked to non-equivocally forward the encrypted shares corresponding to each $[v^{(k)}]$ to every other party. We stress that this information is *not* broadcasted, and rather communicated over the *point-to-point* channels, which costs $\mathcal{O}(n^3\kappa)$ bits. Once this information is *non-equivocally received* by a party, it *re-computes* the encrypted shares of v, verifies them with the P_{king}'s broadcast, and broadcasts the verification result. If $t+1$ parties broadcasts "positively" for P_{king}, then at least one honest party must have successfully verified those encrypted shares; so every party terminates the protocol with $[v]$ and the broadcasted encrypted shares of $[v]$.

The protocol Sup-Second is presented in Fig. 3.5.

Protocol $\mathsf{Sup\text{-}Second}(P_{\mathsf{king}}, \tau)$: τ is the session id

I. SHARING RANDOM VALUES—Every party $P_i \in \mathcal{P}$ including P_{king} executes the following code:

1. Select a random value $v^{(i)}$ and invoke an instance $\mathsf{Sup\text{-}Sh}(P_i, \tau, P_{\mathsf{king}}, v^{(i)})$ of $\mathsf{Sup\text{-}Sh}$ as a D to generate $[v^{(i)}]$ under the supervision of P_{king}; let this instance of $\mathsf{Sup\text{-}Sh}$ be denoted as $\mathsf{Sup\text{-}Sh}_i$. Moreover, let $\phi_i(\cdot), \psi_i(\cdot), \{\mathbf{c}_{v_{i,j}}, \mathbf{c}_{r_{i,j}}\}_{j\in[1,n]}, \{\mathsf{Com}_{v_{i,j}}\}_{j\in[1,n]}$ and $\mathsf{Com}_{v^{(i)}}$ denote the sharing polynomial, randomness polynomial, encrypted share-pair, committed shares and commitment, generated during $\mathsf{Sup\text{-}Sh}_i$, where $v^{(i)} = \phi_i(0), v_{i,j} = \phi_i(j), r_{i,j} = \psi_i(j)$, $\mathbf{c}_{v_{i,j}} = \mathsf{Enc}_{\mathsf{pk}_j}(v_{i,j}, \cdot), \mathbf{c}_{r_{i,j}} = \mathsf{Enc}_{\mathsf{pk}_j}(r_{i,j}, \cdot), \mathsf{Com}_{v_{i,j}} = \mathsf{Commit}(v_{i,j}, r_{i,j})$ and $\mathsf{Com}_{v^{(i)}} = \mathsf{Commit}(\phi_i(0), \psi_i(0))$.
2. For $j \in [1, n]$, participate in the instance $\mathsf{Sup\text{-}Sh}_j$, invoked by P_j as a D.

II. COLLECTING AND DISTRIBUTING THE INFORMATION FOR THE FINAL OUTPUT—Only P_{king} executes the following code:

1. Include party P_k in an accumulative set $\mathcal{T}_{\mathsf{king}}$, which is initially $\emptyset$, if the instance $\mathsf{Sup\text{-}Sh}_k$ is (locally) terminated.
2. Wait till $|\mathcal{T}_{\mathsf{king}}| = t + 1$. Then using the linearity property of the encryption scheme, compute $\mathbf{c}_{v_j} = \boxplus_{P_k\in\mathcal{T}_{\mathsf{king}}}\mathbf{c}_{v_{k,j}}$ and $\mathbf{c}_{r_j} = \boxplus_{P_k\in\mathcal{T}_{\mathsf{king}}}\mathbf{c}_{r_{k,j}}$ for $j \in [1, n]$ and broadcast $\mathcal{T}_{\mathsf{king}}, \{\mathbf{c}_{v_j}, \mathbf{c}_{r_j}\}_{j\in[1,n]}$.
3. For every $P_k \in \mathcal{T}_{\mathsf{king}}$, forward the encrypted share-pairs $\{\mathbf{c}_{v_{k,j}}\}_{\sigma_k}, \{\mathbf{c}_{r_{k,j}}\}_{\sigma_k}$, for all $j \in [1, n]$, received as a privileged party from the dealer P_k during the instance $\mathsf{Sup\text{-}Sh}_k$, to every party in $\mathcal{P}$.

III. RESPONDING TO P_{king} AND TERMINATION—Every party $P_i \in \mathcal{P}$ including P_{king} executes the following code:

1. Include party P_k in an accumulative set $\mathcal{T}_i$, which is initially $\emptyset$, if the instance $\mathsf{Sup\text{-}Sh}_k$ is (locally) terminated.
2. Wait to receive $\mathcal{T}_{\mathsf{king}}$ and $\{\mathbf{c}_{v_j}, \mathbf{c}_{r_j}\}_{j\in[1,n]}$ from the broadcast of P_{king}.
3. Upon receiving $\{\mathbf{c}_{v_{k,j}}\}_{\sigma_k}, \{\mathbf{c}_{r_{k,j}}\}_{\sigma_k}$ for all $j \in [1, n]$ from P_{king} corresponding to each $P_k \in \mathcal{T}_{\mathsf{king}}$, wait till $\mathcal{T}_{\mathsf{king}} \subseteq \mathcal{T}_i$. Once $\mathcal{T}_{\mathsf{king}} \subseteq \mathcal{T}_i$, broadcast the message $(\mathtt{OK}, i)$ only if $\mathbf{c}_{v_j} = \boxplus_{P_k\in\mathcal{T}_{\mathsf{king}}}\mathbf{c}_{v_{k,j}}$ and $\mathbf{c}_{r_j} = \boxplus_{P_k\in\mathcal{T}_{\mathsf{king}}}\mathbf{c}_{r_{k,j}}$ holds for every $j \in [1, n]$.
4. Wait to receive the $(\mathtt{OK}, \star)$ message from the broadcast of at least $t + 1$ parties. Upon receiving, wait till $\mathcal{T}_{\mathsf{king}} \subseteq \mathcal{T}_i$ and then compute $v_i = \sum_{P_k\in\mathcal{T}_{\mathsf{king}}} v_{k,i}, r_i = \sum_{P_k\in\mathcal{T}_{\mathsf{king}}} v_{k,i}, \{\mathsf{Com}_{v_j} = \oplus_{P_k\in\mathcal{T}_{\mathsf{king}}}\mathsf{Com}_{v_{k,j}}\}_{j\in[1,n]}$ and $\mathsf{Com}_v = \oplus_{P_k\in\mathcal{T}_{\mathsf{king}}}\mathsf{Com}_{v^{(k)}}$, where $v_{k,i}, r_{k,i}$ denotes the share-pair obtained at the end of the instance $\mathsf{Sup\text{-}Sh}_k$ and $\{\mathsf{Com}_{v_{k,j}}\}_{j\in[1,n]}$ and $\mathsf{Com}_{v^{(k)}}$ denotes the vector of committed shares and the commitment obtained at the end of the instance $\mathsf{Sup\text{-}Sh}_k$. Finally, output $v_i, r_i, \{\mathbf{c}_{v_j}, \mathbf{c}_{r_j}\}_{j\in[1,n]}, \{\mathsf{Com}_{v_j}\}_{j\in[1,n]}$ and Com_v and terminate.

Figure 3.5: Supervised generation of $[v]$ for a random v under the supervision of P_{king}; if the protocol terminates then each party will be a privileged party and will have all n encrypted shares of v.

Lemma 3.5.1. *For every possible $\mathcal{A}$ and every possible scheduler, protocol* Sup-Second *achieves the following properties up to a negligible probability in κ:*

(1) TERMINATION: *if P_{king} is honest, all honest parties eventually terminate the protocol. Even if P_{king} is corrupted and some honest party terminates, then every other honest party eventually does the same.*

(1) CORRECTNESS: *if the honest parties terminate the protocol, then the parties output $[\cdot]$-sharing $[v]$ of a value v. Moreover, each party will be a privileged party having all the encrypted share-pairs of v.*

(1) PRIVACY: *the output shared value will be random from the viewpoint of $\mathcal{A}$.*

(1) COMMUNICATION COMPLEXITY: *the protocol has communication complexity of $\mathcal{O}(n^3\kappa)$ bits.*

Proof. 1. TERMINATION: in order to show termination, we need to show two properties; first, if P_{king} is honest, then every honest party eventually terminates and second, if any honest party terminates, all other honest parties will do the same.

(a) *Honest king leads to termination:* if the king is honest, then the set $\mathcal{T}_{\mathsf{king}}$ will eventually reach the size $t+1$, because of Theorem 3.4.1 and since there are $t+1$ honest dealers in the executions of Sup-Sh. In particular, since the honest party P_{king} terminates for all executions corresponding to $\mathcal{T}_{\mathsf{king}}$, all honest P_i will eventually terminate for the same instances by the termination property of Sup-Sh. Since P_{king} is honest, the checks done by the honest parties in the *response and termination* phase will succeed and they will broadcast $(\texttt{OK}, \star)$. Thus, eventually the number of received $(\texttt{OK}, \star)$ broadcasts will reach $t+1$ and the honest parties terminate.

(b) *If an honest party terminates, then all honest parties do so:* let P_i be the honest party that terminates. We show that if a party P_j is honest, then P_j eventually terminates. The set $\mathcal{T}_i$ contains all parties for which P_i terminated the corresponding Sup-Sh when P_i terminated. By termination of the Sup-Sh protocol, it follows that eventually $\mathcal{T}_i \subseteq \mathcal{T}_j$. Since P_{king} broadcasts $\mathcal{T}_{\mathsf{king}}$ all parties will eventually receive the same $\mathcal{T}_{\mathsf{king}}$ and the condition $\mathcal{T}_{\mathsf{king}} \subseteq \mathcal{T}_j$ will eventually be satisfied. In addition, since P_i terminated, it received at least $t+1$ broadcasted messages $(\texttt{OK}, \star)$ which will — since these messages were broadcast — eventually arrive at P_j. Consequently, this condition is satisfied as well. Thus P_j finally terminates.

2. CORRECTNESS: At the end of the protocol execution every party outputs $v_i, r_i, \{\mathbf{c}_{v_j}, \mathbf{c}_{r_j}, \mathsf{Com}_{v_j}\}_{j\in[1,n]}$ and Com_v. There is an honest party that verifies that this message is a linear combination of the sharings run before. Since these precomputed sharings follow the correct protocol Sup-Sh and since sharings are linear, it follows that the parties hold a sharing of some value v when terminating.

3. PRIVACY: For all honest parties it holds that their sharing is indistinguishable for any value they shared, by the privacy of Sup-Sh. Since the overall output is a linear combination of $t+1$ sharings, at least one honest sharing is contained in this combination. Assuming the adversary A could distinguish this for any two different values v, v', then the adversary could use this to break the privacy of Sup-Sh, since he can control the t other parties from the output sharing. Thus, the privacy of Sup-Sh implies the privacy of Sup-Second.

   Moreover, the honest party of which a share is contained in the linear combination chooses this share uniformly at random. Consequently, the linear combination contains a value that is uniformly random as well.

4. COMMUNICATION COMPLEXITY: in the *Sharing random values* part of the protocol, there are n instances of the Sup-Sh protocol, i.e., a communication complexity of $\mathcal{O}(n^3\kappa)$.

   The *collection and distribution of the information* is done only by the party P_{king}. During the execution of this part of the protocol, the king broadcasts $\mathcal{T}_{\mathsf{king}}$ and $\{\mathbf{c}_{v_j}, \mathbf{c}_{r_j}\}_{j\in[1,n]}$ and forwards $\{\{\mathbf{c}_{v_{k,j}}\}_{\sigma_k}, \{\mathbf{c}_{r_{k,j}}\}_{\sigma_k}\}_{j\in[1,n]}$ for all $P_k \in \mathcal{T}_{\mathsf{king}}$. The broadcast message has size $\mathcal{O}(n\kappa)$ leading to a communication complexity of $\mathcal{O}(n^3\kappa)$ and the non-equivocally forwarded data has size $\mathcal{O}(n^2\kappa)$ leading to communication complexity $\mathcal{O}(n^3\kappa)$ as well.

   The final *response and termination* part which is executed by all parties consists only of broadcasting $(\texttt{OK}, i)$. This message has size $\mathcal{O}(1)$ because the identity is encoded in the broadcast protocol. However, there are n broadcasts in the worst case, leading to a communication complexity of $\mathcal{O}(n^3\kappa)$.

□

3.5.2 Generating First and Third Components of the Triple

Protocol Sup-FirAndThd takes as input a $[\cdot]$-shared uniformly random, say v, unknown to $\mathcal{A}$ such that *every party is a privileged party for* $[v]$. It then generates $[u]$ for a uniformly random u, unknown to $\mathcal{A}$, along with the $[\cdot]$-sharing $[u \cdot v]$, under the supervision of P_{king}. The protocol follows the same principle as Sup-Second, except that each party P_i now invokes an instance Sup-PreMul-Sh$_i$ of Sup-PreMul-Sh with $[v]$ and a uniformly random value $u^{(i)}$ to generate $[u^{(i)}]$ and $[u^{(i)} \cdot v]$; this is possible as every P_i is a privileged party with respect to $[v]$. Now the parties set $u = \sum_{P_k\in\mathcal{T}_{\mathsf{king}}} u^{(k)}$ and $w = \sum_{P_k\in\mathcal{T}_{\mathsf{king}}} u^{(k)} \cdot v$, where $\mathcal{T}_{\mathsf{king}}$ is the set of $t+1$ parties P_k such that the instance Sup-PreMul-Sh$_k$ has been terminated by P_{king}.

Protocol Sup-FirAndThd is presented in Figure 3.6.

Protocol Sup-FirAndThd($P_{\mathsf{king}}, [v], \tau$): τ is the session id

i. SHARING RANDOM VALUES—Every party $P_i \in \mathcal{P}$ including P_{king} executes the following code:

1. Select a random value $u^{(i)}$ and invoke an instance Sup-PreMul-Sh$(P_i, \tau, P_{\mathsf{king}}, \mathcal{P}, [v])$ of Sup-PreMul-Sh on $[v]$ as a D to generate $[u^{(i)}]$ and $[u^{(i)} \cdot v] = [w^{(i)}]$ under the supervision of P_{king}; let this instance of Sup-PreMul-Sh be denoted as Sup-PreMul-Sh$_i$. Moreover, let $\{u_{i,j}, \overline{r}_{i,j}\}_{j\in[1,n]}, \{\mathbf{c}_{u_{i,j}}, \mathbf{c}_{\overline{r}_{i,j}}\}_{j\in[1,n]}, \{\mathsf{Com}_{u_{i,j}}\}_{j\in[1,n]}$ and $\mathsf{Com}_{u^{(i)}}$ denote the vector of share-pairs, vector of encrypted share-pairs, vector of committed shares and the commitment corresponding to $[u^{(i)}]$ generated during Sup-PreMul-Sh$_i$. Similarly, let $\{w_{i,j}, \hat{r}_{i,j}\}_{j\in[1,n]}, \{\mathbf{c}_{w_{i,j}}, \mathbf{c}_{\hat{r}_{i,j}}\}_{j\in[1,n]}, \{\mathsf{Com}_{w_{i,j}}\}_{j\in[1,n]}$ and $\mathsf{Com}_{w^{(i)}}$ denote the vector of share-pairs, vector of encrypted share-pairs, vector of committed shares and the commitment corresponding to $[w^{(i)}]$ generated during Sup-PreMul-Sh$_i$.
2. For $j \in [1, n]$, participate in the instance Sup-PreMul-Sh$_j$, invoked by P_j as a D.

ii. COLLECTING AND DISTRIBUTING THE INFORMATION FOR THE FINAL OUTPUT—Only P_{king} executes the following code:

1. Include party P_k in an accumulative set $\mathcal{T}_{\mathsf{king}}$, which is initially $\emptyset$, if the instance Sup-Sh$_k$ is (locally) terminated.
2. Wait till $|\mathcal{T}_{\mathsf{king}}| = t + 1$. Then broadcast $\mathcal{T}_{\mathsf{king}}$.

iii. RESPONDING TO P_{king} AND TERMINATION—Every party $P_i \in \mathcal{P}$ including P_{king} executes the following code:

1. Include party P_k in an accumulative set $\mathcal{T}_i$, which is initially $\emptyset$, if the instance Sup-Sh$_k$ is (locally) terminated.
2. Wait to receive $\mathcal{T}_{\mathsf{king}}$ from the broadcast of P_{king}.
3. On receiving $\mathcal{T}_{\mathsf{king}}$, check if it is of size $t + 1$ and if so then wait till $\mathcal{T}_{\mathsf{king}} \subseteq \mathcal{T}_i$.
4. Compute $u_i = \sum_{P_k \in \mathcal{T}_{\mathsf{king}}} u_{k,i}, \overline{r}_i = \sum_{P_k \in \mathcal{T}_{\mathsf{king}}} \overline{r}_{k,i}, \{\mathsf{Com}_{u_j} = \oplus_{P_k \in \mathcal{T}_{\mathsf{king}}} \mathsf{Com}_{u_{k,j}}\}_{j\in[1,n]}, \mathsf{Com}_u = \oplus_{P_k \in \mathcal{T}_{\mathsf{king}}} \mathsf{Com}_{u^{(k)}}$, where $u_{k,i}, r_{k,i}, \{\mathsf{Com}_{u_{k,j}}\}_{j\in[1,n]}$ and $\mathsf{Com}_{u^{(k)}}$ is obtained at the end of Sup-PreMul-Sh$_k$, corresponding to $[u^{(k)}]$. Similarly compute $w_i = \sum_{P_k \in \mathcal{T}_{\mathsf{king}}} w_{k,i}, \hat{r}_i = \sum_{P_k \in \mathcal{T}_{\mathsf{king}}} \hat{r}_{k,i}, \{\mathsf{Com}_{w_j} = \oplus_{P_k \in \mathcal{T}_{\mathsf{king}}} \mathsf{Com}_{w_{k,j}}\}_{j\in[1,n]}, \mathsf{Com}_w = \oplus_{P_k \in \mathcal{T}_{\mathsf{king}}} \mathsf{Com}_{w^{(k)}}$, where $w_{k,i}, \hat{r}_{k,i}, \{\mathsf{Com}_{w_{k,j}}\}_{j\in[1,n]}$ and $\mathsf{Com}_{w^{(k)}}$ is obtained at the end of Sup-PreMul-Sh$_k$, corresponding to $[w^{(k)}]$. Finally, output $u_i, \overline{r}_i, \{\mathsf{Com}_{u_j}\}_{j\in[1,n]}, \mathsf{Com}_u$ as well as $w_i, \hat{r}_i, \{\mathsf{Com}_{w_j}\}_{j\in[1,n]}, \mathsf{Com}_w$ and terminate.

Figure 3.6: Supervised generation of $[u]$ and $[w = u \cdot v]$ for a random u under the supervision of P_{king}, where v is an existing $[\cdot]$-shared value, with every party being a privileged party with respect to $[v]$.

Lemma 3.5.2. *For every possible A and every possible scheduler, the protocol* Sup-FirAndThd

achieves the following properties up to a negligible probability in κ:

(1) TERMINATION: *if P_{king} is honest, then all honest parties eventually terminate. Moreover, if one honest party terminates, then all honest parties eventually terminate.*

(1) CORRECTNESS: *after termination, all parties hold a sharing $[u], [w]$ such that $[w] = [u \cdot v]$.*

(1) PRIVACY: *the view of the adversary is indistinguishable for different values of u and v.*

(1) COMMUNICATION COMPLEXITY: *the protocol has communication complexity of $\mathcal{O}(n^3\kappa)$ bits.*

Proof. The proof completely follows the proof of Sup-Second, except that properties are now implied from Sup-Sh, instead of Sup-PreMul-Sh. □

3.5.3 Sup-Second+Sup-FirAndThd$\Longrightarrow$SupTripGen

Finally SupTripGen consists of two steps: **(1).** The parties execute Sup-Second and output $[v]$; **(2).** On terminating Sup-Second, the parties execute Sup-FirAndThd, output $[u]$ and $[w] = [u \cdot v]$ and terminate (cf. Figure 3.7). Lemma 3.5.3 describes the SupTripGen properties, and follows easily from lemmas 3.5.1 and 3.5.2, and the protocol steps:

Lemma 3.5.3. *For every possible $\mathcal{A}$ and every possible scheduler, the protocol* SupTripGen *achieves the following properties up to a negligible probability in κ:*

(1) TERMINATION: *if P_{king} is honest then all honest parties eventually terminate the protocol. Moreover, even if P_{king} is corrupted and some honest party terminates the protocol, then every other honest party eventually does the same.*

(1) CORRECTNESS: *if the honest parties terminate the protocol, then the parties output $[\cdot]$-sharing $([u], [v], [w])$ of a multiplication triple (u, v, w).*

(1) PRIVACY: *the shared multiplication triple (u, v, w) will be random from the viewpoint of $\mathcal{A}$.*

(1) COMMUNICATION COMPLEXITY: *the protocol has communication complexity of $\mathcal{O}(n^3\kappa)$ bits.*

Protocol SupTripGen$(P_{\mathsf{king}}, \tau)$: τ is the session id

i. GENERATING THE SECOND COMPONENT OF THE TRIPLE—The parties in $\mathcal{P}$ execute an instance of Sup-Second$(P_{\mathsf{king}}, \tau)$ to generate a uniformly random $[\cdot]$-shared value, say $[v]$.

ii. GENERATING THE FIRST AND THIRD COMPONENT OF THE TRIPLE—On terminating the instance of Sup-Second$(P_{\mathsf{king}}, \tau)$, the parties execute Sup-FirAndThd$(P_{\mathsf{king}}, [v], \tau)$ to obtain $[u]$ and $[w = u \cdot v]$, output $([u], [v], [w])$ and terminate.

Figure 3.7: Supervised generation of a uniformally random $[\cdot]$-shared multiplication triple, unknown to $\mathcal{A}$, under the supervision of P_{king}.

3.6 The NeqAMPC Protocol

The key idea of the NeqAMPC protocol has already been discussed in Section 3.2. It is a sequence of three phases, where a party proceeds to the next phase only after completing the current phase:

Preprocessing Phase. To generate $c_M + c_R$ $[\cdot]$-shared random multiplication triples, the preprocessing phase protocol $\mathsf{PreProcess}$ performs the following steps: each party $P_i \in \mathcal{P}$ is asked to act as a king and invoke $\frac{c_M+c_R}{t+1}$ *parallel* instances of $\mathsf{SupTripGen}$ to generate $\frac{c_M+c_R}{t+1}$ random $[\cdot]$-shared multiplication triples under its supervision. The parties then execute an instance of ACS and agree on a common subset $\mathcal{T}_{\mathsf{king}}$ of $(n-t) = t+1$ kings whose instances of $\mathsf{SupTripGen}$ (as a king) will eventually be terminated by all the parties. The parties finally output the shared multiplication triples, generated during the instances of $\mathsf{SupTripGen}$, corresponding to the kings in $\mathcal{T}_{\mathsf{king}}$ and terminate; thus they will obtain $|\mathcal{T}_{\mathsf{king}}| \cdot \frac{c_M+c_R}{t+1} = c_M + c_R$ shared multiplication triples. As there exist at least $t+1$ honest parties, whose instances of $\mathsf{SupTripGen}$ as a king will be eventually terminated by all the (honest) parties (see Lemma 3.5.3), protocol $\mathsf{PreProcess}$ will eventually terminate. Similarly, as the shared triples generated in the instances of $\mathsf{SupTripGen}$ corresponding to each king in $\mathcal{T}_{\mathsf{king}}$ remain private, the output shared triples remain private. It is easy to see that $\mathsf{PreProcess}$ has communication complexity $\mathcal{O}(n \cdot \frac{c_M+c_R}{t+1} \cdot n^3\kappa) = \mathcal{O}((c_M + c_R)n^3\kappa)$ bits as $t = \Theta(n)$. As the protocol is quite straightforward, we skip the formal details.

Input Phase. The goal of the input phase protocol Input is to allow each individual party $P_i \in \mathcal{P}$ to generate $[\cdot]$-sharing of its private input x_i for the computation. For this each party $P_i \in \mathcal{P}$ invokes an instance of the sharing protocol Sh (see Section 3.3) as D to generate $[x_i]$. To avoid indefinite waiting, the parties execute an instance of ACS and agree on a common subset of $(n-t)$ parties, say $\mathcal{CORE}$, whose instances of Sh (as a dealer) will eventually be terminated by all the parties. The parties finally output the sharings, generated during the instances of Sh, corresponding to the parties in $\mathcal{CORE}$; on behalf of the remaining parties in $\mathcal{P} \setminus \mathcal{CORE}$, a default $[\cdot]$-sharing of 0 is considered. As there exist at least $t+1$ honest parties, whose instances of Sh as a dealer will eventually be terminated by all the (honest) parties, protocol Input will eventually terminate. The shared inputs generated in the instances of Sh corresponding to the *honest* parties in $\mathcal{CORE}$ remain private due to the privacy property of Sh. The Input protocol runs n instances of Sh, and has communication complexity of $\mathcal{O}(n \cdot n^2\kappa) = \mathcal{O}(n^3\kappa)$ bits. Again as the protocol is quite straight-forward, we skip the formal details.

Computation Phase. The computation phase protocol $\mathsf{Compute}$ performs the shared circuit evaluation on a gate-by-gate basis, by maintaining the following *invariant* for each gate of the circuit: given the $[\cdot]$-sharing of the input(s) of a gate, the protocol allows the parties to securely compute the $[\cdot]$-sharing of the output of the gate. The invariant is trivially maintained for the addition (linear) gates in the circuit, thanks to the linearity property of $[\cdot]$-sharings. For a multiplication gate, the invariant is maintained by applying the Beaver's circuit randomization

technique and using a $[\cdot]$-shared multiplication triple from the pre-processing stage (recall from Section 3.2). For a random gate, a $[\cdot]$-shared multiplication triple from the pre-processing stage is considered and the first component of the triple is associated with the random gate. Finally, once the $[\cdot]$-sharing $[y]$ of the circuit output y is generated, the parties execute the reconstruction protocol Rec, reconstruct y and terminate.

Again as the protocol is standard in the literature (see for example [51]), we omit the complete details and state only the main theorem here.

Theorem 3.6.1 (The NeqAMPC Theorem). *Let $f : \mathbb{Z}_p^n \to \mathbb{Z}_p$ be a function expressed as an arithmetic circuit over $\mathbb{Z}_p$, consisting of c_M multiplication gates and c_R random gates. Assume a non-equivocation oracle associated with every party. Then for every possible $\mathcal{A}$ and for every possible scheduler, there exists a computationally secure AMPC protocol to securely compute f with communication complexity $\mathcal{O}(((c_M + c_R) \cdot n^3 + n^3)\kappa)$ bits.*

3.7 Non-equivocation Implementations

Chun et al. [52] observed that the fundamental distributed computing problem of "Byzantine generals" has been proved unsolvable for three parties when one of those is corrupted [90] precisely because the corrupted party can spread contradictory messages to the remaining two honest parties. They demonstrated that if we can stop the corrupted party from *equivocating* (i.e., making conflicting statements to different honest parties) using a small trusted module on every party, it is possible to improve the resilience of distributed computing tasks in the asynchronous setting. They implemented non-equivocation using a (signed) trusted log abstraction called Attested Append-Only Memory (A2M), and designed a Byzantine-tolerant state machine replication (SMR) system for $n \geq 2t + 1$.

Levin et al. [92] further simplified the trust assumption from [52] and showed that a minimal trusted module called TrInc consisting of only a non-decreasing counter $c \in \mathbb{N}$ and a signing key-pair $(\mathsf{pk}, \mathsf{sk})$ is sufficient to generate A2M logs and to implement SMR with $n \geq 2t + 1$. Conceptually, TrInc provides a unique, once-in-a-lifetime attestations, and implements non-equivocation using the fact that the counter cannot be decreased, and consequently for every counter value c there is at most one message signed by the module.

Levin et al. implemented TrInc on Gemalto .NET SmartCards. It is also possible to implement TrInc over the computers enabled with TPM chips, where its features of trusted identity, sealed storage, and remote code attestation will be used. Although the TPM specification does not readily implement a trusted counter, it can be achieved using a TPM-based hypervisor framework such as TrustVisor [98]. Recently, Kapitza et al. [84] further simplified the TrInc design by replacing individual singing key-pairs with a replicated message authentication code (MAC) key.

Clement el al. [54] observed that the *definitional* non-equivocation itself actually does not provide any improvement to the resiliency bound; however, appropriately combining it with digital signature oracles (or MAC oracles with a key replicated across all oracles) provides the

improvements observed in [52, 92, 55, 84], where the transferability of verifications provided by signatures is a key along with non-equivocation.[6] They further noted that this combination (i.e., transferable non-equivocation) also provides a generic transformation that allows a crash fault tolerant protocol to tolerate the same number of Byzantine faults. Nevertheless, their generic transformation does not consider privacy (or confidentiality), required in the AVSS and AMPC tasks, and we observe in this paper that encryptions and zero-knowledge proofs are required along with signatures when privacy is required.

3.7.1 Realizing the Neq mechanism using TrInc

As discussed earlier, the Neq mechanism as described in Figure 3.1 can easily be realized using a TrInc implementation [92] (a simplified version of a TrInc device's state and API can be found in Figure 3.8). Here, we provide an informal proof for the realization.

Notation	Meaning
K_{priv}	Unique private key
K_{pub}	Public key corresponding to K_{priv}
A	Certificate of the device's validity
id	Identity of the current counter (Initially 0)
c	Current value of the counter

(a) State of a device

Function	Operation
CreateCounter()	Increases id. Sets c to 0 and returns id.
Attest(id,c',h)	Verifies that id is a valid counter with some value c and key K_{pub}. Verifies that $c \leq c'$. If any verification fails, return $\perp$, otherwise create an attestation $a =< \mathsf{COUNTER}, id, c, c', h >_{K_{priv}}$. Set $c = c'$. Return a.
getCertificate()	Returns (K_{pub}, A)

(b) API for the device

Figure 3.8: The API and the state of a simplified TrInc device.

We will use the fixed counter identity 1 for our protocols as we need only one TrInc instance for each party. In addition, we need to assume that the protocol steps are mapped to the natural numbers such that the order of these steps is preserved.

1. Implementation of a (Setup) invocation by party P_i: Invoke CreateCounter() on TrInc. Let i be the return value of the invocation. If $i \neq 1$ return $\perp$. Store the value oldL $= 0$. Invoke TrInc on GetCertificate() and let crt be the result. Send (Registered, P_i, crt) to all parties. Upon receiving the registered message, every party checks the validity of the crt certificate.

[6] They also prove that signatures standalone are not sufficient.

2. Implementation of a $(\mathsf{Neq\text{-}Sign}, P_i, l, m)$ invocation by party P_i: Invoke Attest on TrInc with the arguments $(1, l, m)$. Let a be the result. If $a = \perp$ return $\perp$, otherwise return $t = (a, \mathsf{oldL}, l)$ and update $\mathsf{oldL} = l$.

3. Implementation of a $(\mathsf{Neq\text{-}Verify}, P_i, l, m, \sigma)$ invocation by party P_j: If the message $(\mathsf{Registered}, P_i, crt)$ was not received by P_j, return 0. Otherwise split crt into (K, A) and σ into (s, oldL, l'). If $l' \neq l$ return 0, otherwise compute a signature verification on message $(\mathsf{COUNTER}, 1, \mathsf{oldL}, l, m)$ with signature s and verification key K and return the outcome of the verification.

In the beginning every counter has internal value 0 stored. The method $\mathsf{CreateCounter}()$ increases that value and returns it, as the counter for that identity is created now (with counter value 0). Therefore, the method always returns a value greater than 1 after the first invocation and consequently the setup returns $\perp$ as done by the Neq mechanism. For the first invocation, the setup sends the value $(\mathsf{Registered}, P_i, crt)$ where crt is necessary for the other methods to work. In particular, the checking of crt ensures that the certificate belongs to a TrInc device assuming that the manufacturer was honest and did not sign other devices or keys of this form with the same key.

The signing request $(\mathsf{Neq\text{-}Sign}, P_i, l, m)$ from P_i is done by invoking her TrInc device on $\mathsf{Attest}(1, l, m)$. The Attest method checks that 1 is a valid counter, i.e., if the setup was not done before it outputs $\perp$ which leads to the signing request returning $\perp$. The same holds for the Neq mechanism since $\perp$ is returned in case that the internal list L_i was not generated (by the setup) before. The next step of the attestation process is that the key (or label) l – which we map to an integer – is greater than the previous key value.[7] This check corresponds to the check that l is in the list L_i done by the Neq mechanism. And in the mechanism as in the implementation, a fail of the check leads to the outcome $\perp$. If all checks so far succeeded, the mechanism returns a random value and the TrInc implementation returns a triple t consisting of a signature a, the previous key oldL and l.

Finally, consider the implementation of the verification query. The mechanism returns 0 if the setup was not done before. The implementation does the same, however, since the $\mathsf{Registered}$ message can be forged, there is a different reason here. Since the setup creates the counter, there can only be a signature on any message $(\mathsf{COUNTER}, 1, \_, \_, \_)$ if the counter was created. This is implied by the unforgeability of the signature scheme. Combined with the requirement that the $\mathsf{Registered}$ message has to arrive beforehand, this implies that the setup method was executed (in particular the setup checks that the $\mathsf{Registered}$ message contains a valid TrInc certificate, i.e., there was no other "virtual" counter implemented). In the mechanism's case that $(l, m, \sigma) \in L_i$, i.e., the output 1 is done, the signing request was done before giving the corresponding output since there is no other way to add elements to the list L_i except the signing queries. Consequently, in the TrInc implementation's case, σ has the corresponding form and the signature verification succeeds leading to an output of 1, as well. If the mechanism

[7] This is slightly more restrictive than the Neq mechanism since we require the signature requests to be done by a certain order, but in order to use TrInc it seems to be necessary to require an order.

outputs 0 because $(l, m, \sigma) \notin L_i$ that means that m was never signed with key l by TrInc in the implementation. Thus the implementation outputs only 1 if the signature was forged, i.e., only with negligible probability.

As we have seen, the TrInc usage as described above implements the Neq mechanism with the restriction that the keys are ordered and the assumptions that the signature scheme require and that the TrInc manufacturer can be trusted. Finally, we note that it is easy to realize our Neq mechanism with the modified TrInc mechanism by Kapitza et al. [84] in the similar manner.

3.8 Instantiation of Various Primitives

In this section we instantiate the primitives we used in our protocol construction. These are the following: commitment scheme, encryption scheme and zero-knowledge proofs. As commitment scheme we simply use Pedersen commitments [105], i.e., we commit to m using randomness r by computing $g^m h^r$ for two generators g, h of a suitable group. In particular, this commitment scheme has the properties we required in section 3.1.3.

3.8.1 Encryption scheme Enc

We use the encoding-free ElGamal encryption scheme proposed in [50]. Let p, q be primes such that $q \mid p-1$ and let g be an integer of order pq modulo p^2 that generates a group $\mathbb{G} = \langle g \rangle$. Let $\langle x, y \rangle$ be the unique integer in $\mathbb{Z}_{pq}$ such that $\langle x, y \rangle = x \mod p$ and $\langle x, y \rangle = y \mod q$. The class of an element of $w = g^{\langle x,y \rangle} \in \mathbb{G}$ is x. We denote the class of w as $[\![w]\!]$. It is easy to see that $[\![w \cdot w']\!] = [\![w]\!] + [\![w']\!]$ and that $\mathbb{G}_p := \langle g \mod p \rangle$ has order q.

Definition 3.8.1 (Encoding-Free ElGamal [50])**.**

1. Setup*: Let p, q be primes such that $q \mid p-1$ and let g be a generator of $\mathbb{G}_p$ of order q.*
2. Key generation*: The private key is a random $x \in \mathbb{Z}_q$; the public key is $h = g^x \mod p$.*
3. Encryption*: The encryption algorithm chooses randomly an $r \in \mathbb{Z}_q$ and computes*

$$\mathsf{Enc}(m, r) = (g^r \mod p, m + [\![h^r \mod p]\!] \mod p)$$

4. Decryption*: The decryption works as follows:*

$$\mathsf{Dec}(x, (R, c)) = c - [\![R^x \mod p]\!] \mod p$$

The scheme is CPA-secure under the Decisional Class Diffie-Hellman problem [50] which is defined as the Diffie-Hellman problem under the class operation $[\![\cdot]\!]$. It has been shown that the Computational Class Diffie-Hellman problem is equivalent to the Computational Diffie-Hellman problem. However, the same result for the decisional case has not been shown.

We define two operations on ciphertexts in order to describe their homomorphic properties.

Definition 3.8.2 (Homomorphic operations)**.** *Let $x_1, y_1, x_2, y_2, v \in \mathbb{Z}_p$. Define the following operations:*
- $(x_1, y_1) \boxplus (x_2, y_2) := (x_1 \cdot x_2 \mod p, y_1 + y_2 \mod p)$
- $v \boxdot (x_1, y_1) := (x_1^v \mod p, v \cdot y_1 \mod p)$

Note that $[\![w]\!] + [\![w']\!] = [\![w \cdot w']\!]$ and that the latter operation can also be done by iteratively applying the first operation.

Lemma 3.8.1 (Homomorphic operations)**.** *The operations defined in Definition 3.8.2 implement the following operations on the plaintexts: Let $a, b, c, d, v \in \mathbb{Z}_q$.*
- $\mathsf{Enc}(a, b) \boxplus \mathsf{Enc}(c, d) = (g^{b+d} \mod p, (a + c) + [\![h^{b+d} \mod p]\!] \mod p)$
- $v \boxdot \mathsf{Enc}(a, b) = (g^{vb} \mod p, va + [\![h^{vb} \mod p]\!] \mod p)$

Proof. The proof is straight-forward considering definition 3.8.1. □

The encryption scheme has the properties required in section 3.1.3. For more details, we refer to [50].

3.8.2 Zero-knowledge Proof Schemes

In this subsection we will present a proof scheme using the primitives instantiated previously. The structure of the both ZK protocols is based on Σ-protocols [33]. These protocols have been well-studied and are usually easy to understand. Intuitively, a Σ-protocol is a proof that a party knows a witness w for a statement x such that $(x, w) \in \mathcal{R}$. The relation $\mathcal{R}$, which can be proven, is specific for the Σ protocol.

Definition 3.8.3 (Σ-Protocol [33])**.** *Let $\mathcal{R}$ be a relation. A Sigma Protocol for a relation $\mathcal{R}$ is a 3-round protocol, i.e., it consists of four algorithms (P_1, P_2, V_1, V_2) where P_1, P_2 is for the prover and V_1, V_2 is for the verifier such that the following holds. Let x, w be bitstrings and $(a, s_P) := P_1(x, w)$, $(c, s_V) := V_1(x, a)$, $p := P_2(c, s_P)$ and $d := V_2(s_V, p)$. Then the following holds:*

1. Completeness*: If $(x, w) \in \mathcal{R}$ then a protocol run using P_1, P_2, V_1, V_2 leads to $d = 1$.*
2. Special soundness*: There is an extraction algorithm E such that for any fixed statement x and for any two transcripts (a, c, p) and (a, c', p') such that the V_2 outputs 1 for both and where $c \neq c'$ holds, it follows that $(x, \mathsf{E}(a, c, c', p, p')) \in \mathcal{R}$.*
3. Special honest verifier zero-knowledge*: There is a simulator S such that for any x for which there is a w such that $(x, w) \in \mathcal{R}$ holds, the simulator produces on input x and random input c a transcript (a, c, p) which is computationally indistinguishable from a protocol transcript generated during the execution of the protocol using P_1, P_2, V_1, V_2.*

ZK Proofs technique: hiding the representation with respect to a fixed set of generators. We briefly recall this technique which was described in [45]. For a set of generators $\{g_i\}$

we prove that we know a set $\{x_i\}$ such that $y = \prod_i g_i^{x_i}$ by randomly choosing r_i values and sending $t = \prod_i g_i^{r_i}$ to the verifier. The verifier then sends a challenge c and the prover computes $s_i := r_i - cx_i$ and sends the s_i to the verifier. The verifier accepts iff $t = y^c \prod_i g_i^{s_i}$.

ZK-Proof Scheme PoE

The relation for PoE we need to prove is the following:

$$\exists m, r, r_1, r_2 : \mathsf{Com}_m = \mathsf{Commit}(m, r) \wedge \mathbf{c}_m = \mathsf{Enc}(m, r_1) \wedge \mathbf{c}_r = \mathsf{Enc}(r, r_2)$$

Using the instantiations we get the statement:

$$\exists m, r, r_1, r_2 : \mathsf{Com}_m = g^m h^r \wedge (\mathbf{c}_{m,1}, \mathbf{c}_{m,2}) = (g^{r_1}, m + [\![h^{r_1}]\!]) \wedge (\mathbf{c}_{r,1}, \mathbf{c}_{r,2}) = (g^{r_2}, r + [\![h^{r_2}]\!])$$

In order to prove the equations for Com_m and $\mathbf{c}_{m,1}, \mathbf{c}_{r,1}$, we use the technique mentioned above. For the remaining part, i.e., $\mathbf{c}_{m,2} = m + [\![h^{r_e}]\!]$ ($\mathbf{c}_{r,2}$ works analogously), we give a Σ-protocol following the idea of the one above. This is done by computing r_i and s_i as in [45], however, the t is computed and verified differently; this is done by computing t as $r_m + [\![h^{r_e}]\!]$ and verifying it to be $c \cdot \mathbf{c}_{m,2} + s_m + [\![h^{s_e}]\!]$. Given this construction it is straightforward to prove that the corresponding protocol is indeed a Σ-protocol.

Combining these two Σ-protocols we get a Σ-protocol for PoE.

Zero-knowledge Proof Scheme PoCM

As for the previous proof scheme, we will use a Σ-protocol in order to construct an interactive ZK proof scheme. The overall statement that we show is the following.

$$\begin{aligned}\exists u, \rho, m(\cdot), \hat{m}(\cdot), \{k_j, \hat{k}_j\}_{j\in[1,n]} : \mathsf{Com}_u = \mathsf{Commit}(u, \rho) \ \wedge \ \deg(m(\cdot)) \leq t \ \wedge \\ \deg(\hat{m}(\cdot)) \leq t \ \wedge \ m(0) = 0 = \hat{m}(0) \ \wedge \ \mathsf{Com}_{u\cdot v_j + m_j} = u \odot \mathsf{Com}_{v_j} \oplus \mathsf{Com}_{m_j} \ \wedge \\ \mathbf{c}_{u\cdot v_j + m_j} = u \boxdot \mathbf{c}_{v_j} \boxplus \mathsf{Enc}_{\mathsf{pk}_j}(m_j, k_j) \ \wedge \mathbf{c}_{u\cdot r_j + \hat{m}_j} = u \boxdot \mathbf{c}_{r_j} \boxplus \mathsf{Enc}_{\mathsf{pk}_j}(\hat{m}_j, \hat{k}_j)\end{aligned}$$

We need to mention here that it is unintuitive that $\mathbf{c}_{r_j}$ is multiplied by u instead of r. However, this is correct since we need to maintain the property that every party knows how to open their share, i.e., $\mathsf{Com}_{u\cdot v_j + m_j}$ which is derived from Com_{v_j} by exponentiating by u. Hence not only the value of v_j is multiplied by u but also the value of the corresponding randomness. This is also the reason why we need to rerandomize by adding Com_{m_j}; if we would not a party could try out whether u has some particular value u' by exponentiating the value Com_{v_j} by u' and equality-checking with the new value. Now, we show how to achieve a zero-knowledge construction for the overall statement.

The check for the polynomial degree can be done using the representation problem, e.g., in order to check that variables x and y satisfy the equation $2x + 5y = 1$ we can check that we know the representation of g^1 with respect to the base $\{g^2, g^5\}$. Consequently, we can check the degree by evaluating the (inverse) Vandermonde matrix on the quantified values and comparing the result with constants.

However, in order to give an instantiation with respect to the previously defined instantiations of Com and Enc we need to existentially quantify over the corresponding randomnesses as well. In addition, we split the proof into two statements which can be combined into a proof for the conjunction using standard techniques, i.e., using the same r_x, s_x for shared variables x.

The first relation can be proven using results described in [45]. We want to prove the knowledge of a representation of the commitments — with respect to the base of the Pedersen commitments — Com_u and Com_{m_j} together with an additional verification step to verify the property we require for $\mathsf{Com}_{u \cdot v_j + m_j}$. This additional check can be rephrased as follows:

$$X := \mathsf{Com}_{u \cdot v_j + m_j} \ominus \mathsf{Com}_{m_j} = (g^{v_j} h^{r_{v_j}})^u$$

where r_{v_j} is the randomness used to construct Com_{v_j} which is unknown to the prover.

Hence, we need to verify that we know the representation of X with respect to the base Com_{v_j} (and that this is the same as for the representation of Com_u). Therefore we can simply use the proof scheme [45], as for PoE, for the first part of the scheme.

However, it is not obvious that we can use this technique in order to verify $\mathbf{c}_{u \cdot v_j + m_j} = u \boxdot \mathbf{c}_{v_j} \boxplus \mathsf{Enc}_{\mathsf{pk}_i}(m_j, k_j)$. Looking at the first component of our instantiation of Enc leads to an equation of the form $X = Y^u \cdot g^{k_j}$ which corresponds to the representation knowledge problem, since the X is represented via the base $\{Y, g\}$ using u and k_j. Hence we can use the same technique. We need to show the correspondence for the second part as well. This part looks like

$$X = uY + m_j + k_j \cdot Z$$

here Y is second part of $\mathbf{c}_{v_j}$ and Z is $[\![h]\!]$. This is a linear version of the base representation problem and the technique can be applied here as well. The corresponding t constructed using randomness r_1, r_2, r_3 of this equation is $r_1 \boxdot \mathbf{c}_{v_j} \boxplus \mathsf{Enc}_{\mathsf{pk}_i}(r_2, r_3)$.

3.8.3 Asynchronous Reliable Broadcast (r-broadcast) Using Transferable Non-equivocation

The r-broadcast primitive allows a $\mathsf{S} \in \mathcal{P}$ to reliably send a message m to all the parties. Using transferable non-equivocation, it can achieved for $t < n$ [55, 54]. The high-level idea of this r-broadcast protocol is simple, and it follows from the crash-fault tolerant r-broadcast [74]: S first (non-equivocally) sends m_{σ_S} to all the parties; this prevents a corrupted S from sending different messages to different honest parties. However, a corrupted S can avoid sending m_{σ_S} to some honest parties. Thus, to ensure that all the honest parties eventually receive m_{σ_S}, whenever an honest party (non-equivocally) receives some message m_{σ_S}, before delivering the message, it once non-equivocally forwards it to every other party. This ensures that whenever an honest party receives m_{σ_S} then it will be eventually received by every other honest party. We present the pseudocode for the protocol r-broadcast in Figure 3.9.

Protocol r-broadcast(τ_b, m)

SENDER S (WITH INPUT m) —S executes the following code:

1. Send (τ_b, m_{σ_S}) to every party in $\mathcal{P}$.

PARTY P_i — Every party in $\mathcal{P}$, including S, executes this code:

1. If (τ_b, m_{σ_S}) is received from S or forwarded by some party $P_j \in \mathcal{P}$ (for the first time), then forward (τ_b, m_{σ_S}) to every party in $\mathcal{P}$, output (τ_b, m) and terminate.

Figure 3.9: Asynchronous r-broadcast protocol using transferable non-equivocation for $t < n$ [55, 54]

The properties of the protocol are summarized in Theorem 3.8.2, which follows immediately from the protocol description and the properties of transferable non-equivocation.

Theorem 3.8.2. *Protocol* r-broadcast *achieves the following properties for every instance τ_b, and every possible $\mathcal{A}$ and scheduler:*

(1) TERMINATION: *If* S *is honest, then all the honest parties eventually terminate the protocol. Moreover, even if* S *is corrupted and some honest party terminates the protocol, then except with negligible probability, every other honest party eventually terminates the protocol.*

(2) CORRECTNESS:

(A) *If* S *is honest then except with negligible probability, all honest parties output (τ_b, m).*

(B) *If* S *is corrupted and some honest party outputs (τ_b, m'), then except with negligible probability, all the honest parties output (τ_b, m').*

(3) COMMUNICATION COMPLEXITY: *The protocol incurs communication of $\mathcal{O}(n^2(\ell + \kappa))$ bits, where the message m is of size ℓ bits.*

3.9 Analysis of the AMPC Protocol of [32]

The AMPC protocol of [32] operates over $\mathbb{Z}_N$. The input stage consists of a synchronous broadcast round, where every party encrypts its input and broadcasts it, along with a non-interactive zero-knowledge (NIZK) proof that it knows the underlying plaintext, corresponding to the ciphertext. Thus the input stage consists of a broadcast of $\mathcal{O}(n\kappa)$ bits. The secure evaluation of the circuit is then done using the *king-slave* paradigm, where every party in $\mathcal{P}$ acts as a king and all the n parties (including the king) act as slaves and perform the computation on behalf of the king, so as to enable the king to obtain the output of the function (to be computed). So in principle, the actual circuit is evaluated n times, once on behalf of each party. We focus on the actual communication done among the slaves to evaluate the circuit on the behalf of a single king.

Due to the homomorphic property of the encryption scheme, evaluating the addition gates required no interaction among the slaves. For a multiplication gate, a random encrypted multiplication triple unknown to $\mathcal{A}$ is generated for the slaves, under the supervision of the king. For this, the parties begin with a publicly known default encrypted multiplication triple, which is then randomized to new encrypted triples, for $t + 1$ iterations, by different slaves; the triple obtained after $t + 1$th iteration is taken as the final triple. In every iteration, to perform the randomization of an encrypted triple, the king sends a randomization request to all the n slaves. A slave, on receiving a randomization request, performs the randomization, and to prove to the king that he has the performed the randomization correctly, the slave provides a NIZK proof of $\mathcal{O}(\kappa)$ bits to every other slave, so as to obtain a threshold signature. In short, in every iteration, each slave performs a randomization and communicates $\mathcal{O}(n\kappa)$ bits to the other slaves to prove that he has the performed the randomization correctly.

Each iteration involves a communication of $\mathcal{O}(n^2\kappa)$ bits in total, and so $t + 1$ iterations require a total communication of $\mathcal{O}(n^3\kappa)$ bits. Thus evaluating a single multiplication gate under the king requires a communication of $\mathcal{O}(n^3\kappa)$ bits and so for c_M multiplication gates, it will incur a total communication of $\mathcal{O}(c_M n^3\kappa)$ bits for a single king. Therefore, for n kings, the protocol will require an overall communication of $\mathcal{O}(c_M n^4\kappa)$ bits.

Protocol $\mathsf{Sh}(\mathsf{D}, \tau, s)$**:** τ **is the session id**

I. D**-Dependent Phase**:

Share Computation and Certificate Generation—Given the secret $s \in \mathbb{Z}_p$, D executes the following code:

1. Select t-degree polynomials $\phi(\cdot), \psi(\cdot)$ with $\phi(0) = s$. For $j \in [1, n]$, compute *share-pairs* $s_j = \phi(j), r_j = \psi(j)$, ciphertexts $\mathbf{c}_{s_j} = \mathsf{Enc}_{\mathsf{pk}_j}(s_j), \mathbf{c}_{r_j} = \mathsf{Enc}_{\mathsf{pk}_j}(r_j)$ and *committed share* $\mathsf{Com}_{s_j} = \mathsf{Commit}(s_j, r_j)$. Compute $\mathsf{Com}_s = \mathsf{Commit}(s, \psi(0))$.
2. Send messages $\{\mathbf{c}_{s_j}\}_{\sigma_\mathsf{D}}$, $\{\mathbf{c}_{r_j}\}_{\sigma_\mathsf{D}}$ and $\{\mathsf{Com}_{s_j}\}_{\sigma_\mathsf{D}}$ for all $j \in [1, n]$, and $\{\mathsf{Com}_s\}_{\sigma_\mathsf{D}}$ to each P_i. Start constructing a certificate $\alpha^{\mathsf{D},\tau} = \mathsf{certify}_{\mathsf{PoE}}(\mathsf{claim}_{\mathsf{D},\tau})$ claiming that "D has correctly shared a secret in session τ": This indirectly requires a proof that "$\exists \phi(\cdot), \psi(\cdot) : \mathsf{deg}(\phi(\cdot)) \leq t \ \wedge \ \mathsf{deg}(\psi(\cdot)) \leq t \ \wedge \ \mathbf{c}_{s_j} = \mathsf{Enc}_{\mathsf{pk}_j}(\phi(j), \cdot) \ \wedge \ \mathbf{c}_{r_j} = \mathsf{Enc}_{\mathsf{pk}_j}(\psi(j), \cdot) \ \wedge \ \mathsf{Com}_{s_j} = \mathsf{Commit}(\phi(j), \psi(j)) \ \wedge \ \mathsf{Com}_s = \mathsf{Commit}(\phi(0), \psi(0))$". For this, run an instance of PoE for each $(\mathbf{c}_{s_j}, \mathbf{c}_{r_j}, \mathsf{Com}_{s_j})$ triplet with every party P_i. Broadcast $\alpha^{\mathsf{D},\tau}$ once it is constructed.

Share Verification and Certification—Every party $P_i \in \mathcal{P}$ including D executes the following code:

1. On receiving $\{\mathbf{c}_{s_j}\}_{\sigma_\mathsf{D}}$, $\{\mathbf{c}_{r_j}\}_{\sigma_\mathsf{D}}$ and $\{\mathsf{Com}_{s_j}\}_{\sigma_\mathsf{D}}$ for all $j \in [1, n]$, and $\{\mathsf{Com}_s\}_{\sigma_\mathsf{D}}$ from D, perform the following verifications:
   (a) Verify if $\{\mathsf{Com}_{s_j}\}_{j \in [1,n]}$ and Com_s define unique t-degree polynomials.
   (b) If the above verification is successful, then participate in the PoE instances of D to verify $\mathsf{claim}_{\mathsf{D},\tau}$ and enable D to construct a certificate $\alpha^{\mathsf{D},\tau}$ for $\mathsf{claim}_{\mathsf{D},\tau}$.

II. D**-Independent Phase and Termination**—Every party $P_i \in \mathcal{P}$ including D executes the following code:

1. Upon receiving the broadcasted certificate $\alpha^{\mathsf{D},\tau}$ from D, verify $\alpha^{\mathsf{D},\tau}$ for $\mathsf{claim}_{\mathsf{D},\tau}$. Upon successful verification, if $\{\mathbf{c}_{s_j}\}_{\sigma_\mathsf{D}}$, $\{\mathbf{c}_{r_j}\}_{\sigma_\mathsf{D}}$ and $\{\mathsf{Com}_{s_j}\}_{\sigma_\mathsf{D}}$ for all $j \in [1, n]$ have been received from D,
   (a) then compute $s_i = \mathsf{Dec}_{\mathsf{sk}_i}(\mathbf{c}_{s_i}), r_i = \mathsf{Dec}_{\mathsf{sk}_i}(\mathbf{c}_{r_i})$, and $\forall j \in [1, n]$ forward *only* $\{\mathbf{c}_{s_j}\}_{\sigma_\mathsf{D}}, \{\mathbf{c}_{r_j}\}_{\sigma_\mathsf{D}}, \{\mathsf{Com}_{s_j}\}_{\sigma_\mathsf{D}}$ to P_j,
   (b) else wait for $\{\mathbf{c}_{s_i}\}_{\sigma_\mathsf{D}}$, $\{\mathbf{c}_{r_i}\}_{\sigma_\mathsf{D}}$, and $\{\mathsf{Com}_{s_i}\}_{\sigma_\mathsf{D}}$ to be forwarded by some party. Once received, compute the share-pair $s_i = \mathsf{Dec}_{\mathsf{sk}_i}(\mathbf{c}_{s_i})$ and $r_i = \mathsf{Dec}_{\mathsf{sk}_i}(\mathbf{c}_{r_i})$.
2. Forward $\{\mathsf{Com}_{s_i}\}_{\sigma_\mathsf{D}}$ to each P_j. On receiving $t + 1$ $\{\mathsf{Com}_{s_j}\}_{\sigma_\mathsf{D}}$, homomorphically compute $\{\mathsf{Com}_{s_j}\}_{j \in [1,n]}, \mathsf{Com}_s$ and terminate.

Protocol $\mathsf{Rec}(\mathsf{D}, \tau, [s])$**:** τ **is the session id**

Every party $P_i \in \mathcal{P}$ executes the following code:

1. Send the share-pair (s_i, r_i) to every party $P_j \in \mathcal{P}$.
2. On receiving the share-pair (s_j, r_j) from party P_j, verify whether $\mathsf{Com}_{s_j} \stackrel{?}{=} \mathsf{Commit}(s_j, r_j)$. If the verification is successful, then include s_j in a set $\mathcal{T}_j$, initialized to $\emptyset$.
3. Once $|\mathcal{T}_j| = t + 1$, construct a t-degree polynomial $\phi(\cdot)$ by interpolating the points $\{(j, s_j)\}_{s_j \in \mathcal{T}_j}$. Output $s = \phi(0)$ and terminate.

Figure 3.3: AVSS with $n = 2t + 1$ using Transferable Non-equivocation

Protocol $\mathsf{Sup\text{-}PreMul\text{-}Sh}(\mathsf{D}, \tau, P_{\mathsf{king}}, \mathcal{P}, [v])$**:** τ **is the session id**

Let $\{\mathbf{c}_{v_j}, \mathbf{c}_{r_j}, \mathsf{Com}_{v_j} = \mathsf{Commit}(v_j, r_j)\}_{j\in[1,n]}$ and $\mathsf{Com}_v = \mathsf{Commit}(v, \cdot)$ be information corresponding to $[v]$ available to *all* parties;

I. **Generating** $[u]$: On having a value u, D invokes an instance $\mathsf{Sup\text{-}Sh}(\mathsf{D}, \tau, P_{\mathsf{king}}, u)$ of Sup-Sh to generate $[u]$ under the supervision of P_{king} and every party in $\mathcal{P}$ participates in this instance and wait for its termination. Let $\mathsf{Com}_u = \mathsf{Commit}(u, \cdot)$ be the commitment of u which is computed and communicated during this instance of Sup-Sh (along with the other information corresponding to $[u]$).

II. **Generating** $[u \cdot v]$—The following code is executed by the respective parties only upon terminating the Sup-Sh instance of D:

a. D**-Dependent Phase**

1. **Share Computation and Certificate Generation**—The following code is executed only by D:
   (a) Select random *masking polynomials* $m(\cdot)$ and $\hat{m}(\cdot)$ of degree at most t with $m(0) = \hat{m}(0) = 0$. For $j \in [1, n]$, compute $m_j = m(j), \hat{m}_j = \hat{m}(j), \mathsf{Com}_{m_j} = \mathsf{Commit}(m_j, \hat{m}_j)$ and $\mathsf{Com}_m = \mathsf{Commit}(m(0), \hat{m}(0))$.
   (b) For $j \in [1, n]$, using the homomorphic property of the encryption and commitment scheme, compute the *new encrypted share pair* $\mathbf{c}_{u\cdot v_j+m_j} = u \boxdot \mathbf{c}_{v_j} \boxplus \mathsf{Enc}_{\mathsf{pk}_j}(m_j, \cdot), \mathbf{c}_{u\cdot r_j+\hat{m}_j} = u \boxdot \mathbf{c}_{r_j} \boxplus \mathsf{Enc}_{\mathsf{pk}_j}(\hat{m}_j, \cdot)$ and the *new committed share* $\mathsf{Com}_{u\cdot v_j+m_j} = u \odot \mathsf{Com}_{v_j} \oplus \mathsf{Com}_{m_j}$. In addition, compute the *new commitment* $\mathsf{Com}_{u\cdot v} = u \odot \mathsf{Com}_v \oplus \mathsf{Com}_m$.
   (c) Send messages $\{\mathbf{c}_{u\cdot v_j+m_j}\}_{\sigma_\mathsf{D}}, \{\mathbf{c}_{u\cdot r_j+\hat{m}_j}\}_{\sigma_\mathsf{D}}, \{\mathsf{Com}_{u\cdot v_j+m_j}\}_{\sigma_\mathsf{D}}$ and $\{\mathsf{Com}_{m_j}\}_{\sigma_\mathsf{D}}$ for all $j \in [1, n]$ and $\{\mathsf{Com}_u\}_{\sigma_\mathsf{D}}, \{\mathsf{Com}_{u\cdot v}\}_{\sigma_\mathsf{D}}$ and $\{\mathsf{Com}_m\}_{\sigma_\mathsf{D}}$ to each P_i. Start constructing a certificate $\beta^{\mathsf{D},\tau} = \mathsf{certify}_{\mathsf{PoCM}}(\mathsf{claim}_{\mathsf{D},\tau})$ claiming that "D has correctly done the pre-multiplication in session τ." This claim indirectly requires the following proof:

$$\exists u, \rho, m(\cdot), \hat{m}(\cdot), \{k_j, \hat{k}_j\}_{j\in[1,n]} : \mathsf{Com}_u = \mathsf{Commit}(u, \rho) \ \wedge \ \mathsf{deg}(m(\cdot)) \le t \ \wedge \ \mathsf{deg}(\hat{m}(\cdot)) \le t$$
$$\wedge \ m(0) = 0 = \hat{m}(0) \ \wedge \mathsf{Com}_{u\cdot v_j+m_j} = u \odot \mathsf{Com}_{v_j} \oplus \mathsf{Com}_{m_j}$$
$$\wedge \mathbf{c}_{u\cdot v_j+m_j} = u \boxdot \mathbf{c}_{v_j} \boxplus \mathsf{Enc}_{\mathsf{pk}_j}(m_j, k_j) \wedge \mathbf{c}_{u\cdot r_j+\hat{m}_j} = u \boxdot \mathbf{c}_{r_j} \boxplus \mathsf{Enc}_{\mathsf{pk}_j}(\hat{m}_j, \hat{k}_j)$$

   For this, run an instance of PoCM for

$$(\{\mathbf{c}_{v_j}, \mathbf{c}_{r_j}, \mathsf{Com}_{v_j}, \mathbf{c}_{u\cdot v_j+m_j}, \mathbf{c}_{u\cdot r_j+\hat{m}_j}, \mathsf{Com}_{u\cdot v_j+m_j}, \mathsf{Com}_{m_j}\}_{j\in[1,n]}, \mathsf{Com}_u, \mathsf{Com}_m)$$

   with every party P_i. Broadcast the certificate $\beta^{\mathsf{D},\tau}$ once it is constructed.
2. **Share Verification and Certification**—Every $P_i \in \mathcal{P}$ upon receiving $\{\mathbf{c}_{u\cdot v_j+m_j}\}_{\sigma_\mathsf{D}}, \{\mathbf{c}_{u\cdot r_j+\hat{m}_j}\}_{\sigma_\mathsf{D}}, \{\mathsf{Com}_{u\cdot v_j+m_j}\}_{\sigma_\mathsf{D}}$ and $\{\mathsf{Com}_{m_j}\}_{\sigma_\mathsf{D}}$ for all $j \in [1, n]$ and $\{\mathsf{Com}_u\}_{\sigma_\mathsf{D}}, \{\mathsf{Com}_{u\cdot v}\}_{\sigma_\mathsf{D}}$ and $\{\mathsf{Com}_m\}_{\sigma_\mathsf{D}}$ from D, participate (as a V) in the instance of PoCM with D to verify the claim $\mathsf{claim}_{\mathsf{D},\tau}$ and enable D to construct the certificate $\beta^{\mathsf{D},\tau}$ for $\mathsf{claim}_{\mathsf{D},\tau}$. Upon successful verification, broadcast the message $(\mathtt{approve}, \mathsf{D})$ if $P_i = P_{\mathsf{king}}$.

b. D**-Independent Phase and Termination**—Every party $P_i \in \mathcal{P}$ executes the following code:

1. Upon receiving the broadcasted certificate $\beta^{\mathsf{D},\tau}$ from D and the broadcasted message $(\mathtt{approve}, \mathsf{D})$ from P_{king}, verify $\beta^{\mathsf{D},\tau}$ for $\mathsf{claim}_{\mathsf{D},\tau}$. Upon successful verification:
   (a) If $\forall j \in [1, n], \{\mathbf{c}_{u\cdot v_j+m_j}\}_{\sigma_\mathsf{D}}, \{\mathbf{c}_{u\cdot r_j+\hat{m}_j}\}_{\sigma_\mathsf{D}}, \{\mathsf{Com}_{u\cdot v_j+m_j}\}_{\sigma_\mathsf{D}}$ have been received from D, then compute $w_i = \mathsf{Dec}_{\mathsf{sk}_i}(\mathbf{c}_{u\cdot v_i+m_i}), \hat{r}_i = \mathsf{Dec}_{\mathsf{sk}_i}(\mathbf{c}_{u\cdot r_i+\hat{m}_i})$, and forward *only* $\{\mathbf{c}_{u\cdot v_j+m_j}\}_{\sigma_\mathsf{D}}, \{\mathbf{c}_{u\cdot r_j+\hat{m}_j}\}_{\sigma_\mathsf{D}}, \{\mathsf{Com}_{u\cdot v_j+m_j}\}_{\sigma_\mathsf{D}}$ to each P_j,
   (b) else wait for $\{\mathbf{c}_{u\cdot v_i+m_i}\}_{\sigma_\mathsf{D}}$, $\{\mathbf{c}_{u\cdot r_i+\hat{m}_i}\}_{\sigma_\mathsf{D}}$, and $\{\mathsf{Com}_{u\cdot v_i+m_i}\}_{\sigma_\mathsf{D}}$ to be forwarded by some party. Once received, compute the share-pair $w_i = \mathsf{Dec}_{\mathsf{sk}_i}(\mathbf{c}_{u\cdot v_i+m_i})$ and $\hat{r}_i = \mathsf{Dec}_{\mathsf{sk}_i}(\mathbf{c}_{u\cdot r_i+\hat{m}_i})$.
2. Forward $\{\mathsf{Com}_{u\cdot v_i+m_i}\}_{\sigma_\mathsf{D}}$ to every P_j. On receiving $t + 1$ $\{\mathsf{Com}_{u\cdot v_i+m_i}\}_{\sigma_\mathsf{D}}$ messages, homomorphically compute $\{\mathsf{Com}_{u\cdot v_j+m_j}\}_{j\in[1,n]}, \mathsf{Com}_{u\cdot v}$ and terminate.

Figure 3.4: Supervised Pre-multiplication Protocol for generating $[u]$ **and** $[u \cdot v]$ **under** P_{king}

Chapter 4

Privacy Specification: Precedent-based Reasoning

In this chapter, we introduce PriCL: the first framework for expressing and automatically reasoning about privacy case law by means of precedent. PriCL is parametric in an underlying logic for expressing world properties, and provides support for court decisions, their justification, the circumstances in which the justification applies as well as court hierarchies. Moreover, the framework offers a tight connection between privacy case law and the notion of norms that underlies existing rule-based privacy research. In terms of automation, we identify the major reasoning tasks for privacy cases such as deducing legal permissions or extracting norms. For solving these tasks, we provide generic algorithms that have particularly efficient realizations within an expressive underlying logic. Finally, we derive a definition of deducibility based on legal concepts and subsequently propose an equivalent characterization in terms of logic satisfiability.

Precedent-based Reasoning. Privacy regulations such as HIPAA, COPPA, or GLBA in the United States impose legal grounds for privacy [102, 118, 119]. In order to effectively reason about such regulations, e.g., for checking compliance, it is instrumental to come up with suitable formalizations of such frameworks along with the corresponding automated reasoning tasks.

There are currently two orthogonal approaches to how regulations are expressed and interpreted in real life that both call for such a formalization and corresponding reasoning support. One approach is based on providing an explicit set of rules that define what is allowed and what is forbidden. The alternative is to consider precedents and case law, which is the approach predominantly followed in many countries such as the US. Precedents are cases that decide a specific legal context for the first time and thus serve as a point of reference whenever a future similar case needs to be decided. Moreover, even judges in countries that do not base their legal system on precedents often use this mechanism to validate their decision or shorten the process of argumentation.

Case law is particularly suitable for resolving vague formulations that naturally occur in privacy regulations like the definition of ‘disclosure’ in COPPA: “The term ‘disclosure’ means

[...] the release of personal information collected from a child in identifiable form". Here, case law could reference decisions that define what circumstances are qualified as a non-identifiable form of personal data, thereby aiding the user by providing judicially accurate interpretation of such terms.

While rule-based frameworks have received tremendous attention in previous research (see the section on related work below) there is currently no formalization for case law that is amenable to automated reasoning.

Our contribution. Our contribution to this problem space is threefold:

- We derive important legal concepts from actual judicial processes and relevant requirements from related work. The resulting framework PriCL, can be applied to the judicature of many different countries as it does not assume any specific argumentation.
- We tailor the framework for privacy regulations. In particular, our privacy specific case law framework is compatible with former policy languages since it has only minimal requirements regarding the logic. Therefore, it is possible to embed other formalizations into our framework.
- We define the major reasoning tasks that are needed to apply the framework to privacy cases. In particular, these tasks allow us to derive requirements for the underlying logic which we analyze. Several logics allow an embedding of the reasoning tasks by giving an equivalent characterization of the tasks. Consequently, we are able to select a well suited logic.

In total, the case law framework that we introduce gives a new approach for compliance with privacy regulations. In particular, it makes it possible to implicitly use any regulation if it was previously referenced by a judge. Moreover, it also provides for reasoning tasks in cases where no regulation is applicable but judicial precedents exist.

Related work. There are plenty of privacy regulations that companies are required to comply with. In the US there are regulations for specific sectors, e.g., HIPAA for health data, COPPA for children's data, or GLBA and RFPA for financial data. In the EU, the member states have general data protection codes. The legislative efforts to harmonize these national codes via the EU Data Protection Regulation [66] are proceeding and already provide for identifying legislative trends. The importance and impact of these privacy regulations has brought the interpretation thereof to the attention of more technically focused privacy research [89, 23, 5, 65, 40, 103].

Policy languages were mainly developed in order to model these regulations and to reflect companies' policies. Many of the modern logics modeling regulations are based on temporal logic [70, 25, 61, 117, 24] and were successfully used to model HIPAA and GLBA [62] and should be applicable to other regulations as well. While these logics focus on expressiveness in order to reflect the regulations, the logics for company policies focus on enforcement [16, 8] and thus also on authorization [4, 8]. Consequently, company policies are mostly based on access control policies [101, 85].

Bridging the gap between the regulation policies and the company's policies leads to automating compliance checks [112]. For many deployed policies, i.e., the ones that are efficiently enforceable, this is currently not possible due to the lack of decidability regarding the logics used to formalize regulations. However, for these cases there exist run-time monitoring tools that allow compliance auditing on log files [23, 70, 27, 25]. In particular, such auditing was invented for HIPAA [70].

A different approach for achieving compliance is guaranteeing privacy-by-design [97, 48, 77]. However, the policy of these systems still needs to be checked for compliance with the relevant privacy regulations.

There is also an orthogonal approach when designing privacy policies that focus on the end user, i.e., designing a policy that is formal and can be formulated in an user-understandable way [6]. First attempts using P3P [59, 109, 7] were unsuccessful. However, it is important to incorporate the user in the process of policy design in order to gain her trust [86, 69].

4.1 Ingredients

In the first step we illustrate which components are essential for a case law framework. To that end, we analyze actual judicial processes and derive ingredients for the framework from the relevant legal principles. In particular, the court decision and its justification give insights into how the decision is made and which judicial concepts have to be reflected by our framework. Hence, in the following, we analyze a representative court decision[1] and discuss the implications for our framework.

The conflict. *"This matter involves three certified questions from the Circuit Court of Harrison County regarding whether applicable state and federal privacy laws allow dissemination of confidential customer information by an insurance company to an unaffiliated third party during the adjustment or litigation of an insurance claim."*

Every case reaching a court is based on a conflict, i.e., there is some question, as the one above, for which different parties have different opinions on its truth value. In the example case, the parties are a plaintiff, who was injured in a car accident, and an insurance company, which refused to disclose the home address of the other person involved in the accident. The insurance company claimed that to do so would violate the privacy provisions of the Gramm-Leach-Bliley-Act (GLBA) and the West Virginia Insurance Commission's Privacy Rule. As a requirement for the framework, we can conclude that there has to be a conflict that needs to be resolved by a decision. This decision can be an arbitrary statement; hence, we call it a *decision formula*.

Sub-cases. A decision's justification usually involves decisions of several *sub-cases* in order to arrive at the final decision formula, e.g. the court needs to decide whether a specific law is applicable before examining what follows from its application. Each of these individual sub-case decisions may become a precedent for decisions which deal with a similar sub-case.

[1] The quotes are taken from MARTINO v. BARNETT, Supreme Court of Appeals of West Virginia, No. 31270, Decided: March 15, 2004. The decision text is public at `http://caselaw.findlaw.com/wv-supreme-court-of-appeals/1016919.html`.

The circumstances. *"[The plaintiff] concedes that under the definitions of the GLBA [...] information he requests is technically nonpublic personal information of a customer which the Act generally protects from disclosure to nonaffiliated third parties."*

Every case contains some factual background. These facts constitute some statements which are not under discussion but measurably true, e.g., that an address is nonpublic personal information. We summarize these facts in a *case description*.

Referencing related court decisions. *"[T]he United States District Court for the Southern District of West Virginia handed down an opinion in Marks v. Global Mortgage Group, Inc., 218 F.R.D. 492 (S.D.W.Va.2003), providing us with timely and pertinent considerations."*

The key of case law is referencing other cases in order to derive statements. In the example case, this capability is used to introduce an argumentation from a different court. This mechanism is also used when statements are derived from regulations as in the following example:

"[T]he GLBA sets forth a procedure whereby financial institutions falling within the purview of the Act may not disclose nonpublic personal information without first notifying its clients of the financial institution's disclosure policies and affording them the opportunity to bar any disclosure of such information by 'opting out.' See 15 U.S.C. 6802(a) and (b)."

Consequently, the framework has to be capable of introducing statements during the case justification by *references* to their origin.

Argumentation structure of the justification. *"[The] GLBA provides exceptions to its notification and opt-out procedures, including [...]"*

The argumentation structure of the justification is not linear, i.e., of the form $A \Rightarrow B \Rightarrow \ldots \Rightarrow$. But the arguments can be ordered in a tree form. The exceptions stipulated by the GLBA are enumerated and then discussed in the case justification. If more than one is applicable, these may serve as *independent decision grounds*, each being a potential precedent in its own right.[2] As a consequence, we believe that a *proof tree* fits the overall structure best.

World knowledge. *"[We] conclude that nonpublic personal information may be subject to release pursuant to judicial process."*

In the argumentation, the court leaves to the reader's knowledge that the plaintiff's litigation actually is a "judicial process". These open ends in the argumentation are neither explicitly covered by a decision nor by a case reference. Therefore, we need some world knowledge KB_W that will cover these axiomatic parts of the argumentation.

Precedents and stare decisis. The doctrine of *stare decisis* (to stand by things decided) or binding precedents is unique to common law systems. The decisions of superior courts are binding for later decisions of inferior courts (*vertical stare decisis*). These binding precedents are applied to similar cases by analogy.

A special case is the binding nature of previous decisions on the same hierarchical level or by the deciding court itself (*horizontal stare decisis*). While the details of binding precedents of different courts on the same level is subject to an ongoing scholarly debate, a *court reversing itself* is a more infrequent occurrence but usually has high impact (for example, in the years

[2] O'Gilvie v. United States, 519 U.S. 79, 84 (1996).

1946-1992, the U.S. Supreme Court reversed itself in 130 cases[3]) and needs to be reflected in our framework.[4]

In addition to the binding precedent, there also exists the persuasive precedent: *"While we recognize that the decision of the Marks court does not bind us, we find the reasoning in Marks regarding a judicial process exception to the GLBA very persuasive and compelling"*.

Here, a court is not bound by an earlier decision, in our example because the earlier decision was made by an inferior court, but finds the argumentation so persuasive that it is voluntarily used as a precedent.

Stare decisis does not apply in civil law systems, like those of Germany or France. However, these systems have a *jurisprudence constante*, facilitating predictable and cohesive court decisions. Though civil law judges are not obliged to follow precedents, they may use prior decisions as persuasive precedents and oftentimes do so.

Material difference. Stare decisis only applies if the subsequent court has to decide on a case or sub-case that is similar to the precedent. Therefore, if the court finds *material difference* between the cases, it is not bound by stare decisis. In practice, judges may claim material difference on unwarranted grounds, which may lead to conflicting decisions of analogous cases within our framework. Thus, we need to be able to account for *false material difference*.

Involving court hierarchies. *"[W]e look initially to federal decisions interpreting the relevant provisions of the GLBA for guidance with regard to the reformulated question. However, the issue proves to be a novel one in the country since few courts, federal or state, have addressed the exceptions to the GLBA."*

For our framework we need to take into account court hierarchies to identify binding precedents. In common law jurisdictions, inferior courts are bound by the decisions of superior courts; in civil law jurisdictions superior courts usually have higher authority without being strictly binding. In federal states like the USA or Germany we need to account for parallel hierarchies on state and on federal levels. This complex hierarchy has significant implications on stare decisis. For example, state courts in the United States are not considered inferior to federal courts but rather constitute a parallel court system. While state courts must follow decisions of the United States Supreme Court on questions of federal law, federal courts must follow decisions of the courts of each state on questions of that state's law. If there is no decision on point from the highest court of a state, the federal courts must either attempt to predict how the state courts would resolve the issue, by looking at decisions from state appellate courts at all levels, or, if allowed by the constitutions of the relevant states, consult the statesupreme courts.

Hence, in our framework every case needs to be annotated by a court which is part of a *court hierarchy*, to identify the character of precedents, binding or potentially persuasive.

---

[3]Congressional Research Service — Supreme Court Decisions Overruled by Subsequent Decision (1992). `http://www.gpo.gov/fdsys/pkg/GPO-CONAN-1992/html/GPO-CONAN-1992-13.htm` The U.S. Supreme Court has explained its practice as follows: "[W]hen convinced of former error, this Court has never felt constrained to follow precedent." — Smith v. Allwright, 321 U.S. 649, 665 (1944)

[4]Federal and state supreme courts are allowed to overrule their own precedents. State Oil Co. v. Khan, 522 U.S. 3, 20 (1997); Freeman & Mills, Inc. v. Belcher Oil Co., 11 Cal. 4th 85, 93 (1995).

Ratio decidendi and obiter dicta. Regarding the court's decision text, we need to differentiate between two types of statements. The actual binding property of a precedent has only those statements and legal reasoning that are necessary for the rationale of the decision. These necessary statements as called *ratio decidendi* and constitute the binding precedent. Further statements and reasoning that are not essentially necessary for the decision are called *obiter dicta*. These are not binding but can be referenced as persuasive precedents.

For our reasoning framework we need to differentiate and annotate statements into these two different categories to correctly identify binding precedents.

4.2 Defining The PriCL Framework

Reflecting the observations just made, we define cases (Section 4.2.1) and case law databases (Section 4.2.2). Thereby we also explain how to model the legal principles described in Section 4.1. Then, we define how the database can be used in order to deduce facts outside the framework (Section 4.2.3). We analyze our framework, validating a number of basic desirable properties of case law databases (Section 4.2.4). We finally show, for privacy regulations specifically, that our framework matches the requirements identified by previous work [23] (Section 4.2.5).

Throughout this section, we assume an underlying logic in which world properties are expressed and reasoned about. Our framework is parametric with respect to the precise form of that logic. The requirements the logic has to fulfill are interpreting predicates as relations over objects, supporting universal truth/falseness (denoted respectively as $\top$ and $\bot$), conjunction (denoted $\wedge$), entailment (denoted $A \models B$ if formula A entails formula B), and monotonicity regarding entailment, i.e., if $A \models B$ then $A \wedge C \models B$ for any formula C. As an intuition when reading the following, the reader may assume we are using a first-order predicate logic.

4.2.1 Introducing Cases

As we have seen, a case consists of a decision formula, a case description, a court, and a proof tree. The first three components are straightforward to capture formally as formulas and a set element (courts are represented by a finite set Courts of court identifiers). Designing the proof tree is more involved since it needs to capture the judge's justification. We distinguish between different kinds of nodes in the tree depending on the role the respective statements play in the justification: Does a sentence make an axiomatic statement, or form part of the case description? Does it refer to a previous case, adopting a decision under particular prerequisites? Does it make an assessment on the truth of a particular statement (e.g., that a particular piece of information is or is not to be considered private) under particular prerequisites? All such statements are "standalone" in the sense that they are not implications of previous arguments in the justification at hand. We therefore reflect these "standalone" statements in the leaf nodes of the proof tree, categorized by the three different types of statements mentioned.

The inner nodes of the tree perform logical deductions from their children nodes, represent-

ing the reasoning inherent in the justification, i.e., the conclusions that are made until finally, in the tree root, the decision formula is reached. Thereby, every inner node is annotated by an arbitrary formula. We differentiate between two kinds of reasoning steps, AND-steps and OR-steps.The OR-steps reflect the principle of *independent decision grounds*, i.e., the cases that a judge increases legal certainty by listing arguments that all for themselves are sufficient for the conclusion. The AND-step is the natural conclusion steps that is used to ensure that the decision made is reached through the argumentation.

In order to avoid a recursive definition, we need a (possibly infinite) set of case identifiers $\mathcal{C}_I$. Throughout the paper we assume a fixed given set $\mathcal{C}_I$. This leads to the following definition:

Definition 4.2.1 (Case)**.** *A case C is a tuple* $(\mathsf{df}, \mathsf{CaseDesc}, \mathsf{ProofTree}, \mathsf{crt})$ *such that*

- df *is a formula that we call the* decision formula *of the case C.*
- $\mathsf{CaseDesc}$ *is a formula describing the case's circumstances.*
- $\mathsf{ProofTree}$ *is a (finite) tree consisting of formulas f where the formula of the root node is* df*. Inner nodes are annotated with* AND *or* OR *and leaves are annotated with $l \in \{\mathsf{Axiom}, \mathsf{Assess}\} \cup \{\mathsf{Ref}(i) \mid i \in \mathcal{C}_I\}$. Leaf formulas l are additionally associated with a* prerequisite *formula* pre*. For leaves annotated with* Axiom*, we require that* $\mathsf{pre} = l$*.*
- $\mathsf{crt} \in \mathsf{Courts}$*.*

For leaf formulas l, we refer to l as the node's fact*, and we will often write these nodes as* $\mathsf{pre} \rightarrow \mathsf{fact}$ *where* $\mathsf{fact} = l$*.*

By the prerequisites *of an inner node n with children nodes $n_1, \ldots, n_k$, denoted as* $\mathsf{pres}(n)$*, we refer to* $\bigvee_{1 \leq i \leq k} \mathsf{pres}(n_i)$ *if n is annotated by* OR *and* $\bigwedge_{1 \leq i \leq k} \mathsf{pres}(n_i)$ *if n is annotated by* AND*. The prerequisites of a case C are the prerequisites of the root node and denoted by* pres_C*. We define analogously the* facts *of a node and a case. We will often identify formulas with proof tree nodes. Given a case C, by* df_C *we denote the decision formula of C.*

Let $\mathbf{C}$ be a set of cases and $\mu : \mathbf{C} \rightarrow \mathcal{C}_\mathbf{I}$ a function. If for every reference $\mathsf{Ref}(i)$ *in $\mathbf{C}$, there is an $D \in \mathbf{C}$ with $\mu(D) = i$, we call the set $\mathbf{C}$* closed *under μ.*

We assume *world knowledge* common to all cases. In the example of argumentation ends in Section 4.1, it is assumed that the reader knows that the predicate $\mathsf{is\_judical\_process}$ holds for any case. Formally, the world knowledge is a formula KB_W (naturally, a conjunction of world properties) in the underlying logic.

Definition 4.2.1 is purely syntactic, imposing no restrictions on how the different elements are intended to behave. We will fill in these restrictions one by one as part of spelling out the details of our framework, forcing cases to actually decide a conflict and behave according to the legal principles. One thing the reader should keep in mind is that $\mathsf{pre} \rightarrow \mathsf{fact}$ is *not* intended as a logical implication. Rather, pre are the prerequisites that a judge took into account when making the assessment that fact (e.g., the privacy status of a piece of information) is considered to be true under the circumstances $\mathsf{CaseDesc} \models \mathsf{pre}$. The $\mathsf{pre} \rightarrow \mathsf{fact}$ dependencies thus

model the human element in case law, which we consider to be outside of what we can capture with formal logic. This solely captures human decisions such as trade-off decisions. However, the frameworks allows reasoning about consequence of such decisions. The formulas pres_C, and respectively facts_C, collect all prerequisites needed to apply the proof tree, and respectively all facts needed to execute the proof tree; axiom leaves act in both roles.

In principle, a case has the purpose to decide a formula df. However, while justifying that a formula holds, e.g., that a telecommunication company has to delete connection data after a certain amount of time, the court might decide other essential subquestions. In the given example, this could be that connection data is personal data. This concept is conveniently captured through the notion of *subcases*.

Definition 4.2.2 (Subcase)**.** *Let* $C = (\mathit{df}, \mathit{CaseDesc}, \mathit{ProofTree}, \mathit{crt})$ *be a case and* $n \in \mathit{ProofTree}$ *a node. Let* $\mathit{sub}(n)$ *be the subtree of* $\mathit{ProofTree}$ *with root node* n*. The case* $\mathit{sub}(C, n) := (n, \mathit{CaseDesc}, \mathit{sub}(n), \mathit{crt})$ *is a* subcase *of* C*.*

Another aspect that is of interest when referencing cases is the degree of abstraction. For example, one case could decide that a specific telecommunication company C has to delete connection information D of some user U after a specific time period t. The question of how this decision can be used in order to decide the question for different companies C' or different information D' is covered by the legal concept of material difference. For this work, we assume that a judge specifies the allowed difference in the prerequisites of a decision. However, it could also be modeled by introducing metrics and thresholds when referencing (sub-)cases.

Tailoring cases for privacy.. Our definition of cases, so far, is generic in the sense that it may be applied to any domain of law. To configure our framework to privacy regulations more specifically, a natural approach is to simply restrict the permissible forms of decision formulas. We explicitly leave out legal domains such as individualized sentencing or measuring of damages. Decisions in the privacy context are about whether or not a particular action is legal when executed on particular data. We capture this by assuming a dedicated predicate $\mathsf{is\_legal\_action}$, and restricting the decision formula to be an atomic predicate of the form $\mathsf{is\_legal\_action}(a)$, where a is an action from an underlying set $\mathsf{Actions}$ of possible actions treated as objects (constants) in the underlying logic. This can also be used in other legal domains, but it turns out to be sufficient to connect our formalization of privacy cases with other policy based approaches. Note that, in contrast to other policy frameworks, we do not need to add the context to the predicate, as the context is contained in the case, via nodes of the form *"if the transfer-action a has purpose marketing and the receiver is a third party, then* $\neg \mathit{is\_legal\_action}(a)$*"*. As decisions about the legality of actions are not naturally part of the common world knowledge KB_W, nor of the case description $\mathsf{CaseDesc}$ itself, our modeling decision is to disallow the use of $\mathsf{is\_legal\_action}$ predicates in these formulas. In other words, the world and case context describe the circumstances which are relevant to determining action legality, but they do not themselves define whether or not an action is legal. This yields the following definition:

Definition 4.2.3 (Privacy Case)**.** *Given world knowledge* KB_W *and action set*

Actions, a case $C = (df, CaseDesc, ProofTree, crt)$ *is a* privacy case *if* $df \in \{\neg is\_legal\_action(a), is\_legal\_action(a)\}$ *for some action* $a \in Actions$*, where the* is_legal_action *predicate is not used in either of* KB_W *or* $CaseDesc$*.*

Starting to fill in the intended semantics of cases, i.e., of the structures allowed as per Definition 4.2.1, we first capture the essential properties which a case needs to have to "make sense" as a stand-alone structure. Additional properties regarding cross-case structures will be considered in the next subsection. We will use the word "consistency" to denote this kind of property. The following definition captures the intentions behind cases:

Definition 4.2.4 (Case Consistency)**.** *Let* $C = (df, CaseDesc, ProofTree, crt)$ *be a case.* C *is* consistent *if the following holds (for all nodes* n *where* $n_1, \ldots, n_k$ *are its child nodes)*

(i) $KB_W \wedge CaseDesc \not\models \bot$

(ii) $KB_W \wedge CaseDesc \models pres_C$

(iii) $KB_W \wedge CaseDesc \wedge facts_C \not\models \bot$

(iv) $\bigwedge_{1 \leq i \leq k} n_i \models n$ *if* n *is an* AND *step*
and $\bigvee_{1 \leq i \leq k} n_i \models n$ *if* n *is an* OR *step*

Regarding (i), if the world knowledge contradicts the case description, i.e., $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \models \bot$, then the case could not have happened in reality. Similarly, (iii) the case context must not contradict the facts that the proof tree makes use of (this subsumes (i), which we kept as it makes the definition more readable). As for (ii), the case context must imply the axioms as well as the prerequisites which the present judge (assessments) or other judges (references to other cases; see also Definition 4.2.7) assumed to conclude these facts. (iv) says that inner nodes must represent conclusions drawn from their children (remember here that n_i, for leaf nodes $\mathsf{pre} \rightarrow \mathsf{fact}$, refers to fact).

The OR nodes of the proof tree reflect the legal argumentation structure of *independent decision grounds*, the judge gives several arguments, each of which is sufficient. If the judge of a later case decides that one of these arguments is invalid for the conclusion, he needs to be able to falsify only one of the branches and not the whole tree. In other words, the tree structure gives "syntactic sugar" that makes it possible to reflect the justification more closely and thereby marks which subsets of leaf nodes are sufficient in order to reach decision df.

4.2.2 Combining Cases to Case Law Databases

The quintessential property of case law is that cases make references to other cases. These references are necessary to formulate several legal principles of Section 4.1.

The legal principles *false material difference* and *reversing decisions* define requirements for when not to reference a case, either because it contains a mistake or because the opinion has changed over time. Therefore, we consider the design cleaner if both principles are covered by

the same mechanism of the framework. There are several options to model the principles: first, the reversed decision could be covered by time, i.e., by a requirement to refer to the newest case that is applicable regarding the circumstances. However, the false material difference cannot be covered by that. Another approach is to denote single Assess nodes as unwarranted, i.e., to forbid the reference to be used thereafter. This solution can model both principles *false material difference* and *reversing decisions*. We explicitly decided to model the mechanism of unwarranted nodes outside of the cases. Assume a case would decide that another decision was unwarranted. This leads to another decision that could potentially be marked as unwarranted later on implying that it is again correct to cite the case. Consequently, this would lead to a set of time intervals during which the citation of nodes is warranted. However, after legal consultation we figure out that this complication does not meet practice, i.e., once a decision is unwarranted it will not become warranted again; hence we simplified the mechanism.

We require a different mechanism to differentiate cases we must agree with and cases which we may use as reference. Unwarranting rather defines which decisions must not be referenced. In particular, we need to differentiate between assessments coming from the legal principles *ratio decidendi* and *obiter dicta*. While the part of the decision following *ratio decidendi* leads to a binding precedent, the *obiter dicta* part is not binding. Thus, we introduce predicates may-ref and must-agree. It also provides a mechanisms to respect the *court hierarchy*. Intuitively, may-ref(C_1, C_2) denotes the circumstances that case C_1 may reference case C_2; must-agree(C_1, C_2) analogously denotes that C_1 must agree with C_2.

In addition, we need to introduce the concept of time by a total order $\leq_t$ over cases. This concept allows us to formulate the requirement that references can only point to the past. Using all these constructs, we can define a case law database.

Definition 4.2.5 (Case Law Database (CLD))**.** *A* case law database *is a tuple* $\mathsf{DB} = (\mathbf{C}, \leq_t, \mathsf{must\text{-}agree}, \mathsf{may\text{-}ref}, \mu, U)$ *such that:*

- $\mathbf{C}$ *is a finite set of cases. We will also write* $C \in \mathsf{DB}$ *for* $C \in \mathbf{C}$.
- $\mu : \mathbf{C} \to \mathcal{C}_\mathbf{I}$ *is an injective function such that* $\mathbf{C}$ *is closed under* μ*. In the following we will also write* $\mathsf{Ref}(D)$ *for* $\mathsf{Ref}(i)$ *if* $\mu(D) = i$.
- *Let* $<_{\mathsf{ref}} := \{(C, D) \mid D \text{ contains a } \mathsf{Ref}(C) \text{ node}\}$ *and* $\leq_t$ *is an order that we call* time order *of the cases. It has to hold:*

$$\begin{matrix} \mathsf{must\text{-}agree} \subseteq \\ <_{\mathsf{ref}} \subseteq \end{matrix} \quad \mathsf{may\text{-}ref} \subseteq \leq_t \subseteq \mathbf{C} \times \mathbf{C}$$

- U *specifies the unwarranted nodes, i.e.,* $U : \mathbf{C} \to \mathbf{N}$ *is function such that*
  - $\mathbf{N}$ *is a subset of the nodes labelled with* Assess *or* Ref *in the cases* $\mathbf{C}$.
  - *The set increases monotonic, i.e.,* $C \leq_t D \implies U(C) \subseteq U(D)$.

  We denote the unwarranted *nodes of* DB *by* $U(\mathsf{DB}) := \bigcup_{C \in \mathbf{C}} U(C)$.

The function μ is used to remove the recursive definition of a case and enables us to connect cases via their individual semantics. The property of closedness of μ can be formally considered as

$$\mu(\bigcup_{C \in \mathbf{C}} \{D \mid D \in \mathbf{C} \wedge \mathbf{D} <_{\mathsf{ref}} \mathbf{C}\}) \subseteq \mu(\mathbf{C})$$

or in other words: every case identifier of a referenced case is also contained in the database.

Regarding the relations must-agree and the may-ref we made two design decisions. First, we require to not link must-agree and the actual references $<_{\mathsf{ref}}$. On the one hand, there might be precedents which are not applicable, but on the other hand, we want the freedom to define must-agree and may-ref only depending on the court hierarchy, i.e., independent of the satisfaction of some precedent's preconditions. The second design decision is to base these relations on cases instead of decision nodes. As for the first decision, the purpose is to make an instantiation of the definition only depending on the court, but we need to be careful regarding the principles *ratio decidendi* and *obiter dicta*. Since one of them is not binding, i.e., a must-agree and the other is. This differentiation can be achieved by replacing every case with a set of cases. We require this to be part of the modeling process. However, it is possible to automatically identify parts of the proof that are optional to reach the final decision in the root node.

We did not add further restrictions since they may depend on local law. For example, there is a *vertical stare decisis* in US law, implying that higher court decisions have to be considered. There is also the term of *horizontal stare decisis* that requires respecting siblings in the hierarchy. This principle does not necessarily hold, but is under discussion. However, the definition of must- and may-references allows modeling both.

Example 4.2.1 (Must-agree and may-references for a court hierarchy)**.** *Assume the set of courts* Courts *is partially ordered by* $\leq_{\S}$*, i.e., there is a court hierarchy. In this case, we could model* must-agree *by*

$$\textsf{must-agree} = \{(C_1, C_2) \mid C_i = (\textsf{df}_i, d_i, p_i, \textsf{crt}_i), i \in \{1,2\}, C_1 \leq_t C_2, \textit{ and } \textsf{crt}_1 \leq_{\S} \textsf{crt}_2\}$$

It is easy to see that the must-agree *predicate actually only depends on the* crt *and not on the other parameters of the proof. We call this property* court-dependency.

The key property of unwarranted decisions is that they are time dependent. In order to only use warranted decisions when referencing, we define warranted subcases as follows:

Definition 4.2.6 (Warranted Subcase)**.** *A subcase* (df, CaseDesc, ProofTree, crt) *is* warranted *with respect to a set* N *of nodes if the case* (df, CaseDesc, ProofTree$'$, crt) *is consistent where* ProofTree$'$ *is derived from* ProofTree *by replacing every precondition of a node* $n \in N$ *by* $\bot$*. If a case is not warranted, we call it* unwarranted.

It remains to define when a case law database can be considered to be consistent. To that end, we consider case references and conflicts between cases. Starting with the former, we obtain:

Definition 4.2.7 (Correct Case Reference)**.** *Let* DB *be a case law database and* $C = (\mathsf{df}, \mathsf{CaseDesc}, \mathsf{ProofTree}, \mathsf{crt})$ *a case in* DB*. A leaf node* $\mathsf{pre} \rightarrow \mathsf{fact}$ *in* $\mathsf{ProofTree}$ *annotated with* $\mathsf{Ref}(D)$ references correctly *if* $D_u = (\mathsf{fact}, \mathsf{CaseDesc}_D, \mathsf{ProofTree}_D, \mathsf{crt}_D)$ *is a warranted subcase of a case* $D \in \mathsf{DB}$ *w.r.t.* $U(C)$*,* $\mathsf{may\text{-}ref}(C, D)$ *holds and* $\mathsf{KB}_W \wedge \mathsf{pre} \models \mathsf{pres}_D$*. A case* C references correctly *if all its leaves annotate with* $\mathsf{Ref}(D)$ *reference correctly.*

Consider that, when referencing a (sub)case D as $\mathsf{pre} \rightarrow \mathsf{fact}$ from our case C at hand, we are essentially saying that the same argumentation applied in D can be applied in our case, to prove fact under circumstances pre. So we need to show that this applicability of arguments is actually given. This is ensured by $\mathsf{KB}_W \wedge \mathsf{pre} \models \mathsf{pres}_D$ because pres_D collects all prerequisites, axioms and otherwise, needed to apply D. Note that, if C is consistent, by Definition 4.2.4 (ii) it holds that $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \models \mathsf{pre}$ and thus $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \models \mathsf{pres}_D$. Note further that $\mathsf{KB}_W \wedge \mathsf{pre} \models \mathsf{pres}_D$ defines the role of pre as providing a condition sufficient to entail "the other judge's prerequisites". As the same applies recursively to the case references made in D, we know that pre (given KB_W and $\mathsf{CaseDesc}$) entails *all* judge decisions underlying the assessment fact. We will formalize this in Theorem 4.2.3.

We are now almost in the position to define consistency at the global level of the entire case law database. The last missing piece in the puzzle is to identify when cases should be considered to be in conflict — which naturally occurs in case law databases where different judges may make different decisions. We capture this through pairs of cases whose prerequisites are compatible, while their facts are contradictory:

Definition 4.2.8 (Case Conflict)**.** *Let* C_1 *be a case in* DB *and* C_2 *be a warranted case w.r.t.* $U(C_1)$*. We say that* C_1 *is* in conflict with C_2 *if and only if*

(i) $\mathsf{KB}_W \wedge \mathsf{pres}_{C_1} \wedge \mathsf{pres}_{C_2} \not\models \bot$

(ii) $\mathsf{KB}_W \wedge \mathsf{facts}_{C_1} \wedge \mathsf{facts}_{C_2} \models \bot$

(iii) $\mathsf{must\text{-}agree}(C_1, C_2)$

A case C *is in conflict with* DB *if there is a* $D \in \mathsf{DB}$ *s.t.* C *is in conflict with* D*.*

We ignore the case descriptions here, other than what is explicitly employed as axioms in the proof trees: we consider cases to be in conflict if one *could* construct a case (e.g., $\mathsf{pres}_{C_1} \wedge \mathsf{pres}_{C_2}$) which would make it possible to come to a contradictory decision.

Given these definitions, we achieve an implementation of the conflict decision respecting the circumstances and the argumentation by case consistency. We respect the principles of independent decision grounds, false material difference, and reversing decisions by referential consistency. Avoiding conflicts leads to the implementation of the concepts of binding and persuasive precedents and vertical stare decisis. Finally, by requiring reduced cases, we respect decisions regarding obiter dicta and ratio decidendi. We define case law database consistency as follows:

Definition 4.2.9 (Case law database consistency)**.** *A case law database* $\mathsf{DB} = (\mathbf{C}, \leq_t, \mathsf{must\text{-}agree}, \mathsf{may\text{-}ref}, \mu, U)$ *is*

(i) case-wise consistent *if every* $C \in \mathsf{DB}$ *is consistent,*

(ii) referentially consistent *if every* $C \in \mathsf{DB}$ *references correctly, and*

(iii) hierarchically consistent *if every* $C \in \mathsf{DB}$ *is not in conflict with* DB*.*

(iv) warrants consistently *if for every* C *holds:* $U(C)$ *contains all* $\mathsf{Ref}(D)$ *nodes where* D *is an unwarranted subcase w.r.t.* $U(C)$*.*

We call DB consistent *if it warrants consistently and is hierarchically, referentially and case-wise consistent.*

Items (i) and (ii) are simple element-wise extensions. Item (iii) captures the nature of conflicts given the time line and court hierarchy. A case can only respect earlier decisions, so if there is a conflict then we can only expect the newer case to respect the older one. Since the order $<_\S$ is only a partial order, both courts could be incomparable (neither $\mathsf{crt}_{C_1} <_\S \mathsf{crt}_{C_2}$ nor $\mathsf{crt}_{C_2} <_\S \mathsf{crt}_{C_2}$). Allowing such cases to be in conflict reflects the intuition that local court instances of independent states may have different opinions. Hence the requirement $\mathsf{crt}_{C_2} \geq_\S \mathsf{crt}_{C_1}$ for the newer case C_2 to not be decided at a court below C_1 (as opposed to requiring for the newer case C_2 to be decided at a court strictly above C_1).

4.2.3 Deriving Legal Consequences: Deducibility and Permissibility

In the following we assume that the predicates may-ref and must-agree of the DB do not depend on the case description, the decision formula or the proof tree, but are only court dependent, cf. Example 4.2.1. As a consequence, we know the value of these predicates for formula values and case descriptions which are not contained as a case in the database given only the court level of the case. In other words, we require an operation $\mathsf{DB} \cup \{C\}$ that puts C at the end of the timeline regarding $\leq_t$, assigns a fresh identifier $i \in \mathcal{C}_I$ to C with μ, uses as $U(C) := U(\mathsf{DB})$, and adopts must-agree, may-ref appropriately and is independent of the decision formula and the proof tree. This operation is needed to apply the framework to situations not contained in the database.

Obvious applications of our framework are advanced support for case search (based on logic operations over the case descriptions, decision formulas, etc.), and consistency checking (given a case C, is C consistent and does it reference correctly?). A more advanced task is to evaluate the legality of actions given the cases reflected in the database. For example, when designing a course administration system, one may ask "Am I allowed to store students' grades in the system?" Our formalism supports this kind of question at different levels of strength, namely:

Definition 4.2.10 (Deducibility and Permissibility)**.** *Let* $\mathsf{DB} = (\mathbf{C}, \leq_t, \mathsf{must\text{-}agree}, \mathsf{may\text{-}ref}, \mu, U)$ *be a consistent CLD, and* f *a formula. We say that* f *is* permitted *in* DB *under circumstances* $\mathsf{CaseDesc}$ *and court* crt *if there exists a case*

$C = (f, \mathsf{CaseDesc}, \mathsf{ProofTree}, \mathsf{crt})$ *such that* ProofTree *does not contain nodes labeled with* Assess*, and* $\mathsf{DB} \cup \{C\}$ *is consistent (where* C *is inserted at the end of the timeline* $\leq_t$*). We say that* f *is* uncontradicted *in* DB *under* CaseDesc *and* crt *if* $\neg f$ *is not permitted under* CaseDesc *and* crt*. We say that* f *is* deducible *if it is permitted and uncontradicted.*

For sets F *of formulas, we say that* F *is permitted in* DB *under* CaseDesc *and* crt *if there exists a set of cases* $\{C_f = (f, \mathsf{CaseDesc}, \mathsf{ProofTree}_f, \mathsf{crt}) \mid f \in F\}$ *such that every* $\mathsf{ProofTree}_f$ *does not contain nodes labeled with* Assess*, and* $\mathsf{DB} \cup \{C_f \mid f \in F\}$ *is consistent (where the* C_f *are inserted in any order at the end of the timeline* $\leq_t$*).*

It might be confusing at first why we attach to f the weak attribute of being "permitted" if we can construct a case supporting it. The issue is, both f and $\neg f$ may have such support in the same database. This follows directly from the freedom of different courts to contradict each other. If two courts at the same level decide differently on the same issue, then that is fine by our assumptions (the database is hierarchically consistent unless a lower court contradicts an earlier decision by a higher court), but it allows to come to contradictory conclusions. Hence, to qualify a formula f for the strong attribute of being "deducible", we require the database to permit f and to not permit its contradiction.

Note that permissibility and deducibility are not only dependent on the database, but also dependent on the circumstances CaseDesc and the court crt. For example, when we answer "was it legal to send data D to party P?", it matters for which purpose the data was sent. That information is contained in the CaseDesc. The court level has several interpretations here: the court might be chosen to match the local court of the party asking the question. But the court level can also be viewed as a level of confidence. Permissibility is a "stronger" guarantee for lower court instances, because we can then deduce without incurring conflicts to instances higher up. Hence lower court instances can be used to obtain permissibility "with high confidence", and contradictions "with low confidence". Vice versa, higher court instances can be used to obtain permissibility "with low confidence" and contradictions "with high confidence". Combinations of different court levels for testing the two sides of deducibility – being permitted and being uncontradicted – can be used for fine-grained trade-offs.

The concept of deducibility of a *set* F of formulas is interesting because, in general, this is not the same as deducing each formula in separation. In particular, while each of f and $\neg f$ may be permitted in the same database, $\{f, \neg f\}$ is never permitted because adding the hypothetical supporting cases necessarily incurs a hierarchical conflict. Permissibility of F is also not the same as permissibility of $\bigwedge_{f \in F} f$ because the latter makes a stronger assumption: all cases referred to in order to conclude $\bigwedge_{f \in F} f$ must have compatible prerequisites. So deducibility of formula sets forms a middle ground between individual and conjunctive deducibility. We formalize and prove this observation in the following theorem.

Theorem 4.2.2. *There is a consistent case law database* DB*, case description* CaseDesc *and court* crt*, such that there is a set* F *of formulas for each of the following properties (in* DB *under circumstances* CaseDesc *and court* crt*):*

(i) For every $f \in F$*,* f *is permissible and* F *is not permissible.*

(ii) F is permissible, but $\bigwedge_{f \in F} f$ is not permissible.

Proof. We define $\mathsf{CaseDesc} := A$ for a predicate A and consider the court set $\mathsf{Courts} = \{H_1^1, H_2^1, H^2\}$ such that $H^i <_{\S} H^j$ iff $i < j$ implies $\mathsf{must\text{-}agree}$ and $\mathsf{may\text{-}ref}$ as in example 4.2.1.

Let $\mathsf{Assess}(f)$ be a proof tree consisting of a single assessment node as root node that contains $\top \to f$ and, for a case C and a formula f, let $\mathsf{Ref}(C, f)$ be the proof tree consisting of a single case reference node that refers to C and contains the formula $\top \to f$. Let $B \neq A$ be some predicate. The database DB consists of the following cases:

- $C_1 = (p, \top, \mathsf{Assess}(p), H_1^1)$
- $C_2 = (\neg p, \top, \mathsf{Assess}(\neg p), H_2^1)$
- $C_3 = (A \Rightarrow B, \top, \mathsf{Assess}(A \Rightarrow B), H_1^1)$
- $C_4 = (B \Rightarrow \neg A, \top, \mathsf{Assess}(B \Rightarrow \neg A), H_2^1)$

The time order $\leq_t$ is given by $<$ on the indices. The database is obviously consistent. Let $\mathsf{crt} = H^2$. In the following, we construct the sets F for the respective statement of the theorem.

(i) Define the set $F := \{p, \neg p\}$. The formula p is permitted by DB for case description $\mathsf{CaseDesc}$ and court crt, since $(p, \mathsf{CaseDesc}, \mathsf{Ref}(C_1, p), \mathsf{crt})$ is a case as required by Definition 4.2.10. The same holds for $\neg p$.

Assume that F is permitted. Then there are cases $C_p, C_{\neg p}$ such that $\mathsf{DB} \cup \{C_p, C_{\neg p}\}$ is consistent. However, C_p and $C_{\neg p}$ are in conflict and are at the same court level, i.e., either $\mathsf{must\text{-}agree}(C_p, C_{\neg p})$ holds or $\mathsf{must\text{-}agree}(C_{\neg p}, C_p)$ depending on the order in which the cases are inserted in DB. As a consequence, $\mathsf{DB} \cup \{C_p, C_{\neg p}\}$ cannot be hierarchically consistent. Thus, that database cannot be consistent either. Therefore, F cannot be permitted.

(ii) Let $f_1 = A \Rightarrow B$, $f_2 = B \Rightarrow \neg A$, and $F = \{f_1, f_2\}$. It is easy to see that for a case $C_{f_1 \wedge f_2}$ it holds that $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \wedge \mathsf{facts}_C \models \bot$ if C_3 and C_4 are referenced. That means the case is not consistent. However, without referencing these cases it is impossible to prove $f_1 \wedge f_2$ as a decision formula within DB.

The set F is permitted. Since C_{f_1}, C_{f_2} as constructed in the proof of i are consistent. These cases are also not in conflict. In order to prove the absence of a conflict, we have to check that $\mathsf{KB}_W \wedge \mathsf{pres}_{C_1} \wedge \mathsf{pres}_{C_2} \not\models \bot$ and $\mathsf{KB}_W \wedge \mathsf{facts}_{C_1} \wedge \mathsf{facts}_{C_2} \models \bot$. While the first condition is met, the second does not hold, since we need $\mathsf{CaseDesc} = A$ to entail $\bot$.

□

Characterizing Deducibility. Deducibility is the central concept for answering questions that are not explicitly answered by the database. However, Definition 4.2.10 does not give an algorithmic description of how to decide whether some formula is deducible. It is also inconvenient for proving properties about permissibility and deducibility. Thus, we give an equivalent characterization in the following.

Intuitively, a formula should be permissible if there is a set of warranted decisions which allow us to conclude the predicate and a formula f should be deducible if in addition no set of decisions contradicts f. We call the set that lets us conclude f *supporting sets* which we define in the following. Thereafter, we prove that the intuition matches the definitions of permissibility and deducibility.

Definition 4.2.11 (Supporting set)**.** *Let* $\mathsf{DB} = (\mathbf{C}, \leq_t, \mathsf{must\text{-}agree}, \mathsf{may\text{-}ref}, \mu, U)$ *be a consistent case law database,* f *a formula,* $\mathsf{CaseDesc}$ *a case description and* crt *a court. A set* $\mathcal{A}$ *of leaf nodes in* DB *that are labeled with* Assess *is a* supporting set *for formula* f *if the following holds:*

(1) $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \models \bigwedge_{(\mathsf{pre} \to \mathsf{fact}) \in \mathcal{A}} \mathsf{pre}$

(2) $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \wedge \bigwedge_{(\mathsf{pre} \to \mathsf{fact}) \in \mathcal{A}} \mathsf{fact} \models f$

(3) $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \wedge \bigwedge_{(\mathsf{pre} \to \mathsf{fact}) \in \mathcal{A}} \mathsf{fact} \not\models \bot$

A supporting set is unwarranted if it contains an unwarranted node w.r.t. any $C \in \mathbf{C}$*. If it is not unwarranted it is warranted. A supporting set is* consistent *with* DB *if* $\mathsf{DB} \cup \{(\top, \mathsf{CaseDesc}, \mathsf{ProofTree}, \mathsf{crt})\}$ *is consistent, where* $\mathsf{ProofTree}$ *consists of a root node with annotation* $\top$ *and leaf nodes with annotation* $\mathsf{Ref}(C_n)$ *for* $n \in \mathcal{A}$*, where* C_n *is the case that contains node* n*.*

Note that a supporting set that is consistent with the DB leads to consistency, and correct referencing, and does not create any conflicts. The properties required in the definition are a consequence of the definition of database consistency. A case constructed from a supporting set would simply refer to all decisions and place the formula at the root. Case consistency requires the properties (1)-(3) to hold; referential consistency requires that the referenced leaf nodes are warranted and hierarchical consistency requires that the supporting set is not in conflict with DB.

The following theorem characterizes permissibility and deducibility using supporting sets. This characterization suggests an algorithmic way of deciding the properties and gives a tool for proving properties about case law databases.

Theorem 4.2.3. *Let* DB *be a consistent case law database,* f *a formula,* $\mathsf{CaseDesc}$ *a case description and* crt *a court. The following holds:*

1. *For every* $C \in \mathsf{DB}$ *with warranted node* f *there is a supporting set* $\mathcal{A}$ *that supports* f*.*
2. f *is permitted (under circumstance* $\mathsf{CaseDesc}$ *and court* crt*) if and only if there is a supporting set* $\mathcal{A}$ *that supports* f*, is warranted, and is consistent with* DB*.*

3. *f is deducible if and only if there is a supporting set $\mathcal{A}$ that supports f, is consistent with DB, and for every supporting set $\mathcal{B}$ it holds that $\mathcal{B}$ does not support $\neg f$, is unwarranted, or is not consistent with DB.*

Proof. We prove the theorem step by step in the same order the claims are defined.

1. We show a stronger statement for $\mathsf{CaseDesc} = \mathsf{pres}_C$ (since C is consistent it has to hold that $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \models \mathsf{pres}_C$).

   We start with $\mathcal{A}^C$ as the set of all leaf nodes of C that are annotated by Assess and $\mathsf{Ref}(D)$ for some D. For this set all properties (1)–(3) of Definition 4.2.11 clearly hold by consistency of C. However, the set might contain nodes labeled with $\mathsf{Ref}(D)$ which we need to replace in order to fulfill this criterion of the Theorem, as well.

   For a fixed leaf formula $(\mathsf{pre} \rightarrow \mathsf{fact}) \in \mathcal{A}^C$ corresponding to a $\mathsf{Ref}(D)$ leaf node, take the set $\mathcal{A}^D$ for D defined as $\mathcal{A}^C$ for C. By consistency of D, we get (a) and (b) for $\mathsf{CaseDesc}_D = \mathsf{pres}_D$ and $f = \mathsf{fact}$. By referential consistency it holds that $\mathsf{KB}_W \wedge \mathsf{pre} \models \mathsf{pres}_D$. Therefore, if we replace $\mathcal{A}^C$ by $\mathcal{A}^C \backslash \{(\mathsf{pre} \rightarrow \mathsf{fact})\} \cup \mathcal{A}^D$, property (a) holds for C and the new $\mathcal{A}^C$ since $\mathsf{KB}_W \wedge \mathsf{pre} \models \mathsf{pres}_D$. Property (b) also transfers to the new set, since (b) holds for the old set and (b) holds for D and $\mathcal{A}^D$ with respect to $\mathsf{CaseDesc}_D = \mathsf{pres}_D$ and $f = \mathsf{fact}$.

   The process of successively replacing $\mathsf{Ref}(D)$ nodes in $\mathcal{A}^C$ terminates since $\mathcal{A}^D$ only contains $\mathsf{Ref}(E)$ leaf nodes for $E <_{\mathsf{ref}} D$ and DB is finite.

   Our proof above actually shows that $\mathsf{KB}_W \wedge \mathsf{pres}_C \wedge \bigwedge_{(\mathsf{pre} \rightarrow \mathsf{fact}) \in \mathcal{A}} \mathsf{fact} \models \mathsf{facts}_C$, hence (c) follows from consistency of C.

2. The implication $\Rightarrow$ follows from the first part of the proof since permissibility implies that we can add a case as specified. So consider $\Leftarrow$, i.e., let $\mathcal{A}$ be a set supporting for f in circumstances $\mathsf{CaseDesc}$ for a court crt.

   We can construct a case C by referencing all these decisions and putting f in a root node that has all these references as child nodes. The properties (1)–(3) of $\mathcal{A}$ (Definition 4.2.11) imply consistency of C. The requirement that the nodes are warranted and that C is at the end of the timeline implies that we reference correctly.

   The $\mathsf{DB} \cup \{C\}$ is also hierarchically consistent since C does not introduce new conflicts. Otherwise $\mathcal{A}$ would already be in conflict with DB.

3. The implication $\Rightarrow$ follows immediately from the previous part of the proof since f is deducible if f is permitted and $\neg f$ is not permitted. The other implication also follows from the previous part since the existence of $\mathcal{A}$ implies that f is permitted and the non-existence of support for $\neg f$ is implied by the requirement of $\mathcal{B}$.

□

4.2.4 General Properties of Case Law Databases

Introducing a new framework always comes with the risk of modeling errors. A method for alleviating that risk is to prove properties that the framework is expected to have. In order to validate the framework introduced here, we have proven that (i) case references do not influence decisions (Theorem 4.2.3); in this subsection we additionally prove that (ii) consistency is necessary for property (i) (Theorem 4.2.4), and that (iii) neither $\bot$ nor $\{f, \neg f\}$ are ever permitted (Theorem 4.2.5).

Regarding (i), we have shown that every formula f in the database can be derived from a supporting set of previous decisions (Theorem 4.2.3) with the case description and world knowledge. Hence there is no possible interplay between case references that would make it possible to prove something not ultimately backed up by judges' decisions.

Regarding (ii), Theorem 4.2.3 implies immediately that, whenever a formula f is deducible, then it follows (under the circumstances provided with CaseDesc and crt) from decisions made by judges in previous cases. It is easy to verify that our restrictions are necessary to ensure this, i.e., that this property gets lost if we forsake either case-wise or referential consistency:

Theorem 4.2.4. *Let DB be a case law database, and let f be any formula that does not entail $\bot$. Then there exist cases C_1 and C_2, each with root node f and the empty case desc $\top$, such that (inserting C_i at the end of the timeline $\leq_t$):*

- *If DB is case-wise consistent, then so is DB $\cup \{C_1\}$.*
- *If DB is referentially consistent, then so is DB $\cup \{C_2\}$.*
- *If there is a crt such that must-agree(crt) $= \emptyset$, then in addition this holds: for each of $i = 1, 2$, if DB is hierarchically consistent, then so is DB $\cup \{C_i\}$.*

Proof. Let crt be a court with must-agree(crt) $= \emptyset$. For C_1, select an arbitrary $D \in$ DB, and construct ProofTree containing root node f and a single leaf node $(\top \rightarrow f)$ labeled with Ref(D). Define $C_1 := (f, \top,$ ProofTree, crt$)$. Then DB $\cup \{C_1\}$ is case-wise consistent since DB is case-wise consistent (note that we do not enforce referential consistency, so ignore whether or not f is actually decided by D). Hierarchical consistency holds simply because C_1 does not need to reference other cases.

For C_2,construct ProofTree containing the single node f labeled with Axiom. Define $C_2 := (f, \top,$ ProofTree, crt$)$. This case is not consistent; however, DB $\cup \{C_2\}$ is referentially consistent simply because C_2 does not make any references. Hierarchical consistency holds for the same reason as before. □

We remark that, by restricting the formula f only slightly, the proof of Theorem 4.2.4 can be strengthened so as not to have to rely on a maximal court for ensuring hierarchical consistency. In particular, if f is made of predicates that do not occur anywhere in the case law database, then the cases C_1 and C_2 as constructed cannot be in conflict with any other cases, thus preserving hierarchical consistency for arbitrary courts crt. We finally prove (iii), non-permissibility of either $\bot$ or $\{f, \neg f\}$:

Theorem 4.2.5. *The formula $\bot$ is not permitted in any case law database* DB*, under any circumstances* CaseDesc *and court* crt*. The same holds for $\{f, \neg f\}$ if* crt $\in$ must-agree(crt).

Proof. For $\bot$, this holds simply because deducibility requires us to construct a consistent case with root node $\bot$, and any case C one of whose nodes is $\bot$ is not consistent. To see the latter, just note that, if C was consistent, then by Definition 4.2.4 (v) it follows that $\mathsf{facts}_C \models \bot$, which by Definition 4.2.4 (iii) means that C is not consistent.

For $\{f, \neg f\}$, assume to the contrary that there exist cases $C_f = (f, \mathsf{CaseDesc}, \mathsf{ProofTree}_f, \mathsf{crt})$ and $C_{\neg f} = (\neg f, \mathsf{CaseDesc}, \mathsf{ProofTree}_{\neg f}, \mathsf{crt})$ such that $\mathsf{ProofTree}_f$ and $\mathsf{ProofTree}_{\neg f}$ do not contain nodes labeled with Assess, and $\mathsf{DB} \cup \{C_f, C_{\neg f}\}$ is consistent (where the new cases are inserted in any order at the end of the timeline $\leq_t$). But since $\mathsf{crt} \in \mathsf{must\text{-}agree}(\mathsf{crt})$, the latter one has to respect the first one. We show that C_f and $C_{\neg f}$ are in conflict, thus contradicting the hierarchical consistency of $\mathsf{DB} \cup \{C_f, C_{\neg f}\}$. Obviously, $\mathsf{KB}_W \wedge \mathsf{facts}_{C_f} \wedge \mathsf{facts}_{C_{\neg f}} \models f \wedge \neg f \models \bot$. It remains to show that $\mathsf{KB}_W \wedge \mathsf{pres}_{C_f} \wedge \mathsf{pres}_{C_{\neg f}} \not\models \bot$. By consistency of each of C_f and $C_{\neg f}$, we get (a) $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \not\models \bot$, (b) $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \models \mathsf{pres}_{C_f}$ and (c) $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \models \mathsf{pres}_{C_{\neg f}}$. Putting (b) and (c) together[5] gives $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \models \mathsf{pres}_{C_f} \wedge \mathsf{pres}_{C_{\neg f}}$, which with (a) shows $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \wedge \mathsf{pres}_{C_f} \wedge \mathsf{pres}_{C_{\neg f}} \not\models \bot$, which is stronger than what we needed to prove. Therefore, $\{f, \neg f\}$ is not permitted in DB.

□

Note here that the claim for $\bot$ already holds if we require case-wise consistency. The claim for $\{f, \neg f\}$ requires hierarchical consistency as well.

4.2.5 Privacy Cases and Norms

We now point out an interesting property of privacy cases, and of databases consisting only of privacy cases. We call such databases *privacy case law databases*.

Rule based privacy policies are a well established and widely used concept. The rules that are used are usually reflected by norms defining privacy regulations. However, neither rules nor norms are reflected in the case law framework. In this subsection, we show that we can use a natural definition of norms that can be extracted from privacy cases. In addition, it is possible to transform a privacy case to a normal form such that a norm that decides the case is represented. Consequently, we also consider norm extraction as a reasoning task in Section 4.3.

At the core of privacy regulations are positive and negative norms, as introduced by [23]. Positive norms are permissive in the sense that they describe conditions that allow transactions with personal data ($\phi \Rightarrow \mathsf{is\_legal\_action}(a)$). Negative norms, in contrast, define necessary conditions for such transactions, i.e., they forbid transactions with personal data unless certain conditions are met ($\phi \Rightarrow \neg\mathsf{is\_legal\_action}(a)$). We formulate negative norms as conditions that lead to the denial of transactions.

[5] At this point we require the monotonicity of the underlying logic.

Definition 4.2.12 (Norms)**.** *Let $a \in$ Actions. A* norm *is a formula that has the form $\phi \Rightarrow p$ where* is_legal_action(a) *does not occur in ϕ. The norm is a positive norm, denoted ϕ^+, if $p =$* is_legal_action(a) *and a negative norm, denoted ϕ^-, if $p = \neg$*is_legal_action(a)*. A norm ϕ decides p given f if* $\mathsf{KB}_W \wedge f \models \phi$.

In the case law framework, norms are hidden by judges' assessments. However, in the spirit of Theorem 4.2.3, norms are reflected by sets of cases that could be referenced in order to support either the legality of an action (positive norm) or its illegality (negative norm). In the following theorem, we show that we can extract a norm for every privacy case avoiding the recursion of Theorem 4.2.3.

Theorem 4.2.6. *Let* DB *be a consistent privacy case law database and $C =$ (*df, CaseDesc, ProofTree, crt*) $\in$ DB. Then there is a norm ϕ that decides* df *given* CaseDesc*. In particular, there are formulas ϕ_W, ϕ_S such that* is_legal_action(a) *does not occur in these formulas and*

(1) $\mathsf{facts}_C \Rightarrow \phi_W \wedge (\phi_S \Rightarrow \mathsf{df})$

(2) $\phi_W \wedge (\phi_S \Rightarrow \mathsf{df}) \Rightarrow \mathsf{df}$

Proof. We show the statement for $\mathsf{df} = \mathsf{is\_legal\_action}(a)$ for some a. The proof for $\neg\mathsf{is\_legal\_action}(a)$ is analogous. Given consistency of C, we get that $\mathsf{facts}_C \models \mathsf{df}$. Transforming facts_C to a CNF formula, we can write facts_C as $\phi_W \wedge \phi_L$ where $\mathsf{is\_legal\_action}(a)$ only occurs in ϕ_L. Since $\phi_W \wedge \phi_L \models \mathsf{is\_legal\_action}(a)$ we can assume that ϕ_L does not contain $\neg\mathsf{is\_legal\_action}(a)$. Otherwise we could remove the $\neg\mathsf{is\_legal\_action}(a)$ maintaining the properties of $\phi_W \wedge \phi_L \models \mathsf{is\_legal\_action}(a)$ as well as .

Every literal l_j of the formula ϕ_L has the form $\mathsf{is\_legal\_action}(a) \vee \bigvee_{1\leq i\leq k} x_i$, which is equivalent to

$$(\underbrace{\bigwedge_{1\leq i\leq k} \neg x_i}_{=:r_j}) \Rightarrow \mathsf{is\_legal\_action}(a)$$

Hence, we can write ϕ_L as

$$(\bigvee_{1\leq j\leq m} r_j) \Rightarrow \mathsf{is\_legal\_action}(a)$$

We define $\phi_S := \bigvee_{1\leq j\leq m} r_j$ and get

$$\phi_W \wedge (\phi_S \Rightarrow \mathsf{is\_legal\_action}(a)) \models \mathsf{is\_legal\_action}(a)$$

where neither ϕ_W nor ϕ_S contain $\mathsf{is\_legal\_action}(a)$. Therefore, it must hold that $\phi_W \models \phi_S$. However, this argumentation was only applicable in the case C since $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \models \mathsf{pres}_C$. Hence we can derive the norm $\phi^+ := \mathsf{pres}_C \wedge \phi_S$ as positive norm. □

The formulas ϕ_W and ϕ_S can be used to construct a normal form of privacy cases. In particular, this normal form is consistent and allows reading off norms.

Corollary 4.2.7 (Normal forms)**.** *Let* $\mathsf{DB} = (\mathbf{C}, \leq_t, \mathsf{must\text{-}agree}, \mathsf{may\text{-}ref}, \mu, U)$ *be a privacy case law database,* $C = (\mathsf{df}, \mathsf{CaseDesc}, \mathsf{ProofTree}, \mathsf{crt}) \in \mathsf{DB}$ *be a case, and* D *be the set of* C*'s leaf nodes.* $N(C)$ *is the case that consists of a root node* df*, two inner nodes* ϕ_w *and* $\phi_S \Rightarrow \mathsf{df}$ *and the leaf nodes* D *as children of both inner nodes. We call* $N(C)$ *the* normal form *of* C*. If* DB *is consistent, then* $(\mathbf{C}\backslash\{C\} \cup \{N(C)\}, \leq_t)$ *is also consistent (where* $N(C)$ *is placed at the position of* C *w.r.t.* $\leq_t$*).*

Proof. The consistency of $N(C)$ follows from the previous theorem. The leaves of $N(C)$ are the same as the leaves of C, and thus referentially consistency follows from C's referential consistency. In addition, df of $N(C)$, as well as pres_C of $N(C)$, are the same as of C, and thus $N(C)$ is in conflict with a case iff C is. Therefore, hierarchical consistency is also maintained. □

In order to define $N(C)$, we need to duplicate the leaf nodes since the transformations to get ϕ_W and ϕ_S ignore which fact is needed to get the corresponding formula. Thus, a leaf node's fact could end up in both formulas ϕ_W and ϕ_S.

In conformance with [23], we can conclude from deducibility of an action that there is a positive norm supporting it and show that no negative norm can be applied, i.e., all negative norms are respected (Theorem 4.2.5).

4.3 Reasoning Tasks

We now discuss the reasoning tasks associated with our framework — how to answer questions such as "are we allowed to send data D to some party P?" — in more detail, giving an algorithmic sketch and a brief complexity analysis (in terms of the number of reasoning operations required) for each.

Consistency. Analyzing and keeping the state of the case law database consistent is of vital importance for its usefulness; cf. Theorem 4.2.5. As in the definition of consistency, we split the task of checking consistency into case-wise, referential, and hierarchical consistency. The algorithms are straightforward and can be found in Algorithm 1, Algorithm 2 and Algorithm 3 for case consistency, referential consistency, and case-wise hierarchical consistency, respectively.

All of these properties are defined per case, i.e., the case wise check of the corresponding property has to be repeated $|\mathsf{DB}|$ times. Following the respective definition, checking case consistency costs $|\mathsf{ProofTree} + 1|$ entailment operations and checking correct referencing for C costs $\mathsf{references}(C)$ where $\mathsf{references}(C)$ is the number of nodes in C annotated by $\mathsf{Ref}(D)$. Hierarchical consistency can be checked along the time line $\leq_t$ only testing for conflicts with earlier cases. So for the i-th case, we need at most $(i-1) \cdot 2$ entailment checks, since every conflict check requires 2 operations. Consequently, we require $|\mathsf{DB}| \cdot (|\mathsf{DB}| + 1)$ entailment checks.

The property whether the case law database warrants consistently can be checked using one entailment test per reference to a subcase containing an unwarranted decisions node.

Algorithm 1: Case consistency

Input : A case $C = (\mathsf{df}, \mathsf{CaseDesc}, \mathsf{ProofTree}, \mathsf{crt})$

Output : $\top$ if C is consistent and $\bot$ otherwise

1 Check that $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \models \mathsf{pres}_C$.
2 Check that $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \wedge \mathsf{facts}_C \not\models \bot$.
3 For every leaf node n in ProofTree labeled with Axiom, check that $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \models n$.
4 For every inner node n in ProofTree annotated by AND with child nodes $n_1, \ldots, n_k$, check that $\bigwedge_{1 \leq i \leq k} n_i \models n$.
5 For every inner node n in ProofTree annotated by OR with child nodes $n_1, \ldots, n_k$, check that $\bigvee_{1 \leq i \leq k} n_i \models n$.
6 If all checks succeed output $\top$; otherwise output $\bot$.

Algorithm 2: Referential consistency

Input : A case $C = (\mathsf{df}, \mathsf{CaseDesc}, \mathsf{ProofTree}, \mathsf{crt})$ and a case law database DB

Output : $\top$ if C is referentially consistent w.r.t. DB and $\bot$ otherwise

1 **for** *every subcase D referenced by leaf node* $\mathsf{pre} \rightarrow \mathsf{fact}$ **do**
2 | check that $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \wedge \mathsf{pre} \models \mathsf{pres}_D$
3 **end**
4 If all checks succeed output $\top$; otherwise output $\bot$.

Deducibility and Permissibility. As deducibility amounts to two consecutive permissibility checks, we consider the latter exclusively. We are given a database DB, a formula whose permissibility should be checked, as well as a case description CaseDesc and a court crt forming the circumstances.By Theorem 4.2.3, permissibility is equivalent to the existence of a supporting set $\mathcal{A}$ for f that is consistent with the database. Thus the task of permissibility, i.e.,

Algorithm 3: Case-wise hierarchical consistency

Input : A case $C = (\mathsf{df}, \mathsf{CaseDesc}, \mathsf{ProofTree}, \mathsf{crt})$ and a hierarchically consistent CLD DB

Output : $\top$ if $\mathsf{DB} \cup \{C\}$ is hierarchically consistent (where C is set to be the maximum w.r.t. $\leq_t$)

1 **for** *every $D \in$ DB for which* $\mathsf{crt} <_{\S} \mathsf{crt}_D$ **do**
2 | check that $\mathsf{KB}_W \wedge \mathsf{pres}_C \wedge \mathsf{pres}_D \not\models \bot$
3 | check that $\mathsf{KB}_W \wedge \mathsf{df} \wedge \mathsf{df}_D \models \bot$.
4 | If both checks succeed output $\bot$.
5 **end**
6 Output $\top$.

giving a "yes" vs. "no" answer, can be reduced to checking the existence of a suitable set $\mathcal{A}$. If the answer is "yes", we can also output a witness, i.e., a hypothetical case C showing permissibility. A straightforward means for doing this is to set $C := (f, \mathsf{CaseDesc}, \mathsf{ProofTree}, \mathsf{crt})$ where $\mathsf{ProofTree}$ consists of root node f, one leaf node l labeled with $\mathsf{Ref}(D)$ for every $D \in \mathcal{A}$, as well as one leaf node $\mathsf{KB}_W \wedge \mathsf{CaseDesc}$ labeled with Axiom. For convenience, we will denote this construction by $C(\mathcal{A})$. See Algorithm 4.

Algorithm 4: Permissibility

Input : A formula f, case description $\mathsf{CaseDesc}$, court crt, and a consistent CLD DB

Output : A case $C = (f, \mathsf{CaseDesc}, \mathsf{ProofTree}, \mathsf{crt})$ such that $\mathsf{DB} \cup \{C\}$ is consistent (where C is set to be the maximum w.r.t. $\leq_t$), or $\bot$ if no such C exists

1 Test whether $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \models \bot$. If so, output $\bot$.
2 Test whether $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \models f$. If so, output $(f, \mathsf{CaseDesc}, \mathsf{ProofTree}, \mathsf{crt})$ where $\mathsf{ProofTree}$ is the proof tree consisting of a leaf node labeled by Axiom containing f.
3 Set $\mathcal{N} := \emptyset$.
4 **for** *every* $D \in \mathsf{DB}$ *and every* $(\mathsf{pre} \rightarrow \mathsf{fact}) \in D$ *labeled* Assess **do**
5 Check if $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \models \mathsf{pre}$
6 Check if $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \wedge \mathsf{fact} \not\models \bot$
7 If both checks succeed, set $\mathcal{N} := \mathcal{N} \cup \{(\mathsf{pre} \rightarrow \mathsf{fact})\}$.
8 **end**
9 **for** $\mathcal{A} \in 2^{\mathcal{N}}$ **do**
10 Check that $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \models \bigwedge_{(\mathsf{pre} \rightarrow \mathsf{fact}) \in \mathcal{A}} \mathsf{pre}$
11 Check that $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \wedge \bigwedge_{(\mathsf{pre} \rightarrow \mathsf{fact}) \in \mathcal{A}} \mathsf{fact} \models f$
12 Check that $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \wedge \bigwedge_{(\mathsf{pre} \rightarrow \mathsf{fact}) \in \mathcal{A}} \mathsf{fact} \not\models \bot$
13 **for** *every* $E \in \mathsf{DB}$ *with* $\mathsf{crt} <_{\S} \mathsf{crt}_E$ **do**
14 Check that E and $C(\mathcal{A})$ are not in conflict (cf. Algorithm 3).
15 **end**
16 If all three tests succeed, go on with step 18, otherwise continue with the next $\mathcal{D}$.
17 **end**
18 If a set $\mathcal{A}$ succeeded, output $C(\mathcal{A})$, otherwise output $\bot$.

The correctness of the algorithm is shown by Theorem 4.2.3; lines 10-12 check that the set supports f and lines 13-15 ensure that it is consistent with the database.

In contrast to our previous algorithms, deducibility checking as per Algorithm 4 requires an exponential number of entailment checks in the worst case (a trivial bound is in the order of 2^N where N is the number of decision nodes in the database). This raises the questions (1) whether or not this exponential overhead is inherent in the complexity of deciding permissibility, and (2) whether it is possible to encode the permissibility test directly into the logic instead. In what follows, we shed some light on (1) and (2).

The answer to (1) is a qualified "yes" in the sense that permissibility checking essentially

pre-fixes entailment checks with an existential quantifier. As entailment checks correspond to universal quantification, this intuitively means that for permissibility we need to test the validity of a $\exists\forall$ formula, instead of a $\forall$ formula for entailment. So, we add a quantifier alternation step, which typically does come at the price of increased complexity. This line of thought also immediately provides an intuitive answer to question (2), namely "yes but only if the underlying logic contains $\exists\forall$ quantification".

Of course, both these answers are only approximate and only speak in broad terms. Whether each is to be answered with "yes" or "no" depends on the precise form of the logic, and on what kind of blow-up we are willing to tolerate. To make matters concrete, we now consider three particular logics, namely first-order predicate logic and propositional logic (i.e., first-order predicate logic given a finite universe and without quantification). We start with the latter.

In what follows, say we need to check whether formula f is permitted in DB under circumstances $\mathsf{CaseDesc}$. We abstract from the complications entailed by maintaining hierarchical consistency, and assume that for crt, it holds that $\mathsf{must\text{-}agree}(\mathsf{crt}) = \emptyset$.

Theorem 4.3.1. *For propositional logic, deciding permissibility is Σ_2^p-complete.*

For an easier understanding, we first give a short sketch of the proof, before we go into details.

Proof sketch. The set $\Sigma_2^p = \mathbf{NP}^{\mathbf{NP}}$, so containment is shown by guessing a supporting set and verifying its properties using an $\mathbf{NP}$ oracle. For the hardness we encode an QBF formula $\exists x \forall y : \phi(x, y)$ in permissibility request for case law database. We do this by encoding all possible values for x in the database and asking for the permissibility of $\phi(x, y)$.

Proof. Recall that $\Sigma_2^p = \mathbf{NP}^{\mathbf{NP}}$. Membership follows because we can guess the set $\mathcal{A}$ and check, using an $\mathbf{NP}$ oracle, the three entailment tests (1–3). The consistency of the set with DB can also be answered by the $\mathbf{NP}$ oracle since verifying a conflict can be done in polynomial time.

For hardness, consider a QBF formula of the form $\exists X \forall Y \phi(X, Y)$ where each of X and Y are variable sets and $\phi(X, Y)$ is an arbitrary propositional formula in the variables $X \cup Y$. Testing validity of $\exists X \forall Y \phi(X, Y)$ is Σ_2^p-hard. To polynomially reduce this to permissibility testing over a propositional logic, we construct a corresponding case law database DB as follows. For each $x \in X$, DB includes a case $(x, \top, \mathsf{ProofTree}, \mathsf{crt})$ where $\mathsf{ProofTree}$ consists of a single Assess node of the form $\top \rightarrow x$, as well as a case $(\neg x, \top, \mathsf{ProofTree}, \mathsf{crt})$ where $\mathsf{ProofTree}$ consists of a single Assess node of the form $\top \rightarrow \neg x$. In other words, for each x we have both truth-value decisions available for $\mathcal{A}$ to choose from. We set $f := \phi(X, Y)$. Obviously, this reduction is polynomial in the size of the formula $\exists X \forall Y \phi(X, Y)$. To see that the reduction is correct, observe that f is permitted in DB iff there exists a truth assignment a to X which, viewed as a conjunction of literals, entails $\phi(X, Y)$, i.e., $a \models \phi(X, Y)$. The latter is the case iff there exists a s.t., for all truth assignments to Y, $\phi(a(X), Y)$ is true (where $\phi(a(X), Y)$ instantiates each $x \in X$ with $a(x)$). This, finally, is the case iff $\exists X \forall Y \phi(X, Y)$ is valid, which is what we needed to show. □

As entailment testing in propositional logic is only **coNP**-complete, Theorem 4.3.1 answers question (1) with "yes", and answers question (2) with "no, unless we are willing to tolerate worst-case exponentially large formulas". Unsurprisingly, the answers for first-order logic are different:

Theorem 4.3.2. *Permissibility is equivalent to validity of a formula whose size is polynomial in the size of* DB, CaseDesc, *and* f *for first-order logic.*

We use existential quantification in order to choose a warranted supporting set and then design the formula such that it is valid if and only if the consistency properties of the case holds that can be constructed from that supporting set (i.e., the case potentially output by Algorithm 4). All parts that are not chosen by the existential quantifier will be equivalent to $\top$.

Proof. Let $\mathcal{L} = \{n_1 = (\mathsf{pre}_1 \rightarrow \mathsf{fact}_1), \ldots, n_k = (\mathsf{pre}_k \rightarrow \mathsf{fact}_k)\}$ be the set of all warranted leaf formulas of cases $C' \in \mathsf{DB}$ with label Assess. We need to construct a first-order formula ϕ that is valid iff there exists $\mathcal{A} \subseteq \mathcal{L}$ such that the three implications (1–3) of Definition 4.2.11 hold. Our idea is to encode the choice of that subset as an "on/off switch" associated with each n_i. The switch will be realized through an existential quantifier over $x_1, \ldots, x_k$ and a unary predicate chosen_i for every $i \in \{1, \ldots, k\}$ which we add to the FOL signature (w.l.o.g. all chosen_i do not occur in any pre_i or fact_i). The meaning of the predicate is that $\mathsf{chosen}_i(x_i)$ holds if and only if n_i is chosen for the set $\mathcal{A}$. In order to ensure that every decision node can be selected using chosen_i we add $\phi_i^{\mathsf{chosen}} := (\exists x : \mathsf{chosen}_i(x)) \wedge (\exists x : \neg\mathsf{chosen}_i(x))$ and define $\phi^{\mathsf{chosen}} := \bigwedge_i \phi_i^{\mathsf{chosen}}$.

We next define the formulas $\phi_i^{\mathsf{preSwitch}} := (\neg\mathsf{chosen}_i(x_i) \vee \mathsf{pre}_i)$ and $\phi_i^{\mathsf{factSwitch}} := (\neg\mathsf{chosen}_i(x_i) \vee \mathsf{fact}_i)$ to implement our switches. Note that, if for x_i it holds that $\neg\mathsf{chosen}_i(x_i)$, then both $\phi_i^{\mathsf{preSwitch}}$ and $\phi_i^{\mathsf{factSwitch}}$ simplify to $\top$; if for x_i it holds that $\mathsf{chosen}_i(x_i)$, then $\phi_i^{\mathsf{preSwitch}}$ simplifies to pre_i and $\phi_i^{\mathsf{factSwitch}}$ simplifies to fact_i. Using these building blocks, we define our correspondences to the implications (1–3), as follows:

- $\phi^{(1)} := \mathsf{KB}_W \wedge \mathsf{CaseDesc} \Rightarrow \bigwedge_{i=1}^{k} \phi_i^{\mathsf{preSwitch}}$.
- $\phi^{(2)} := \mathsf{KB}_W \wedge \mathsf{CaseDesc} \wedge \bigwedge_{i=1}^{k} \phi_i^{\mathsf{factSwitch}} \Rightarrow f$.
- $\phi^{(3)} := \neg(\mathsf{KB}_W \wedge \mathsf{CaseDesc} \wedge \bigwedge_{i=1}^{k} \phi_i^{\mathsf{factSwitch}} \Rightarrow \bot)$.

Our formula ϕ then is defined simply as $\phi := \phi^{\mathsf{chosen}} \Rightarrow \exists \vec{x}.\phi^{(1)} \wedge \phi^{(2)} \wedge \phi^{(3)}$ where $\vec{x} = x_1, \ldots, x_k$. We now prove that ϕ is valid iff there exists $\mathcal{A} \subseteq \mathcal{L}$ such that the three implications (1–3) hold.

"$\Leftarrow$": Assume there is a set $\mathcal{A}$ such that the implications (1–3) hold. Let I be an arbitrary interpretation. We define an assignment a for the x_i as follows: if $(\mathsf{pre}_i \rightarrow \mathsf{fact}_i) \in \mathcal{A}$, then $\mathsf{chosen}_i(x_i) \equiv \top$ and otherwise $\mathsf{chosen}_i(x_i) \equiv \bot$. If we cannot choose x_i that way, it holds that $I \not\models \phi^{\mathsf{chosen}}$, i.e., $I \models \phi$.

Using this choice of the x_i, the formula $\bigwedge_{i=1}^{k} \phi_i^{\mathsf{preSwitch}}$ reduces to $\bigwedge_{(\mathsf{pre}\rightarrow\mathsf{fact})\in\mathcal{A}} \mathsf{pre}$ and $\bigwedge_{i=1}^{k} \phi_i^{\mathsf{factSwitch}}$ reduces to $\bigwedge_{(\mathsf{pre}\rightarrow\mathsf{fact})\in\mathcal{A}} \mathsf{fact}$. Thus, (1) implies $I, a \models \phi^{(1)}$, (2) implies

$I, a \models \phi^{(2)}$ and (3) implies $I, a \models \phi^{(3)}$. Since I was an arbitrary interpretation, it follows that ϕ is valid.

"⇒": For this, we show the contraposition, i.e., assume there is no such set $\mathcal{A}$, we show the formula ϕ is not valid. The latter is equivalent to $\neg\phi$ is satisfiable where $\neg\phi = \phi^{\mathsf{chosen}} \wedge \forall\vec{x}.\neg\phi^{(1)} \vee \neg\phi^{(2)} \vee \neg\phi^{(3)}$. There is no $\mathcal{A}$ means for every $\mathcal{A}$ there is one of (1–3) not satisfied. We construct a I such that $I \models \neg\phi$. We can assume $I \models \phi^{\mathsf{chosen}}$ (we can modify every interpretation s.t. the value on formulas not containing chosen_i for some i does not change).

For every set $\mathcal{A}$ the corresponding choice $\vec{x}$ allows an interpretation that does not satisfy $\phi^{(1)}$, $\phi^{(2)}$, or $\phi^{(3)}$. Thus, there is an interpretation for $\neg\phi$.

□

Theorem 4.3.3. *Permissibility is equivalent to validity of a formula whose size is polynomial in the size of DB, CaseDesc, and f for first-order logic.*

Proof. Permissibility is equivalent to the existence of a supporting set (Theorem 4.2.3). So, let $\mathcal{L} = \{n_1 = (\mathsf{pre}_1 \to \mathsf{fact}_1), \ldots, n_k = (\mathsf{pre}_k \to \mathsf{fact}_k)\}$ be the set of all warranted leaf formulas of cases $C' \in \mathsf{DB}$ with label Assess. We need to construct a first-order formula ϕ that is valid iff there exists $\mathcal{A} \subseteq \mathcal{L}$ such that the three implications (1–3) of Definition 4.2.11 hold:

(1) $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \models \bigwedge_{(\mathsf{pre}\to\mathsf{fact})\in\mathcal{A}} \mathsf{pre}$

(2) $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \wedge \bigwedge_{(\mathsf{pre}\to\mathsf{fact})\in\mathcal{A}} \mathsf{fact} \models f$

(3) $\mathsf{KB}_W \wedge \mathsf{CaseDesc} \wedge \bigwedge_{(\mathsf{pre}\to\mathsf{fact})\in\mathcal{A}} \mathsf{fact} \not\models \bot$

There are two main problems to solve: first, the formula has to allow to select an arbitrary subset of the nodes. The second problem is that the first two properties formulate the validity of

$$\phi_{\mathsf{val}} := \mathsf{KB}_W \wedge \mathsf{CaseDesc} \Rightarrow \left(\bigwedge_{(\mathsf{pre}\to\mathsf{fact})\in\mathcal{A}} \mathsf{pre} \wedge \left(\bigwedge_{(\mathsf{pre}\to\mathsf{fact})\in\mathcal{A}} \mathsf{fact} \Rightarrow f \right) \right)$$

whereas the last requirement corresponds to the satisfiability of the corresponding $\phi_{\mathsf{sat}} := \mathsf{KB}_W \wedge \mathsf{CaseDesc} \wedge \bigwedge_{(\mathsf{pre}\to\mathsf{fact})\in\mathcal{A}} \mathsf{fact}$. The formula f is permitted if ϕ_{val} is valid and ϕ_{sat} is satisfiable.

In the following we first show how to (i) decide permissibility given a formula $\phi^{(3)}$ that is valid iff ϕ_{sat} is satisfiable and then we show that (ii) there is formula $\phi^{(3)}$ that is valid iff ϕ_{sat} is satisfiable.

(i) **Decision node set selection.**

We solve the first problem by encoding the choice of that subset as an "on/off switch" associated with each n_i. The switch will be realized through an existential quantifier over $x_1, \ldots, x_k$ and a unary predicate chosen_i for every $i \in \{1, \ldots, k\}$ which we add to the FOL signature (w.l.o.g. all chosen_i do not occur in any pre_i or fact_i). The

meaning of the predicate is that $\mathsf{chosen}_i(x_i)$ holds if and only if n_i is chosen for the set $\mathcal{A}$. In order to ensure that every decision node can be selected using chosen_i we add $\phi_i^{\mathsf{chosen}} := (\exists x : \mathsf{chosen}_i(x)) \wedge (\exists x : \neg\mathsf{chosen}_i(x))$ and define $\phi^{\mathsf{chosen}} := \bigwedge_i \phi_i^{\mathsf{chosen}}$. We next define the formulas $\phi_i^{\mathsf{preSwitch}} := (\neg\mathsf{chosen}_i(x_i) \vee \mathsf{pre}_i)$ and $\phi_i^{\mathsf{factSwitch}} := (\neg\mathsf{chosen}_i(x_i) \vee \mathsf{fact}_i)$ to implement our switches. Note that, if for x_i it holds that $\neg\mathsf{chosen}_i(x_i)$, then both $\phi_i^{\mathsf{preSwitch}}$ and $\phi_i^{\mathsf{factSwitch}}$ simplify to $\top$; if for x_i it holds that $\mathsf{chosen}_i(x_i)$, then $\phi_i^{\mathsf{preSwitch}}$ simplifies to pre_i and $\phi_i^{\mathsf{factSwitch}}$ simplifies to fact_i. Using these building blocks, we define our correspondences to the implications (1–3), as follows:

- $\phi^{(1)} := \mathsf{KB}_W \wedge \mathsf{CaseDesc} \Rightarrow \bigwedge_{i=1}^{k} \phi_i^{\mathsf{preSwitch}}$.
- $\phi^{(2)} := \mathsf{KB}_W \wedge \mathsf{CaseDesc} \wedge \bigwedge_{i=1}^{k} \phi_i^{\mathsf{factSwitch}} \Rightarrow f$.

The formula ϕ then is defined simply as $\phi := \phi^{\mathsf{chosen}} \Rightarrow \exists\vec{x}.\phi^{(1)} \wedge \phi^{(2)} \wedge \phi^{(3)}$ where $\vec{x} = x_1, \ldots, x_k$. We now prove that ϕ is valid iff there exists $\mathcal{A} \subseteq \mathcal{L}$ such that the three implications (1–3) hold.

(ii) **Satisfiability to validity for ϕ_{sat}**

While it is easy to formulate the satisfiability of ϕ_{sat} as second order validity by existentially quantifying the predicates, it is also possible to a construction similar to the chosen_i in order to find a formula $\phi^{(3)}$ that is valid if and only if ϕ_{sat} is satisfiable.

We introduce one predicate chosen_P which has the arity equal to the maximal arity occurring in ϕ_{sat} plus 1. We want to model every predicate $p(\vec{x})$ in ϕ_{sat} by $\mathsf{chosen}_P(p, \vec{x})$. Now, we need to ensure that every possible predicate can be chosen, which is more difficult than simply selecting one point. In particular, we cannot ensure that every possible subset of the domain variables can be selected by p, however, we can ensure that every subset that is specified by a first-order formula can be selected which is sufficient for going to a first-order formula for ϕ_{sat} that is valid iff ϕ_{sat} is satisfiable.

$$\begin{aligned}\phi_p := {} & (\forall\vec{x}.\exists p.\mathsf{chosen}_P(p,\vec{x})) \wedge (\forall p.\exists q.\forall x.\mathsf{chosen}_p(p,\vec{x}) \Leftrightarrow \neg\mathsf{chosen}_P(q,\vec{x})) \\ & \wedge (\forall p,q.\exists r.\forall\vec{x}.\mathsf{chosen}_P(r,\vec{x}) \Leftrightarrow (\mathsf{chosen}_P(p,\vec{x}) \wedge \mathsf{chosen}_P(q,\vec{x}))) \\ & \wedge \bigwedge_{1\le i\le n} (\forall p.\exists q.\forall\vec{x} = (x_1,\ldots,x_n).\mathsf{chosen}_p(q,\vec{x}) \\ & \qquad\qquad \Leftrightarrow (\forall x_i'\mathsf{chosen}_P(p,(x_1,\ldots,x_{i-1},x_i',x_{i+1},\ldots,x_n))\end{aligned}$$

Combining negation, conjunction and for all quantification we can represent every formula. Using this requirement formulation, we get that ϕ_{sat} is satisfiable if and only if $\phi_p \Rightarrow \exists\vec{p}.\phi_{\mathsf{sat}}[\mathsf{chosen}_P(p,\vec{x})/p(\vec{x})]$ is valid. We denote $\phi_{\mathsf{sat}}[\mathsf{chosen}_P(p,\vec{x})/p(\vec{x})]$ as $\phi^{(3)}$.

In total this leads to the formula

$$\phi := (\phi^{\mathsf{chosen}} \wedge \phi_p) \Rightarrow (\phi^{(1)} \wedge \phi^{(2)} \wedge \phi^{(3)})$$

which is valid if and only if f is permitted.

□

Norm extraction. As seen in Section 4.2.5, privacy cases induce normative rules. The format of rules gives the advantage that these are easy to enforce and bridge the gap towards privacy policies. As shown by Theorem 4.2.6 we extract a norm for every case in the database. The assumption is that the case is consistent with respect to an underlying consistent privacy case law database DB. The algorithm (Algorithm 5) basically turns the proof of Theorem 4.2.6 into an algorithm transforming the logical formula of the case's facts.

Algorithm 5: Norm extraction

Input : A case $C = (\mathsf{df}, \mathsf{CaseDesc}, \mathsf{ProofTree}, \mathsf{crt})$
Output : A norm ϕ that decides df

```
1 phi := ⊤
2 for leaf node n in C do
3 |   φ := φ∧CNF(n)
4 end
5 Remove ¬df from φ
6 Remove all clauses not containing df
7 Remove df from φ
8 φ := pre_C ∧ ¬φ
9 Output φ
```

Let f be the size of the biggest formula in the leaves of C and n the number of nodes in C. Then the size of the norm can become $\mathcal{O}(2^f \cdot n + |\mathsf{pre}_C|)$. The computation needs operations linear in that size. However, there is no need for any operations to decide $\models$ in order to solve this reasoning task.

4.4 Logic Selection

For modeling purposes — naturally modeling the background knowledge base, the detailed aspects characterizing a case description, and the reasoning applied in arguments — as well as for computational purposes — effectively realizing the desired reasoning tasks — the choice of logic is, of course, of paramount importance. The only hard requirement ("must have") that the logic, $\mathcal{L}$, must meet is:

(i) **Sufficient expressivity** to tackle our framework and reasoning tasks. Precisely, the minimal requirement is for $\mathcal{L}$ to provide a language $\mathcal{L}_{\mathcal{F}}$ for formulas, with reasoning support for tests of the form (a) $\bigwedge_{\phi\in\Phi} \models \bot$ and (b) $\bigwedge_{\phi\in\Phi} \models \psi$: These are the only tests our reasoning tasks demand from the underlying logic. If $\mathcal{L}_{\mathcal{F}}$ is closed under conjunction and contains $\bot$ (as will be the case in our logic of choice), the requirement simply becomes to be able to test whether $\phi \models \psi$.

The soft requirements ("nice to have") on the logic are:

(ii) **Suitable for modeling real-world phenomena and knowledge**, ideally an established paradigm for such modeling tasks.

(iii) **Decidability, and as low complexity as possible**, of the relevant reasoning (e.g., satisfiability checks; cf. (i)).

(iv) **Effective tool support** established and available.

What we have just outlined is essentially a "wanted poster" for *description logic (DL)* [9]. This is a very well investigated family of fragments of first-order logic (several decades of research in AI and related areas), whose mission statement is to provide a language for modeling real-world phenomena and knowledge (ii), while retaining decidability and exploring the trade-off of expressivity vs. complexity (iii). Effective tool support (iv) has been an active area for two decades. Every DL provides a language to describe "axioms", and even the most restricted DLs (in particular, the DL-Lite family [44] which constitutes the "lower extreme" of the DL complexity scale) make it possible to answer queries about the truth of an axiom relative to a conjunction of axioms, which is exactly the test we require.

To make things concrete, we briefly consider the description logic *attributive concept language with complements*, for short $\mathcal{ALC}$, which was introduced in 1991 [110],[6] and is widely regarded as the canonical "basic" description logic variant (most other DLs extend $\mathcal{ALC}$, in a variety of directions). Description logic is a form of predicate logic that considers only 1-ary and 2-ary predicates, referred to as *concepts* and *roles*, respectively. Assuming a set N_C of concept names and a set N_R of role names, DL makes it possible to construct *complex concepts*, which correspond to a particular subset of predicate-logic formulas with exactly one free variable. For $\mathcal{ALC}$, the set of complex concepts is the smallest set such that

1. $\top, \bot$ and every concept name $A \in N_C$ are complex concepts, and
2. if C and D are complex concepts and $r \in N_R$, then $C \sqcap D$, $C \sqcup D$, $\neg C$, $\forall r.C$, and $\exists r.C$ are complex concepts.

Here, $\sqcap$ denotes concept intersection (logical conjunction), $\sqcup$ denotes concept union (logical disjunction), and $\neg C$ denotes concept complement (logical negation). $\forall r.C$ collects the set of all objects x such that, whenever x stands in relation r to y, $y \in C$. Similarly, $\exists r.C$ collects the set of all objects x such that there exists y where x stands in relation r to y and $y \in C$.

$\mathcal{ALC}$ allows *concept inclusion* axioms, of the form $C \sqsubseteq D$, where C, D are complex concepts, meaning that C is a subset of D (universally quantified logical implication). $\mathcal{ALC}$ furthermore allows *assertional* axioms, of the form $x : C$ or $(x, y) : r$, where C is a complex concept, r is a role, and x and y are individual names (i.e., constants). An $\mathcal{ALC}$ knowledge base consists of finite sets of concept inclusion axioms and assertional axioms (called the *TBox* and *ABox* respectively), interpreted as conjunctions. The basic reasoning services provided by $\mathcal{ALC}$ (and most other DLs) are testing whether a knowledge base KB is satisfiable, and testing whether $\mathsf{KB} \models \phi$ where ϕ is an axiom. These decision problems are decidable, and more

[6] For a comprehensive overview of current techniques and results regarding $\mathcal{ALC}$, see [10].

precisely, ExpTime-complete for $\mathcal{ALC}$. (In some DL-Lite variants, the decision problems are in **NP**, or even polynomial-time solvable.)

For our purposes, we can assume as our formulas $\mathcal{L}_{\mathcal{F}}$ conjunctions of axioms, i.e., the smallest set that contains $\bot$, all axioms of the underlying DL (e.g., $\mathcal{ALC}$), as well as $\phi \wedge \psi$ if ϕ and ψ are members of $\mathcal{L}_{\mathcal{F}}$. In order to test whether $\phi \models \psi$, we then simply call the DL reasoning service "$\phi \models \psi_i$?" for every conjunct ψ_i of ψ and return "yes" iff all these calls did. In other words, we may use conjunctions of DL axioms in the knowledge base, case descriptions, and proof tree nodes.

4.5 Concluding PriCL

In this paper, we introduced PriCL, the first framework for automated reasoning about case law. We showed that it complies with natural requirements of consistency and tailored the framework for privacy case law. Moreover, we showed a tight connection between privacy case law and the notion of norms that underlies existing rule-based privacy research. We identified the major reasoning tasks such as checking the case law database for consistency, extracting norms and deducing whether an action is legal or not. For all these tasks, we gave algorithms deciding them and we did an analysis that leads to $\mathcal{ALC}$ as a suitable instantiation for the logic. In particular, $\mathcal{ALC}$ provides efficient realizations while being sufficiently expressive and suitable for modeling real-world phenomena and knowledge.

For future research, we need to construct a significantly large data base consisting of real world cases. Here, the challenge is to differentiate between statements made as world knowledge statement, those made because of the case descriptions and those referenced. The reason for this is that there is no clean language-wise separation in the argumentation.

Bibliography

[1] M. Abadi and C. Fournet. “Mobile values, new names, and secure communication”. In: *POPL ’01: Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages*. London, United Kingdom: ACM Press, 2001, pp. 104–115. ISBN: 1-58113-336-7. DOI: http://doi.acm.org/10.1145/360204.360213.

[2] M. Abadi and A. D. Gordon. “A Calculus for Cryptographic Protocols: The Spi Calculus”. In: *Proc. 4th ACM Conference on Computer and Communications Security*. 1997, pp. 36–47.

[3] M. Abadi and P. Rogaway. “Reconciling two views of cryptography (The computational soundness of formal encryption)”. In: *Journal of Cryptology* 15.2 (2002), pp. 103–127.

[4] A. Anderson. “A comparison of two privacy policy languages: EPAL and XACML”. In: (2005).

[5] G. J. Annas. “HIPAA regulations-a new era of medical-record privacy?” In: *New England Journal of Medicine* 348.15 (2003), pp. 1486–1490.

[6] A. I. Antón, E. Bertino, N. Li, and T. Yu. “A roadmap for comprehensive online privacy policy management”. In: *Communications of the ACM* 50.7 (2007), pp. 109–116.

[7] P. Ashley. “Enforcement of a P3P Privacy Policy.” In: *AISM*. Citeseer. 2004, pp. 11–26.

[8] P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter. “Enterprise privacy authorization language (EPAL 1.2)”. In: *Submission to W3C* (2003).

[9] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, eds. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[10] F. Baader, I. Horrocks, and U. Sattler. “Description Logics”. In: *Handbook of Knowledge Representation*. Elsevier, 2008. Chap. 3, pp. 135–180.

[11] M. Backes, A. Datta, and A. Kate. “Asynchronous Computational VSS with Reduced Communication Complexity”. In: *Proc. of CT-RSA’13*. 2013, pp. 259–276.

[12] M. Backes, A. Kate, and A. Patra. “Computational Verifiable Secret Sharing Revisited”. In: *Proc. of ASIACRYPT’11*. 2011, pp. 590–609.

[13] M. Backes, F. Bendun, M. Maffei, E. Mohammadi, and K. Pecina. “Symbolic Malleable Zero-Knowledge Proofs”. In: *IEEE 28th Computer Security Foundations Symposium, CSF 2015, Verona, Italy, 13-17 July, 2015*. 2015, pp. 412–426. DOI: 10.1109/CSF.2015.35. URL: https://doi.org/10.1109/CSF.2015.35.

[14] M. Backes, D. Hofheinz, and D. Unruh. “CoSP: A general framework for computational soundness proofs”. In: *ACM CCS 2009*. 2009, pp. 66–78.

[15] M. Backes, C. Hriţcu, and M. Maffei. “Type-checking Zero-knowledge”. In: *15th ACM Conference on Computer and Communications Security (CCS 2008)*. ACM Press, 2008, pp. 357–370.

[16] M. Backes, G. Karjoth, W. Bagga, and M. Schunter. “Efficient comparison of enterprise privacy policies”. In: *Proc. of Symposium on Applied Computing*. ACM. 2004, pp. 375–382.

[17] M. Backes, S. Lorenz, M. Maffei, and K. Pecina. “Anonymous webs of trust”. In: *Proceedings of the 10th international conference on Privacy enhancing technologies*. PETS’10. Berlin, Germany: Springer-Verlag, 2010, pp. 130–148. ISBN: 3-642-14526-4, 978-3-642-14526-1. URL: http://dl.acm.org/citation.cfm?id=1881151.1881159.

[18] M. Backes, M. Maffei, and E. Mohammadi. “Computationally Sound Abstraction and Verification of Secure Multi-Party Computations”. In: *FSTTCS*. Ed. by K. Lodaya and M. Mahajan. Vol. 8. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010, pp. 352–363. ISBN: 978-3-939897-23-1.

[19] M. Backes, M. Maffei, and D. Unruh. “Zero-Knowledge in the Applied Pi-calculus and Automated Verification of the Direct Anonymous Attestation Protocol”. In: *IEEE Symposium on Security and Privacy 2008*. May 2008, pp. 158–169.

[20] M. Backes and B. Pfitzmann. “Symmetric Encryption in a Simulatable Dolev-Yao Style Cryptographic Library”. In: *Proc. 17th IEEE Computer Security Foundations Workshop (CSFW)*. 2004, pp. 204–218.

[21] M. Backes and D. Unruh. “Computational Soundness of Symbolic Zero-Knowledge Proofs”. In: *Journal of Computer Security* 18.6 (2010). Preprint on IACR ePrint 2008/152, pp. 1077–1155.

[22] M. Backes and D. Unruh. “Computational Soundness of Symbolic Zero-Knowledge Proofs Against Active Attackers”. In: *21st IEEE Computer Security Foundations Symposium, CSF 2008*. 2008.

[23] A. Barth, A. Datta, J. C. Mitchell, and H. Nissenbaum. “Privacy and contextual integrity: Framework and applications”. In: *Proc. of S&P*. IEEE. 2006, 15–pp.

[24] A. Barth, J. C. Mitchell, A. Datta, and S. Sundaram. “Privacy and Utility in Business Processes.” In: *CSF* 7 (2007), pp. 279–294.

[25] D. Basin, F. Klaedtke, S. Marinovic, and E. Zălinescu. "Monitoring compliance policies over incomplete and disagreeing logs". In: *Proc. of Runtime Verification*. Springer. 2013, pp. 151–167.

[26] D. Basin, S. Mödersheim, and L. Viganò. "OFMC: A symbolic model checker for security protocols". In: *International Journal of Information Security* (2004).

[27] D. A. Basin, F. Klaedtke, S. Müller, and B. Pfitzmann. "Runtime Monitoring of Metric First-order Temporal Properties". In: *Proc. of FSTTCS*. 2008, pp. 49–60.

[28] D. Beaver. "Efficient Multiparty Protocols Using Circuit Randomization". In: *Proc. of CRYPTO'91*. 1991, pp. 420–432.

[29] Z. Beerliová-Trubíniová and M. Hirt. "Efficient Multi-party Computation with Dispute Control". In: *Proc. of TCC'06*. 2006, pp. 305–328.

[30] Z. Beerliová-Trubíniová and M. Hirt. "Perfectly-Secure MPC with Linear Communication Complexity". In: *Proc. of TCC'08*. 2008, pp. 213–230.

[31] Z. Beerliová-Trubíniová and M. Hirt. "Simple and Efficient Perfectly-Secure Asynchronous MPC". In: *Proc. of ASIACRYPT'07*. 2007, pp. 376–392.

[32] Z. Beerliová-Trubíniová, M. Hirt, and J. B. Nielsen. "On the Theoretical Gap between Synchronous and Asynchronous MPC Protocols". In: *Proc. of PODC'10*. 2010, pp. 211–218.

[33] M. Bellare and O. Goldreich. "On Defining Proofs of Knowledge". In: *CRYPTO*. 1992, pp. 390–420.

[34] M. Ben-Or, R. Canetti, and O. Goldreich. "Asynchronous Secure Computation". In: *Proc. of STOC'93*. 1993, pp. 52–61.

[35] M. Ben-Or, S. Goldwasser, and A. Wigderson. "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation". In: *Proc. of STOC'88*. 1988, pp. 1–10.

[36] M. Ben-Or, B. Kelmer, and T. Rabin. "Asynchronous Secure Computations with Optimal Resilience (Extended Abstract)". In: *Proc. of PODC'94*. 1994, pp. 183–192.

[37] E. Ben-Sasson, S. Fehr, and R. Ostrovsky. "Near-Linear Unconditionally-Secure Multiparty Computation with a Dishonest Minority". In: *Proc. of CRYPTO'12*. 2012, pp. 663–680.

[38] B. Blanchet. "An Efficient Cryptographic Protocol Verifier Based on Prolog Rules". In: *14th IEEE Computer Security Foundations Workshop (CSFW-14)*. Cape Breton, Nova Scotia, Canada: IEEE Computer Society, June 2001, pp. 82–96.

[39] G. Bracha. "An Asynchronous [(n-1)/3]-Resilient Consensus Protocol". In: *Proc. of PODC'84*. 1984, pp. 154–162.

[40] T. D. Breaux and A. I. Antón. "Analyzing regulatory rules for privacy and security requirements". In: *IEEE Trans. on Software Engineering* 34.1 (2008), pp. 5–20.

[41] E. F. Brickell, J. Camenisch, and L. Chen. “Direct anonymous attestation”. In: *Proc. 11th ACM Conference on Computer and Communications Security*. ACM Press, 2004, pp. 132–145.

[42] C. Cachin, K. Kursawe, A.Lysyanskaya, and R. Strobl. “Asynchronous Verifiable Secret Sharing and Proactive Cryptosystems”. In: *Proc. of CCS’02*. 2002, pp. 88–97.

[43] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. “Secure and Efficient Asynchronous Broadcast Protocols”. In: *CRYPTO*. 2001, pp. 524–541.

[44] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. “Tractable Reasoning and Efficient Query Answering in Description Logics: The *DL-Lite* Family”. In: *Journal of Automed Reasoning* 39.3 (2007), pp. 385–429.

[45] J. Camenisch and M. Stadler. *Proof Systems for General Statements about Discrete Logarithms*. Tech. rep. 260, Dept. of Computer Science, ETH Zurich. 1997.

[46] R. Canetti. “Studies in Secure Multiparty Computation and Applications”. PhD thesis. The Weizmann Institute of Science, 1996.

[47] R. Canetti and J. Herzog. “Universally composable symbolic analysis of mutual authentication and key exchange protocols”. In: *Proc. 3rd Theory of Cryptography Conference (TCC)*. Vol. 3876. LNCS. Springer, 2006, pp. 380–403.

[48] A. Cavoukian. “Privacy by design”. In: *Report of the Information & Privacy Commissioner Ontario, Canada* (2012).

[49] D. Chaum, C. Crépeau, and I. Damgård. “Multiparty Unconditionally Secure Protocols”. In: *Proc. of STOC’88*. 1988, pp. 11–19.

[50] B. Chevallier-Mames, P. Paillier, and D. Pointcheval. “Encoding-Free Elgamal Encryption Without Random Oracles”. In: *Proc. of PKC’06*. 2006, pp. 91–104.

[51] A. Choudhury, M. Hirt, and A. Patra. “Asynchronous Multiparty Computation with Linear Communication Complexity”. In: *Proc. of DISC’13*. 2013, pp. 388–402.

[52] B.-G. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz. “Attested Append-only Memory: Making Adversaries Stick to their Word”. In: *Proc. of SOSP’07*. 2007, pp. 189–204.

[53] M. R. Clarkson, S. Chong, and A. C. Myers. “Civitas: Toward a Secure Voting System”. In: *Proceedings of the 2008 IEEE Symposium on Security and Privacy*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 354–368. ISBN: 978-0-7695-3168-7. URL: http://portal.acm.org/citation.cfm?id=1398035.

[54] A. Clement, F. Junqueira, A. Kate, and R. Rodrigues. “On the (limited) Power of Non-Equivocation”. In: *Proc. of PODC’12*. 2012, pp. 301–308.

[55] M. Correia, G. S. Veronese, and L. C. Lung. “Asynchronous Byzantine Consensus with 2f+1 Processes”. In: *Proc. of SAC’10*. 2010, pp. 475–480.

[56] V. Cortier and B. Warinschi. “A Composable Computational Soundness Notion. Chicago, USA, October 2011. ACM Press.” In: *Proc. 18th ACM Conference on Computer and Communications Security*. ACM Press, 2011.

[57] V. Cortier and B. Warinschi. “Computationally Sound, Automated Proofs for Security Protocols”. In: *Proc. 14th European Symposium on Programming (ESOP)*. 2005, pp. 157–171.

[58] R. Cramer, I. Damgård, and J. B. Nielsen. “Multiparty Computation from Threshold Homomorphic Encryption”. In: *Proc. of EUROCRYPT’01*. 2001, pp. 280–299.

[59] L. F. Cranor. “P3P”. In: *Making Privacy Policies More Useful, IEEE Security &Privacy. New York* (2003), pp. 50–55.

[60] I. Damgård and J. B. Nielsen. “Scalable and Unconditionally Secure Multiparty Computation”. In: *Advances in Cryptology—CRYPT0*. 2007, pp. 572–590.

[61] A. Datta, J. Blocki, N. Christin, H. DeYoung, D. Garg, L. Jia, D. Kaynar, and A. Sinha. “Understanding and protecting privacy: formal semantics and principled audit mechanisms”. In: *Information Systems Security*. Springer, 2011, pp. 1–27.

[62] H. DeYoung, D. Garg, D. Kaynar, and A. Datta. “Logical specification of the GLBA and HIPAA privacy laws”. In: *CyLab* (2010), p. 72.

[63] D. Dolev and H. R. Strong. “Authenticated Algorithms for Byzantine Agreement”. In: *SIAM J. Comput.* 12.4 (1983), pp. 656–666.

[64] D. Dolev and A. C. Yao. “On the Security of Public Key Protocols”. In: *IEEE Transactions on Information Theory* 29.2 (1983), pp. 198–208.

[65] C. Duma, A. Herzog, and N. Shahmehri. “Privacy in the semantic web: What policy languages have to offer”. In: *Proc. of POLICY*. IEEE. 2007, pp. 109–118.

[66] European Commission. *General Data Protection Regulation*. `http://ec.europa.eu/justice/data-protection/document/review2012/com_2012_11_en.pdf`.

[67] S. Even and O. Goldreich. “On the Security of Multi-Party Ping-Pong Protocols”. In: *Proc. 24th IEEE Symposium on Foundations of Computer Science (FOCS)*. 1983, pp. 34–39.

[68] P. Feldman and S. Micali. “Byzantine Agreement in Constant Expected Time (and Trusting No One)”. In: *FOCS*. 1985, pp. 267–276.

[69] C. Flavián and M. Guinalíu. “Consumer trust, perceived security and privacy policy: three basic elements of loyalty to a web site”. In: *Industrial Management & Data Systems* 106.5 (2006), pp. 601–620.

[70] D. Garg, L. Jia, and A. Datta. “Policy auditing over incomplete logs: theory, implementation and applications”. In: *Proc. of CCS*. ACM. 2011, pp. 151–162.

[71] R. Gennaro, M. Rabin, and T. Rabin. “Simplified VSS and Fast-Track Multiparty Computations with Applications to Threshold Cryptography”. In: *Proc. of PODC’98*. 1998, pp. 101–111.

[72] O. Goldreich, S. Micali, and A. Wigderson. “How to play ANY mental game”. In: *Proc. of STOC ’87*. 1987, pp. 218–229. ISBN: 0-89791-221-7.

[73] S. Goldwasser, S. Micali, and C. Rackoff. “The Knowledge Complexity of Interactive Proof Systems”. In: *SIAM Journal on Computing* 18.1 (1989), pp. 186–207.

[74] A. S. Gopal and S. Toueg. “Reliable Broadcast in Synchronous and Asynchronous Environments (Preliminary Version)”. In: *WDAG*. 1989, pp. 110–123.

[75] J. Groth. “Simulation-sound nizk proofs for a practical language and constant size group signatures”. In: *In proceedings of ASIACRYPT ’06, LNCS series*. Springer-Verlag, 2006, pp. 444–459.

[76] J. Groth and R. Ostrovsky. “Cryptography in the Multi-string Model”. In: *CRYPTO*. Ed. by A. Menezes. Vol. 4622. Lecture Notes in Computer Science. Full version available at `http://www.cs.ucla.edu/~rafail/PUBLIC/85.pdf`. The definition of extraction zero-knowledge is only contained in the full version. Springer, 2007, pp. 323–341. ISBN: 978-3-540-74142-8.

[77] S. Gürses, C. Gonzalez Troncoso, and C. Diaz. “Engineering privacy by design”. In: *Computers, Privacy & Data Protection* (2011).

[78] V. Hadzilacos and S. Toueg. *A Modular Approach to Fault-Tolerant Broadcasts and Related Problems*. Tech. rep. Ithaca, NY, USA, 1994.

[79] M. Hirt and J. B. Nielsen. “Robust Multiparty Computation with Linear Communication Complexity”. In: *Proc. of CRYPTO’06*. 2006, pp. 463–482.

[80] M. Hirt, J. B. Nielsen, and B. Przydatek. “Asynchronous Multi-Party Computation with Quadratic Communication”. In: *Proc. of ICALP’08*. 2008, pp. 473–485.

[81] M. Hirt, J. B. Nielsen, and B. Przydatek. “Cryptographic Asynchronous Multi-party Computation with Optimal Resilience (Extended Abstract)”. In: *Proc. of EUROCRYPT’05*. 2005, pp. 322–340.

[82] A. Jaffe, T. Moscibroda, and S. Sen. “On the Price of Equivocation in Byzantine Agreement”. In: *Proc. of PODC’12*. 2012, pp. 309–318.

[83] D. Kähler, R. Küsters, and T. Wilke. *Deciding Properties of Contract-Signing Protocols*. Tech. rep. IFI 0409. CAU Kiel, 2004.

[84] R. Kapitza, J. Behl, C. Cachin, T. Distler, S. Kuhnle, S. V. Mohammadi, W. Schröder-Preikschat, and K. Stengel. “CheapBFT: Resource-efficient Byzantine Fault Tolerance”. In: *Proc. of EuroSys’12*. 2012, pp. 295–308.

[85] J. Karat, C.-M. Karat, E. Bertino, N. Li, Q. Ni, C. Brodie, J. Lobo, S. Calo, L. Cranor, P. Kumaraguru, and R. Reeder. “Policy framework for security and privacy management.” In: *IBM Journal of Research and Development* 53.2 (2009), p. 4.

[86] P. G. Kelley, J. Bresee, L. F. Cranor, and R. W. Reeder. "A nutrition label for privacy". In: *Proceedings of the 5th Symposium on Usable Privacy and Security*. ACM. 2009, p. 4.

[87] R. Kemmerer, C. Meadows, and J. Millen. "Three Systems for Cryptographic Protocol Analysis". In: *Journal of Cryptology* 7.2 (1994), pp. 79–130.

[88] S. Kremer and M. Ryan. "Analysis of an Electronic Voting Protocol in the Applied Pi-Calculus." In: *Proc. 14th European Symposium on Programming (ESOP)*. LNCS. Springer-Verlag, 2005, pp. 186–200.

[89] R. Lämmel and E. Pek. "Understanding privacy policies". In: *Empirical Software Engineering* 18.2 (2013), pp. 310–374.

[90] L. Lamport, R. E. Shostak, and M. C. Pease. "The Byzantine Generals Problem". In: *ACM Trans. Program. Lang. Syst.* 4.3 (1982), pp. 382–401.

[91] P. Laud. "Symmetric Encryption in Automatic Analyses for Confidentiality against Active Adversaries". In: *Proc. 25th IEEE Symposium on Security & Privacy*. 2004, pp. 71–85.

[92] D. Levin, J. R. Douceur, J. R. Lorch, and T. Moscibroda. "TrInc: Small Trusted Hardware for Large Distributed Systems". In: *Proc. of NSDI'09*. 2009, pp. 1–14.

[93] H. Li and B. Li. "An Unbounded Simulation-Sound Non-interactive Zero-Knowledge Proof System for NP". In: *CISC*. 2005, pp. 210–220.

[94] G. Lowe. "Breaking and fixing the Needham-Schroeder public-key protocol using FDR". In: *Proc. 2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Vol. 1055. LNCS. Springer, 1996, pp. 147–166.

[95] L. Lu, J. Han, Y. Liu, L. Hu, J.-P. Huai, L. Ni, and J. Ma. "Pseudo Trust: Zero-Knowledge Authentication in Anonymous P2Ps". In: *IEEE Trans. Parallel Distrib. Syst.* 19 (10 2008), pp. 1325–1337. ISSN: 1045-9219. DOI: 10.1109/TPDS.2008.15. URL: http://dl.acm.org/citation.cfm?id=1449379.1449451.

[96] M. Maffei and K. Pecina. "Position Paper: Privacy-aware Proof-Carrying Authorization". In: *PLAS 2011*. To appear. 2011.

[97] M. Maffei, K. Pecina, and M. Reinert. "Security and privacy by declarative design". In: *Proc. of CSF*. IEEE. 2013, pp. 81–96.

[98] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig. "TrustVisor: Efficient TCB Reduction and Attestation". In: *Proc. of the 2010 IEEE Symposium on Security and Privacy*. 2010, pp. 143–158.

[99] M. Merritt. "Cryptographic Protocols". PhD thesis. Georgia Institute of Technology, 1983.

[100] D. Micciancio and B. Warinschi. "Soundness of Formal Encryption in the Presence of Active Adversaries". In: *Proc. 1st Theory of Cryptography Conference (TCC)*. Vol. 2951. LNCS. Springer, 2004, pp. 133–151.

[101] Q. Ni, E. Bertino, J. Lobo, C. Brodie, C.-M. Karat, J. Karat, and A. Trombeta. "Privacy-aware role-based access control". In: *Proc. of TISSEC* 13.3 (2010), p. 24.

[102] Office for Civil Rights, U.S. Department of Health and Human Services. *Summary of the HIPAA privacy rule*. 2003.

[103] S. E. Oh, J. Y. Chun, L. Jia, D. Garg, C. A. Gunter, and A. Datta. "Privacy-preserving audit for broker-based health information exchange". In: *Proc. of Data and application security and privacy*. ACM. 2014, pp. 313–320.

[104] A. Patra, A. Choudhary, and C. P. Rangan. "Efficient Asynchronous Byzantine Agreement with Optimal Resilience". In: *Proc. of PODC'09*. 2009, pp. 92–101.

[105] T. P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: *Proc. of CRYPTO'91*. 1991, pp. 129–140.

[106] T. Rabin and M. Ben-Or. "Verifiable Secret Sharing and Multiparty Protocols with Honest Majority (Extended Abstract)". In: *Proc. of STOC'89*. 1989, pp. 73–85.

[107] A. Sahai. "Non-Malleable Non-Interactive Zero Knowledge and Adaptive Chosen-Ciphertext Security". In: *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*. FOCS '99. Washington, DC, USA: IEEE Computer Society, 1999, pp. 543–. ISBN: 0-7695-0409-4. URL: `http://dl.acm.org/citation.cfm?id=795665.796535`.

[108] A. Sahai. *Simulation-Sound Non-Interactive Zero Knowledge*. Tech. rep. IBM RESEARCH REPORT RZ 3076, 2001.

[109] F. Salim, N. P. Sheppard, and R. Safavi-Naini. "Enforcing P3P policies using a digital rights management system". In: *Privacy Enhancing Technologies*. Springer. 2007, pp. 200–217.

[110] M. Schmidt-Schauß and G. Smolka. "Attributive Concept Descriptions with Complements". In: *Artificial Intelligence* 48.1 (1991), pp. 1–26.

[111] S. Schneider. "Security Properties and CSP". In: *Proc. 17th IEEE Symposium on Security & Privacy*. 1996, pp. 174–187.

[112] S. Sen, S. Guha, A. Datta, S. K. Rajamani, J. Tsai, and J. M. Wing. "Bootstrapping Privacy Compliance in Big Data Systems". In: *Proc. of S& P*.

[113] A. Shamir. "How to Share a Secret". In: *Commun. ACM* 22.11 (1979), pp. 612–613.

[114] K. Srinathan and C. P. Rangan. "Efficient Asynchronous Secure Multiparty Distributed Computation". In: *Proc. of INDOCRYPT'00*. 2000, pp. 117–129.

[115] M. Stadler. "Publicly Verifiable Secret Sharing". In: *Proc. of EUROCRYPT'96*. 1996, pp. 190–199.

[116] S. Toueg. "Randomized Byzantine Agreements". In: *Proc. of PODC'84*. 1984, pp. 163–178.

[117] M. C. Tschantz, A. Datta, and J. M. Wing. “Formalizing and enforcing purpose restrictions in privacy policies”. In: *Proc. of S& P*. IEEE. 2012, pp. 176–190.

[118] United States Congress. *Financial services modernization act of 1999*. 2010.

[119] United States federal law. *Children’s Online Privacy Protection Act*. 1998.

[120] A. Yao. “Protocols for secure computations”. In: *Proc. of FOCS’82*. IEEE, 1982, pp. 160–164.

Appendix A

Appendix

A.1 Postponed Soundness Proof Details

In this section, we give the complete soundness proof. We enumerated the lemmas as in chapter 2.

To prove the Theorem, we will use Theorem 2.1.1. Thus it is a simulator based proof. We first define the simulator in a generic way, such that it is easier to prove that the simulator is indistinguishable from a computational execution. Then we change this simulator leading to other simulators which are all indistinguishable. The last simulator in the chain of modified ones can then easily shown to be DY-style. Finally, combining these properties, we can apply Theorem 2.1.1 which then proves the Theorem.

Definition of the Simulator.. Given an adversary E and a polynomial p we construct a simulator Sim with respect to E and p. We assume that for each $m \in \{0,1\}^*$ there is an $N^m \in \mathbf{N}_E$. For a fixed execution, we may assume without loss of generality that the set $\mathbf{N}_P$ is split into two disjoint sets $\mathcal{N}$ and $\mathcal{R}$. Our protocol conditions enforce that nonces used for algorithmic randomness are not used somewhere else in the protocol. These will be considered to be in the set $\mathcal{R}$.

If the simulator receives a label L from a control node it forwards it to the adversary, waits for an answer, and forwards the answer to the protocol. For the other queries we will use three functions $\ell : \mathbf{T} \to \mathbb{N}$, $\beta : \mathbf{T} \to \{0,1\}^*$ and $\tau : \{0,1\}^* \to \mathbf{T}$ which are defined below. The simulator chooses for each $N \in \mathbf{N}_P$ an $r_N \in \{0,1\}^*$ (Sim samples according to A_N on the fly and caches the result). Upon receiving a $t \in \mathbf{T}$ from the protocol, the simulator computes $\beta(t)$ and forwards it to the adversary E. When it receives a $m \in \{0,1\}^*$ from the adversary it computes $\tau(m)$ and forwards the result to the protocol. Finally, when it receives $(\mathbf{info}, \nu, t)$ from the protocol it adds $\ell(t)$ to len and if $len > p(k)$ the simulator terminates, otherwise it answers $(\mathbf{proceed})$. Initially len is set to 0.

Remember, for a constructor ZK, we denote its computational implementation by A_{ZK}.

The partial functions $\beta : \mathbf{T} \to \{0,1\}^*$ and ℓ

- $\beta(N) \coloneqq r_N$ if $N \in \mathbf{N}_P$
- $\beta(N^m) \coloneqq m$
- $\beta(\mathsf{Enc}(\mathrm{ek}(N), t, M)) \coloneqq A_{\mathsf{Enc}}(A_{\mathrm{ek}}(r_N), \beta(t), r_M)$ if $M \in \mathbf{N}_P$
- $\beta(\mathsf{Enc}(\mathrm{ek}(M), t, N^m)) \coloneqq m$ if $M \in \mathbf{N}_P$
- $\beta(\mathrm{ek}(N)) \coloneqq A_{\mathrm{ek}}(r_N)$ if $N \in \mathbf{N}_P$
- $\beta(\mathrm{ek}(N^m)) \coloneqq m$
- $\beta(\mathrm{dk}(N)) \coloneqq A_{\mathrm{dk}}(r_N)$ if $N \in \mathbf{N}_P$
- $\beta(\mathrm{dk}(N^m)) \coloneqq d$ such that $\tau(d) = \mathrm{dk}(N^m)$ was computed earlier
- $\beta(\mathrm{sig}(\mathsf{sk}(N), t, M)) \coloneqq A_{\mathrm{sig}}(A_{\mathsf{sk}}(r_N), \beta(t), r_M)$ if $N, M \in \mathbf{N}_P$
- $\beta(\mathrm{sig}(\mathsf{sk}(M), t, N^s)) \coloneqq s$
- $\beta(\mathsf{vk}(N)) \coloneqq A_{\mathsf{vk}}(r_N)$ if $N \in \mathbf{N}_P$
- $\beta(\mathsf{vk}(N^m)) \coloneqq m$
- $\beta(\mathsf{sk}(N)) \coloneqq A_{\mathsf{sk}}(r_N)$ if $N \in \mathbf{N}_P$
- $\beta(\mathsf{sk}(N^m)) \coloneqq s$ such that $\tau(s) = \mathsf{sk}(N^m)$ was computed earlier
- $\beta(\mathrm{crs}(N)) \coloneqq A_{\mathrm{crs}}(r_N)$ if $N \in \mathbf{N}_P$
- $\beta(\mathrm{crs}(N^c)) \coloneqq c$
- $\beta(\mathrm{ZK}(\mathrm{crs}(N_1), t_1, t_2, N_2)) \coloneqq A_{\mathrm{ZK}}(A_{\mathrm{crs}}(r_{N_1}), \beta(t_1), \beta(t_2), r_{N_2})$ if $N_1, N_2 \in \mathbf{N}_P$
- $\beta(\mathrm{ZK}(\mathrm{crs}(t_0), t_1, t_2, N^s)) \coloneqq s$
- $\beta(\mathrm{pair}(t_1, t_2)) \coloneqq A_{\mathrm{pair}}(\beta(t_1), \beta(t_2))$
- $\beta(\mathrm{string}_0(t)) \coloneqq A_{\mathrm{string}_0}(\beta(t))$
- $\beta(\mathrm{string}_1(t)) \coloneqq A_{\mathrm{string}_1}(\beta(t))$
- $\beta(\mathrm{empty}) \coloneqq A_{\mathrm{empty}}()$
- $\beta(\mathrm{garbage}(N^c)) \coloneqq c$
- $\beta(\mathrm{garbageEnc}(t, N^c)) \coloneqq c$
- $\beta(\mathrm{garbageSig}(t, N^s)) \coloneqq s$
- $\beta(\mathrm{garbageZK}(t_1, t_2, N^z)) \coloneqq z$
- $\beta(t) \coloneqq \bot$ if no case matches

The function ℓ is defined by $\ell(t) := |\beta(t)|$.

The function $\tau : \{0,1\}^* \to \mathbf{T}$

(by taking the first matching rule)

- $\tau(r) \coloneqq N$ if $r = r_N$ for some $N \in \mathcal{N}$ and N occurred in a term sent from Π^C
- $\tau(r) \coloneqq N^r$ if r is of type nonce
- $\tau(c) \coloneqq \mathsf{Enc}(\mathrm{ek}(M), t, N)$ if c has earlier been output by $\beta(\mathsf{Enc}(\mathrm{ek}(M), t, N))$ for some $M \in \mathbf{N}, N \in \mathbf{N}_P$
- $\tau(c) \coloneqq \mathsf{Enc}(\mathrm{ek}(M), \tau(m), N^c)$ if c is of type ciphertext and $\tau(A_{\mathrm{ekof}}(c)) = \mathrm{ek}(M)$ for some $M \in \mathbf{N}_P$ and $m \coloneqq A_{\mathsf{Dec}}(A_{\mathrm{dk}}(r_N), c) \neq \bot$
- $\tau(c) \coloneqq \mathrm{garbageEnc}(\tau(A_{\mathrm{ekof}}(c)), N^c)$ if c is of type ciphertext
- $\tau(c) \coloneqq \mathrm{ek}(N)$ if $c = A_{\mathrm{ek}}(r_N)$ for some N that occurred in a subterm of the form $\mathrm{ek}(N)$

or $\mathrm{dk}(N)$ before
- $\tau(c) := \mathrm{ek}(N^c)$ if c is of type encryption key
- $\tau(c) := \mathrm{dk}(N)$ if $c = A_{\mathrm{dk}}(r_N)$ for some N that occurred in a subterm of the form $\mathrm{ek}(N)$ or $\mathrm{dk}(N)$ before
- $\tau(c) := \mathrm{dk}(N^e)$ if c is of type decryption key and e is the encryption key corresponding to c
- $\tau(s) := \mathrm{sig}(\mathsf{sk}(M), t, N)$ if s has earlier been output by $\beta(\mathrm{sig}(\mathsf{sk}(M), t, N))$ for some $M, N \in \mathbf{N}_P$
- $\tau(s) := \mathrm{sig}(\mathsf{sk}(M), \tau(m), N^s)$ if s is of type signature and $\tau(A_{\mathrm{vkof}}(s)) = \mathsf{vk}(M)$ for some $M \in \mathbf{N}$ and $m := A_{\mathrm{verify}}(A_{\mathrm{vkof}}(s), s) \neq \bot$
- $\tau(s) := \mathrm{garbageSig}(\tau(A_{\mathrm{vkof}}(s)), N^s)$ if s is of type signature
- $\tau(s) := \mathsf{vk}(N)$ if $s = A_{\mathsf{vk}}(r_N)$ for some N that occurred in a subterm of the form $\mathsf{vk}(N)$ or $\mathsf{sk}(N)$ before
- $\tau(s) := \mathsf{vk}(N^s)$ if s is of type verification key
- $\tau(s) := \mathsf{sk}(N)$ if $s = A_{\mathsf{sk}}(r_N)$ for some N that occurred in a subterm of the form $\mathsf{vk}(N)$ or $\mathsf{sk}(N)$ before
- $\tau(s) := \mathsf{sk}(N^c)$ if s is of type signing key and c is the signing key corresponding to s
- $\tau(z) := \mathrm{crs}(N)$ if $z = A_{\mathrm{crs}}(r_N)$ for some N that occurred in a subterm of the form $\mathrm{crs}(N)$ before
- $\tau(z) := \mathrm{crs}(N^z)$ if z is of type common reference string
- $\tau(z) := \mathrm{ZK}(\mathrm{crs}(N_1), t_1, t_2, N_2)$ if z has earlier been output by $\beta(\mathrm{ZK}(\mathrm{crs}(N_1), t_1, t_2, N_2))$ for some $N_1, N_2 \in \mathbf{N}_P$
- $\tau(z) := \mathrm{ZK}(\mathrm{crs}(N), x, w, N^z)$ if z is of type zero-knowledge proof and $\tau(z)$ was computed earlier and has output $\mathrm{ZK}(\mathrm{crs}(N), x, w, N^z)$
- $\tau(z) := \mathrm{ZK}(\mathrm{crs}(N), x, w, N^z)$ if z is of type zero-knowledge proof, $\tau(A_{\mathrm{crsof}}(z)) = \mathrm{crs}(N)$ for some $N \in \mathbf{N}_P$, $A_{\mathrm{verify}_{\mathrm{ZK}}}(A_{\mathrm{crsof}}(z), z) = z$, $m_x := A_{\mathrm{getPub}}(z) \neq \bot$, $x := \tau(m_x) \neq \bot$ and $w := \mathbf{SymbExtr}(S, x)$ where S is the set of terms sent to the adversary so far.
  If $w = \bot$, we say an *extraction-failure* on (N, z, m_x) occurred, see below for the behavior of Sim in this case.
- $\tau(z) := \mathrm{garbageZK}(c, x, N^z)$ if z is of type zero-knowledge proof, $c := \tau(A_{\mathrm{crsof}}(z))$ and $x := \tau(A_{\mathrm{getPub}}(z))$.
- $\tau(m) := \mathrm{pair}(\tau(A_{\mathrm{fst}}(m)), \tau(A_{\mathrm{snd}}(m)))$ if m is of type pair
- $\tau(m) := \mathrm{string}_0(\tau(m'))$ if m is of type payload-string and $m' := A_{\mathrm{unstring}_0}(m) \neq \bot$
- $\tau(m) := \mathrm{string}_1(\tau(m'))$ if m is of type payload-string and $m' := A_{\mathrm{unstring}_1}(m) \neq \bot$
- $\tau(m) := \mathrm{empty}$ if m is of type payload-string and $m = A_{\mathrm{empty}}()$
- $\tau(m) := \mathrm{garbage}(N^m)$ otherwise

When an extraction-failure on (N, z, m_x) occurs (i.e., when in the computation of τ, $\mathbf{SymbExtr}(S, x)$ returns $w = \bot$), the simulator computes $(\mathrm{crs}, \mathrm{simtd}, \mathrm{extd}) \leftarrow \mathbf{K}(1^\eta; r_N)$ to get the extraction trapdoor extd corresponding to $\mathrm{crs} = A_{\mathrm{crs}}(r_N)$. Then the simulator invokes $m_w := \mathbf{E}(m_x, z, \mathrm{extd})$ and computes $x := \tau(m_x)$ as well as $w := \tau^*(m_w)$. If $(x, w) \notin R_{\mathrm{adv}}^{\mathrm{sym}}$,

we say a *ZK-break* occurred. Then (no matter whether a ZK-break occurred or not), the simulator aborts.

We define τ^* by the same case distinction as τ but remove the case in which an extraction failure may occur (i.e., the case where we invoke $\mathbf{SymbExtr}(S, x)$). Consequently, every adversary generated ZK-proof is parsed as $\mathrm{garbageZK}$ by τ^*. Thus, by definition, there is no extraction failure during a computation of τ^*.

Soundness Proof.. The previously defined simulator is indistinguishable from a computational execution and DY. To prove this we start by constructing a faking simulator in several steps. The construction is split in steps because it is easier to prove some properties for the intermediate simulators and show that they carry over to the final one than showing them for the final simulator directly. Thus, in the following subsection, we define the faking simulator in detail.

The faking simulator.

- We define Sim_1 like Sim but we change β to use zero-knowledge oracles instead of computing A_{crs} and A_{ZK}. More precisely, assume an oracle $\mathcal{O}_{\mathrm{ZK}}$ that internally picks $(\mathrm{crs}, \mathrm{simtd}, \mathrm{extd}) \leftarrow \mathbf{K}(1^\eta)$ and that responds to three kinds of queries: Upon a $(\mathtt{crs})$-query, it returns crs, and upon a $(\mathtt{prove}, x, w)$-query, it returns $\mathbf{P}(x, w, \mathrm{crs})$ if $(x, w) \in R^{\mathrm{comp}}_{\mathrm{honest}}$ and $\perp$ otherwise. Upon a $(\mathtt{extd})$-query, it returns extd. For each $N \in \mathbf{N}_P$, Sim_1 maintains an instance $\mathcal{O}^N_{\mathrm{ZK}}$ of $\mathcal{O}_{\mathrm{ZK}}$. Then Sim_1 computes $\beta(\mathrm{crs}(N))$ with $N \in \mathbf{N}_P$ as $\beta(\mathrm{crs}(N)) := \mathcal{O}^N_{\mathrm{ZK}}(\mathtt{crs})$, and Sim_1 computes $\beta(\mathrm{ZK}(\mathrm{crs}(N_1), t_1, t_2, N_2))$ with $N_1, N_2 \in \mathbf{N}_P$ as $\beta(\mathrm{ZK}(\mathrm{crs}(N_1), t_1, t_2, N_2)) := \mathcal{O}^N_{\mathrm{ZK}}(\mathtt{prove}, \beta(t_1), \beta(t_2))$. In case of an extraction-failure, Sim_1 performs a $(\mathtt{extd})$-query to get extd. (Here and in the descriptions of $\mathrm{Sim}_2, \ldots, \mathrm{Sim}_5, \mathrm{Sim}_f$, we implicitly require that $\beta(t)$ caches the results of the oracle queries and does not repeat the oracle query when β is applied to the *same* term t again.)
  In the definition of $\tau(z) = \mathrm{crs}(N)$ for $N \in \mathbf{N}_P$, instead of checking $z = A_{\mathrm{crs}}(r_N)$, Sim_1 checks whether z is equal to the $(\mathtt{crs})$-query outcomes for all oracles $\mathcal{O}^N_{\mathrm{ZK}}$ which have been used so far.
- We define Sim_2 like Sim_1, except that we replace the oracle $\mathcal{O}_{\mathrm{ZK}}$ by an oracle $\mathcal{O}_{\mathrm{sim}}$. That oracle behaves like $\mathcal{O}_{\mathrm{ZK}}$, except that upon a $(\mathtt{prove}, x, w)$-query, it returns $\mathbf{S}(x, \mathrm{crs}, \mathrm{simtd})$ if $(x, w) \in R^{\mathrm{comp}}_{\mathrm{honest}}$ and $\perp$ otherwise.
- We define Sim_3 like Sim_2, except that we replace the oracle $\mathcal{O}_{\mathrm{sim}}$ by an oracle $\mathcal{O}'_{\mathrm{sim}}$. That oracle behaves like $\mathcal{O}_{\mathrm{sim}}$, except that upon a $(\mathtt{prove}, x, w)$-query, it returns $\mathbf{S}(x, \mathrm{crs}, \mathrm{simtd})$ (even if $(x, w) \notin R^{\mathrm{comp}}_{\mathrm{honest}}$). Therefore the simulator only queries $(prove, x)$ and does not compute w any more.
- We define Sim_4 like Sim_3, but we change β and τ to use encryption oracles instead of computing A_{Enc}, A_{Dec}, A_{ek}, A_{dk}. More precisely, assume an oracle $\mathcal{O}_{\mathsf{Enc}}$ that internally picks $(\mathrm{ek}, \mathrm{dk}) \leftarrow \mathrm{KeyGen}_{\mathsf{Enc}}(1^\eta)$ and that responds to three kinds of queries: Upon an $(\mathtt{ek})$-query, it returns ek. Upon a $(\mathtt{enc}, m)$-query, it returns $\mathrm{ENC}(\mathrm{ek}, m)$. Upon a $(\mathtt{dec}, c)$-query, it returns $\mathrm{DEC}(\mathrm{dk}, c)$. Sim_4 maintains an instance $\mathcal{O}^N_{\mathrm{enc}}$ for each

$N \in \mathbf{N}_P$. Then Sim_4 computes $\beta(\mathrm{ek}(N))$ with $N \in \mathbf{N}_P$ as $\beta(\mathrm{ek}(N)) := \mathcal{O}^N_{\mathrm{enc}}(\mathtt{ek})$. And it computes $\beta(\mathsf{Enc}(\mathrm{ek}(N), t, M))$ with $N, M \in \mathbf{N}_P$ as $\beta(\mathsf{Enc}(\mathrm{ek}(N), t, M)) := \mathcal{O}^N_{\mathrm{enc}}(\mathtt{enc}, \beta(t))$. And it computes $\beta(\mathrm{dk}(N)) := \bot$. And in the computation of $\tau(c)$ for c of type ciphertext, the computation of $A_{\mathsf{Dec}}(A_{\mathrm{dk}}(r_N), c)$ is replaced by $\mathcal{O}^N_{\mathrm{enc}}(\mathtt{dec}, c)$. In the definition of $\tau(c) = \mathrm{ek}(N)$ and $\tau(c) = \mathrm{dk}(N)$ for $N \in \mathbf{N}_P$, instead of checking $c = A_{\mathrm{ek}}(r_N)$ and $c = A_{\mathrm{dk}}(r_N)$, Sim_4 checks whether c is equal to the corresponding query outcomes for all oracles $\mathcal{O}^N_{\mathsf{Enc}}$ which have been used so far.

- We define Sim_5 like Sim_4, except that we replace the oracle $\mathcal{O}_{\mathsf{Enc}}$ by an oracle $\mathcal{O}_{\mathrm{fake}}$. That oracle behaves like $\mathcal{O}_{\mathsf{Enc}}$, except that upon an $(\mathtt{enc}, x)$-query, it returns $\mathrm{ENC}(\mathrm{ek}, 0^{|x|})$.
- We define Sim_f like Sim_5, but we change β to use signing oracles instead of computing $A_{\mathsf{vk}}, A_{\mathsf{sk}}, A_{\mathrm{sig}}$. More precisely, we assume an oracle $\mathcal{O}_{\mathrm{sig}}$ that internally picks $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathrm{KeyGen}_{\mathrm{sig}}(1^\eta)$ and that responds to two kinds of queries: Upon a $(\mathtt{vk})$-request, it returns vk, and upon a $(\mathtt{sig}, m)$-request, it returns $\mathrm{SIG}(\mathsf{sk}, m)$. Sim_f maintains an instance $\mathcal{O}^N_{\mathrm{sig}}$ for each $N \in \mathbf{N}_P$. Then Sim_f computes $\beta(\mathsf{vk}(N))$ with $N \in \mathbf{N}_P$ as $\beta(\mathsf{vk}(N)) := \mathcal{O}^N_{\mathrm{sig}}(\mathtt{vk})$. And $\beta(\mathsf{sk}(N))$ with $N \in \mathbf{N}_P$ as $\beta(\mathsf{sk}(N)) := \bot$. And $\beta(\mathrm{sig}(\mathsf{sk}(N), t, M))$ with $N, M \in \mathbf{N}_P$ as $\beta(\mathrm{sig}(\mathsf{sk}(N), t, M)) := \mathcal{O}^N_{\mathrm{sig}}(\mathtt{sig}, \beta(t))$. In the definition of $\tau(c) = \mathsf{vk}(N)$ and $\tau(c) = \mathsf{sk}(N)$ for $N \in \mathbf{N}_P$, instead of checking $c = A_{\mathsf{vk}}(r_N)$ and $c = A_{\mathsf{sk}}(r_N)$, Sim_f checks whether c is equal to the corresponding query outcomes for all oracles $\mathcal{O}^N_{\mathrm{sig}}$ which have been used so far.

Dolev-Yaoness

The next steps towards the soundness proof are the following. First, we analyze the underivable terms structure. Doing so, we exclude cases in the proof of DY-ness using structural arguments. Thus, when showing DY-style, we only need to consider the cases involving cryptographic arguments.

Lemma A.1.1. *For any invocation of τ or τ^* in the hybrid execution of Sim_f, let m denote the input to τ or τ^*, let u' denote the output of τ or τ^*, and let S be the set of all messages sent from Π^C to Sim_f up to that invocation of τ or τ^*.*

Let C be a context and $u \in \mathbf{T}$ such that $u' = C[u]$ and $S \not\vdash u$.

Then there is a term t_{bad} and a context D such that D can be obtained by the following grammar:

$$\begin{aligned} D ::= \Box \mid \mathrm{pair}(t, D) \mid \mathrm{pair}(D, t) \mid \mathsf{Enc}(\mathrm{ek}(N), D, M) \\ \mid \mathsf{Enc}(D, t, M) \mid \mathrm{sig}(\mathsf{sk}(M), D, M') \\ \mid \mathrm{ZK}(t, D, t', M) \mid \mathrm{ZK}(D, t, t', M) \\ \mid \mathrm{garbageEnc}(D, M) \mid \mathrm{garbageSig}(D, M) \\ \mid \mathrm{garbageZK}(D, t, M) \mid \mathrm{garbageZK}(t, D, M) \\ \text{with } M, M' \in \mathbf{N}_E, t, t' \in \mathbf{T} \end{aligned}$$

with $u = D[t_{bad}]$ such that $S \not\vdash t_{bad}$ and such that one of the following holds:

1. $t_{bad} \in \mathcal{N}$
2. $t_{bad} = \mathsf{Enc}(p, m, N)$ *with* $N \in \mathbf{N}_P$
3. $t_{bad} = \mathrm{sig}(k, m, N)$ *with* $N \in \mathbf{N}_P$
4. $t_{bad} = \mathrm{ZK}(\mathrm{crs}(M), x, w, N)$ *with* $M, N \in \mathbf{N}_P$
5. $t_{bad} = \mathrm{sig}(\mathsf{sk}(N), m, M)$ *with* $N \in \mathbf{N}_P$, $M \in \mathbf{N}_E$
6. $t_{bad} = \mathrm{crs}(N)$ *with* $N \in \mathbf{N}_P$
7. $t_{bad} = \mathrm{ek}(N)$ *with* $N \in \mathbf{N}_P$
8. $t_{bad} = \mathsf{vk}(N)$ *with* $N \in \mathbf{N}_P$
9. $t_{bad} = \mathsf{sk}(N)$ *with* $N \in \mathbf{N}_P$
10. $t_{bad} = \mathrm{dk}(N)$ *with* $N \in \mathbf{N}_P$

Proof. We prove the lemma by structural induction on u. We formulate the proof for an invocation of τ, for an invocation of τ^* the proof is identical. There are the following cases:

Case 1: " $u \in \{\mathrm{ek}(N), \mathsf{vk}(N), \mathrm{crs}(N), \mathrm{dk}(N), \mathsf{sk}(N)\}$ with $N \notin \mathbf{N}_P$"

Let $u = C(N)$ for $C \in \{\mathrm{ek}, \mathsf{vk}, \mathrm{crs}, \mathrm{dk}, \mathsf{sk}\}$. By protocol conditions 1 and 8 each C-node has as annotation a nonce from $\mathbf{N}_P$. Therefore u cannot be honestly generated, that means there is some $e \in \{0,1\}^*$ such that $\tau(e) = u$ and u has the form $C(N^e)$. But then $S \vdash u$ contradicting the premise of the lemma.

Case 2: " $u \in \{\mathrm{ek}(N), \mathsf{vk}(N), \mathrm{crs}(N), \mathrm{dk}(N), \mathsf{sk}(N)\}$ with $N \in \mathbf{N}_P$"

Then the claim is fulfilled with $D \coloneqq \Box$ and $t_{bad} = u$.

Case 3: "$u = \mathrm{garbage}(u_1)$"

By protocol condition 2 no garbage term is generated by the protocol. Therefore there is a $c \in \{0,1\}^*$ such that $\tau(c) = \mathrm{garbage}(N^c) = u$. But this means that $S \vdash u$, contradicting the premise of the lemma.

Case 4: "$u = \mathrm{garbageEnc}(u_1, u_2)$ or $u = \mathrm{garbageSig}(u_1, u_2)$"

By protocol condition 2 no garbage term is generated by the protocol. So there exists a $c \in \{0,1\}^*$ with $\tau(c) = \mathrm{garbageEnc}(u_1, N^c)$ or $\tau(c) = \mathrm{garbageSig}(u_1, N^c)$. Since $S \vdash N^c$ it follows that $S \nvdash u_1$, because $S \nvdash u$. Applying the induction hypothesis on u_1 leads to a context D' and a term t_{bad}. Using this term t_{bad} and the context $\mathrm{garbageEnc}(D', N^c)$, respectively $\mathrm{garbageSig}(D', N^c)$, shows the claim.

Case 5: "$u = \mathrm{garbageZK}(u_1, u_2, u_3)$"

As in the previous case follows $u_3 = N^c$ with $c \in \{0,1\}^*$, so $S \nvdash u_1$ or $S \nvdash u_2$. For the first case we can apply the induction hypothesis to u_1 leading to t_{bad} and context D'. Then we use context $\mathrm{garbageZK}(D', u_2, u_3)$ to satisfy the lemma. In the other case we apply the induction hypothesis to u_2 leading to context $\mathrm{garbageZK}(u_1, D', u_3)$ and t_{bad}.

Case 6: "$u = \mathrm{pair}(u_1, u_2)$"

Since $S \not\vdash u$ there is an $i \in \{1, 2\}$ such that $S \not\vdash u_i$. Let D be the context and t_{bad} the term given by applying the induction hypothesis to u_i. Then $D_1 \coloneqq \mathrm{pair}(D, M)$ or $D_2 \coloneqq \mathrm{pair}(M, D)$ is the context for the term u depending on i with the same term t_{bad}.

Case 7: "$u = \mathrm{empty}$"

This case cannot happen because $S \vdash \mathrm{empty}$, so the premise of the lemma is not fulfilled.

Case 8: "$u = \mathrm{string}_0(u_1)$ or $u = \mathrm{string}_1(u_1)$"

Again the premise is not fulfilled since inductively $S \vdash u_1$ with base case $u_1 = \mathrm{empty}$ and therefore $S \vdash \mathrm{string}_\mathrm{i}(u_1)$ for $i \in \{0, 1\}$.

Case 9: "$u = N$ with $N \in \mathbf{N}_P \backslash \mathcal{N}$"

This case is impossible since u is not in the range of τ.

Case 10: "$u = N$ with $N \in \mathcal{N}$"

The context $D \coloneqq \square$ and term $t_{bad} := u$ satisfy the lemma in this case.

Case 11: "$u = N$ with $N \in \mathbf{N}_E$"

In this case $S \vdash u$ by definition and therefore the lemma's premise does not hold.

Case 12: "$u = \mathsf{Enc}(u_1, u_2, N)$ with $N \in \mathbf{N}_P$"

The lemma is satisfied by $t_{bad} = u$ and $D = \square$.

Case 13: "$u = \mathsf{Enc}(u_1, u_2, u_3)$ with $u_3 \notin \mathbf{N}_P$ and $S \not\vdash u_1$"

Since $u_3 \notin \mathbf{N}_P$ it follows that u cannot be honestly generated because of protocol condition 3. Therefore there is a $c \in \{0, 1\}^*$ with $\tau(c) = \mathsf{Enc}(\mathrm{ek}(M), u_2, N^c) = u$ for some $M \in \mathbf{N}_P$. Apply the induction hypothesis to u_1 getting t_{bad} and context D we can define $D' \coloneqq \mathsf{Enc}(D, u_2, N^c)$ fulfilling the claim of the lemma with t_{bad}.

Case 14: "$u = \mathsf{Enc}(u_1, u_2, u_3)$ with $u_3 \notin \mathbf{N}_P$ and $S \vdash u_1$"

Since $u_3 \notin \mathbf{N}_P$ it follows that u cannot be honestly generated because of protocol condition 3. Therefore there is an $c \in \{0, 1\}^*$ with $\tau(c) = \mathsf{Enc}(\mathrm{ek}(M), u_2, N^c) = u$ for some $M \in \mathbf{N}_P$. Since $S \vdash u_1$, $S \vdash N^c$, and $S \not\vdash u$, it follows that $S \not\vdash u_2$. Let D be the context and t_{bad} be the term resulting by the induction hypothesis applied to u_2. Then $D' \coloneqq \mathsf{Enc}(\mathrm{ek}(M), D, N^c)$ together with t_{bad} satisfies the lemma.

Case 15: "$u = \mathrm{sig}(u_1, u_2, N)$ with $N \in \mathbf{N}_P$"

Use context $D \coloneqq \square$ and $t_{bad} = u$.

Case 16: "$u = \mathrm{sig}(\mathsf{sk}(N), u_1, u_3)$ with $u_3 \notin \mathbf{N}_P$ and $N \in \mathbf{N}_P$"

Since $u \in \mathbf{T}$ and $u_3 \notin \mathbf{N}_P$ follows that $u_3 \in \mathbf{N}_E$. Therefore the context $D := \Box$ and $t_{bad} = u$ proves the claim.

Case 17: "$u = \mathrm{sig}(u_1, u_2, u_3)$ and $u_3 \notin \mathbf{N}_P$ and u_1 is not of the form $\mathsf{sk}(N)$ with $N \in \mathbf{N}_P$"

Since $u_3 \notin \mathbf{N}_P$ we get by protocol condition 3 that u is not honestly generated, i.e., there is an $s \in \{0,1\}^*$ such that $\tau(s) = \mathrm{sig}(\mathsf{sk}(M), u_2, N^s) = u$ with $M \in \mathbf{N}$. Because u_1 has not the form $\mathsf{sk}(N)$ for any $N \in \mathbf{N}_P$ follows that $M \in \mathbf{N}_E$, so $S \vdash M$ and therefore $S \vdash \mathsf{sk}(M)$. In total we have $S \vdash u_1$, $S \vdash u_3$ but $S \nvdash u$ which implies that $S \nvdash u_2$. Applying the induction hypothesis to u_2 leads to a context D and a term t_{bad}. Defining $D' := \mathrm{sig}(\mathsf{sk}(M), D, N^s)$ completes the claim.

Case 18: "$u = \mathrm{ZK}(\mathrm{crs}(M), u_1, u_2, N)$ with $N, M \in \mathbf{N}_P$"

Defining $t_{bad} = u$ and $D := \Box$ suffices.

Case 19: "$u = \mathrm{ZK}(\mathrm{crs}(M), u_1, u_2, N)$ with $N \notin \mathbf{N}_P$, $M \in \mathbf{N}_P$"

Consider the following cases:

- $S \nvdash \mathrm{crs}(M)$
  Define $t_{bad} = \mathrm{crs}(M)$ and $D := \mathrm{ZK}(\Box, u_1, u_2, N)$ to satisfy the lemma.
- $S \nvdash u_2$
  Since $N \notin \mathbf{N}_P$ the term u was not honestly generated. That means that u_2 was constructed using the $\mathbf{SymbExtr}$ and therefore $S \vdash u_2$. So this case cannot happen.
- $S \nvdash u_1$
  In this case we use the induction hypothesis on u_1 to get the term t_{bad} and a context D. Then using t_{bad} and $D' := \mathrm{ZK}(\mathrm{crs}(M), D, u_2, N)$ satisfies the lemma.

Case 20: "$u = \mathrm{ZK}(\mathrm{crs}(M), u_1, u_2, N)$ with $M \notin \mathbf{N}_P$"

This case cannot occur because u is not in the range of τ.

□

In any hybrid execution, terms are generated via the functions τ and τ^*. In their definition, the case distinction whether β outputted the input bitstring or not is done very often. Thus, in the following lemma, we show that β is only called on derivable terms in the execution of Sim_f.

Lemma A.1.2. *For any (direct or recursive) call of the function $\beta(t)$ performed by Sim_f, it holds that $S \vdash t$ where S is the set of all terms sent by Π^C to Sim_f up to that point.*

Proof. Prove it by induction on the recursion depth of the β-function. The base case is that $\beta(t)$ is directly invoked. But then t itself was received by the protocol, i.e., $t \in S$ and therefore $S \vdash t$.

So let $\beta(t)$ be called as subroutine of some t'. By induction hypothesis we have $S \vdash t'$. We need to show that $S \vdash t$. According to the definition of β there are the following possibilities for t':

1. $t' = \mathrm{sig}(\mathsf{sk}(N), t, M)$ with $N, M \in \mathbf{N}_P$
2. $t' = \mathrm{pair}(t_1, t_2)$ with $t \in \{t_1, t_2\}$
3. $t' = \mathrm{string}_0(t)$ or $t' = \mathrm{string}_1(t)$
4. $t' = \mathsf{Enc}(\mathrm{ek}(N^e), t, M)$ with $M \in \mathbf{N}_P$
5. $t' = \mathrm{ZK}(\mathrm{crs}(M), t, t_2, N)$ with $N, M \in \mathbf{N}_P$

Note that the case $t' = \mathsf{Enc}(\mathrm{ek}(N), t, M)$ with $N, M \in \mathbf{N}_P$ does not occur because – in contrast to Sim – the simulator Sim_f does not recursively invoke β on t but uses an oracle and produces $\mathrm{ENC}(\mathrm{ek}_N, 0^{\ell(t)})$. The case $t' = \mathrm{ZK}(\mathrm{crs}(M), t_1, t, N)$ is not possible, either, because the simulator Sim_f calls the simulation oracle to construct the proof and therefore $\beta(\cdot)$ is not called on the witness t.

Case 1: $S \vdash \mathrm{sig}(\mathsf{sk}(N), t, M) = t'$. Using $\mathrm{verify}(\mathrm{vkof}(t'), t') = t$ we get $S \vdash t$.

Case 2: $S \vdash \mathrm{pair}(t_1, t_2) = t'$. With $\mathrm{fst}(t') = t_1$, $\mathrm{snd}(t') = t_2$, and $t \in \{t_1, t_2\}$ we get $S \vdash t$.

Case 3: The cases $t' = \mathrm{string}_0(t)$ and $t' = \mathrm{string}_1(t)$ work as the two preceding using $\mathrm{unstring}_0$ and $\mathrm{unstring}_1$.

Case 4: $S \vdash \mathsf{Enc}(\mathrm{ek}(N^e), t, M)$. Because $S \vdash N^e$ it follows that $S \vdash \mathrm{dk}(N^e)$, so decryption can be applied resulting in t.

Case 5: $S \vdash \mathrm{ZK}(\mathrm{crs}(M), t, t_2, N) = t'$. The lemma follows by applying the destructor getPub.

□

We combine the preceding lemmas to achieve DY-style of Sim_f. The lemma is generalized to not only show DY-style of Sim_f, but also that each output of τ and τ^* in an execution is derivable. Doing so, we are able to reuse the lemma when proving the absence of extraction failures.

Lemma A.1.3 (Sim_f is Dolev-Yao)**.** *For any invocation $\tau(m)$ of τ or $\tau^*(m)$ of τ^* in the hybrid execution of Sim_f, the following holds with overwhelming probability: Let S be the set of terms t that the protocol sent to the adversary up to the invocation $\tau(m)$ or $\tau^*(m)$. Then $S \vdash \tau(m)$ or $S \vdash \tau^*(m)$, respectively.*

In particular, Sim_f is DY for $\mathbf{M}$ and Π.

Proof. Assume there occurs an m as input of τ or τ^* such that $S \not\vdash \tau(m)$ or $S \not\vdash \tau^*(m)$, respectively. Consider the first such input m.

Now we can use lemma A.1.1 with context $C = \Box$ and $u' = u = t$ leading to a term t_{bad} and a context D such that t_{bad} is of the form 1-10 given by the lemma. Let m_{bad} be the corresponding bitstring, i.e. $\tau(m_{bad}) = t_{bad}$. For each of these cases we will derive that it can only happen with negligible probability. Note that τ^* only differs from τ in the case a ZK-proof $\mathrm{ZK}(\mathrm{crs}(N), x, w, M)$ is output with $M \in \mathbf{N}_E$. We formulate the proof for an invocation of τ; the case of τ^* is identical.

Case 1: "$t_{bad} = N \in \mathcal{N}$".
By construction of β and Sim_f follows that Sim_f has only access to r_N if $\beta(N)$ is computed directly or in τ. Because $S \not\vdash N$ we get by Lemma A.1.2 that β was never invoked on N, i.e. Sim_f has only access to r_N via τ. Considering the definition of τ, we see that r_N is used for comparisons. In particular, if $\tau(r)$ is called for an r having type nonce then the simulator checks for all $N \in \mathbf{N}_P$ that occurred in a term sent by the protocol, whether $r = r_N$. This check does not help guessing r_N because it only succeeds if r_N was guessed correctly and therefore the probability that $m_{bad} = r_N$ as input of τ is negligible.

Case 2: "$t_{bad} = \mathsf{Enc}(p, m, N)$ with $N \in \mathbf{N}_P$".
By definition τ only returns t_{bad} if $\beta(t_{bad})$ was called earlier. But since $S \not\vdash t_{bad}$ and Lemma A.1.2 this case cannot occur.

Case 3: "$t_{bad} = \mathrm{sig}(k, m, N)$ with $N \in \mathbf{N}_P$".
This case is completely analogue to the case that $t_{bad} = \mathsf{Enc}(p, m, N)$ with $N \in \mathbf{N}_P$.

Case 4: "$t_{bad} = \mathrm{ZK}(\mathrm{crs}(M), x, w, N)$ with $N, M \in \mathbf{N}_P$".
By definition of τ, t_{bad} is only returned if it was a result of $\beta(t_{bad})$ earlier. But because $S \not\vdash t_{bad}$ and Lemma A.1.2 this can not be the case.

Case 5: "$t_{bad} = \mathrm{crs}(N)$ with $N \in \mathbf{N}_P$".
By definition of τ, the oracle $\mathcal{O}^N_{\mathrm{ZK}}$ constructed the bitstring m_{bad}. Thus β was either called on $\mathrm{crs}(N)$ or on some ZK-proof of the form $\mathrm{ZK}(\mathrm{crs}(N), \cdot, \cdot, \cdot)$. In the first case, by Lemma A.1.2, it follows $S \vdash t_{bad}$. In the latter case, by the same lemma, it follows $S \vdash \mathrm{ZK}(\mathrm{crs}(N), \cdot, \cdot, \cdot)$ and thus $S \vdash \mathrm{crs}(N)$ using destructor getPub.

Case 6: "$t_{bad} = \mathrm{sig}(\mathsf{sk}(N), m', M)$ with $N \in \mathbf{N}_P$, $M \in \mathbf{N}_E$".
Because $S \not\vdash t_{bad}$ follows that β was not invoked on t_{bad}. Therefore m_{bad} is a signature that was not produced by the signing oracle, but it is valid with respect to verification key vk_N. Because of the strong existential unforgeability this can only be the case with negligible probability.[1]

[1] Note that an adversary against this property is allowed to use the extraction trapdoor. The same holds the property of CCA.

Case 7: "$t_{bad} = \mathrm{ek}(N)$ with $N \in \mathbf{N}_P$".

By definition of τ and since $\tau(m_{bad}) = \mathrm{ek}(N)$, it follows that the oracle $\mathcal{O}^N_{\mathsf{Enc}}$ produced this key in an earlier call of β. Thus one of the following terms have been called by β earlier: $\mathrm{ek}(N)$, $\mathrm{dk}(N)$, or $\mathsf{Enc}(\mathrm{ek}(N), \cdot, \cdot)$. The case $\mathrm{dk}(N)$ is impossible because dk is only allowed to be part of the witness in ZK proofs and in decryptions (protocol conditions 4). Since the witnesses are not computed using β in Sim_f, it follows that $\mathrm{dk}(N)$ can not be input to β at all.

Considering the remaining cases, it follows by Lemma A.1.2 that either $S \vdash \mathrm{ek}(N)$ or $S \vdash \mathsf{Enc}(\mathrm{ek}(N), \cdot, \cdot)$. In the latter case $S \vdash \mathrm{ek}(N)$ using destructor ekof. So $S \nvdash t_{bad}$ is impossible.

Case 8: "$t_{bad} = \mathsf{vk}(N)$ with $N \in \mathbf{N}_P$".

This case is analogue to the case $t_{bad} = \mathrm{ek}(N)$ with the possible oracle queries in while computing β on $\mathsf{vk}(N)$, $\mathsf{sk}(N)$, or $\mathrm{sig}(\mathsf{sk}(N), \cdot, \cdot)$. The case $\mathsf{sk}(N)$ corresponds to $\mathrm{dk}(N)$ and thus it is impossible. In the remaining cases, it follows that $S \vdash \mathsf{vk}(N)$ (using vkof constructor on the signature). So $S \nvdash t_{bad}$ is impossible.

Case 9: "$t_{bad} = \mathsf{sk}(N)$ with $N \in \mathbf{N}_P$".

If $t_{bad} = \mathsf{sk}(N)$ then m_{bad} is the bitstring sk_N. Thus the simulator was able to compute sk_N with access only to signatures. By the strong existential unforgeability of the signature scheme, this can only happen with negligible probability.

Case 10: "$t_{bad} = \mathrm{dk}(N)$ with $N \in \mathbf{N}_P$".
If $t_{bad} = \mathrm{dk}(N)$ then m_{bad} is dk_N. So the simulator was able to compute dk_N with only access to an decryption oracle and the public key. By the CCA property, this can only occur with negligible probability.

In total, we get that $S \nvdash t_{bad}$ can only be the case with negligible probability.
Hence, $S \nvdash \tau(m)$ happens only with negligible probability. □

Indistinguishability

The next goal is to exclude extraction failures. First, we take a closer look at the relations and connect them to the functions β and τ. We defined the cryptographic conditions using img_η. In the following lemma we will see how this definition allies to the simulators' executions.

Lemma A.1.4 (Relating the relations)**.** *Let $R^{\mathrm{comp}}_{\mathrm{honest}}, R^{\mathrm{comp}}_{\mathrm{adv}}$ be relations implementing $R^{\mathrm{sym}}_{\mathrm{adv}}$ with usage restriction $R^{\mathrm{sym}}_{\mathrm{honest}}$.*

1. *In the hybrid execution of* Sim *and* Sim_3 *it holds with overwhelming probability: If* $(x, w) \in R^{\mathrm{sym}}_{\mathrm{honest}}$ *and* x, w *occur as node annotation of a ZK node in the execution, then it holds* $(\beta(x), \beta(w)) \in R^{\mathrm{comp}}_{\mathrm{honest}}$.

2. *In the hybrid execution of* Sim_2 *it holds with overwhelming probability: If* $(m_x, m_w) \in R^{\mathrm{comp}}_{\mathrm{adv}}$ *for some bitstrings* m_x, m_w*, then it holds* $(\tau(m_x), \tau^*(m_w)) \in R^{\mathrm{sym}}_{\mathrm{adv}}$.

Proof. We first define an environment η mapping terms to bitstrings. η depends on the current state of the execution. We will use η in both parts of the lemma. So let $t_1, \ldots, t_n$ be the terms sent by the protocol to the simulator so far.

For any term or subterm t that occurs as argument to β or output of τ, we define η as follows:

For $t = N^m$ define $\eta(t) := m$.

For $t = C(t_1, \ldots, t_n, N^m)$ define $\eta(t) := m$ for all C as stated in definition 2.4.1.[2]

For $t = \mathrm{crs}(N)$ with $N \in \mathbf{N}_P$ define $\eta(t)$ to be the crs produced by the oracle $\mathcal{O}^N_{\mathrm{ZK}}$.

For $t = \mathrm{ZK}(\mathrm{crs}(N), x, w, M)$ with $N, M \in \mathbf{N}_P$ define $\eta(t)$ to be proof produced by $\mathcal{O}^N_{\mathrm{ZK}}$ in the computation of $\beta(t)$.

For $t = N$ with $N \in \mathbf{N}_P$ we distinguish 2 cases. If t does neither occur in a term of the form $\mathrm{crs}(t)$ nor in $\mathrm{ZK}(c, x, w, t)$ for some c, x, w then define $\eta(t) := r_N$. Otherwise let $\eta(t)$ be undefined, i.e. $\eta(t) := \bot$.

Note that η is a consistent environment with overwhelming probability.

Most properties of consistency are satisfied by construction. The ZK case holds because of the indistinguishability of true proofs and their simulations. The only property that needs to be proven is the injectivity of η. We distinguish by the type of η's output.

- Type nonce. For $N, M \in \mathbf{N}_P$, a collision occurs with negligible probability, because $r_N = r_M$ occurs with negligible probability. The case $\eta(N^a) = \eta(N^b)$ for $a \neq b$ is even impossible. So consider the case $\eta(N) = \eta(M)$ for $N \in \mathbf{N}_P, M \in \mathbf{N}_E$.

  By protocol condition 2, it follows that M was output of τ, i.e. $M = N^n$ for some $n \in \{0,1\}^*$. First, let N be a nonce occurred inside $\mathrm{crs}(N)$. Then it holds $\eta(N) = \bot \neq n = \eta(N^n)$.

  Otherwise, if N was used before n was received by the simulator, then n would have been parsed to N by construction of τ. So the first occurrence of N has to be after n was received. But then the adversary guessed a nonce. This can only happen with negligible probability.

- Type decryption key. For the same reasons as in the case of type nonce we only consider the case $\eta(\mathrm{dk}(N)) = \eta(\mathrm{dk}(M))$ for $N \in \mathbf{N}_P, M \in \mathbf{N}_E$. By protocol condition 2, it follows that $\mathrm{dk}(M) = \mathrm{dk}(N^d)$ for some $d \in \{0,1\}^*$, $\mathrm{dk}(N^d)$ was subterm of an output

[2] They are $\{\mathrm{ek}, \mathrm{dk}, \mathsf{vk}, \mathsf{sk}, \mathsf{Enc}, \mathrm{sig}, \mathrm{crs}, \mathrm{garbageZK}, \mathrm{garbageSig}, \mathrm{garbageEnc}, \mathrm{garbage}\}$

of τ, and d was not output of β earlier (otherwise d would have been parsed to $\mathrm{dk}(N)$). So the adversary used either no input or only encryptions plus the encryption key to compute $\mathrm{dk}(N)$. By the CCA property, this can only be the case with negligible probability.

- Type signing key. This case is completely analogue to the decryption key type using the strongly existentially unforgeability instead of the CCA property.

- Type encryption key. As in the previous cases we can only need to consider $\eta(\mathrm{ek}(N)) = \eta(\mathrm{ek}(M))$ for $N \in \mathbf{N}_P, M \in \mathbf{N}_E$. By protocol condition 2, it follows that $\mathrm{ek}(M) = \mathrm{ek}(N^e)$ for some $e \in \{0,1\}^*$. But then τ parsed e to $\mathrm{ek}(N^e)$, so neither $\mathrm{ek}(N)$ nor $\mathrm{dk}(N)$ was used. This means the adversary guessed an encryption key without having any information about it. This can only happen with negligible probability.

- Type verification key and common reference string. Analogue to the case of encryption key.

- Type zero-knowledge proof. Because τ is deterministic, the adversary can not generate two different zero-knowledge proofs which are mapped to the same bitstring. So if there is a collision, then between a protocol generated proof and a adversary generated one.

- Type ciphertext and signature. Analogue to the case of zero-knowledge proofs.

- Type pair. If there is a collision of two pairs, then there is a collision in the first argument and in the second. So by induction hypothesis this case occurs with negligible probability.

- Type payload-string. This type does not contain any nonces. So applying η to a term of this type leads to a unique bitstring which cannot be hit by any other term of this type (by implementation condition 17).

- No type. The only term which has no type is $\mathrm{garbage}(t)$ for $t \in \mathbf{T}$. By protocol condition 2 and construction of τ, it has to hold that $t = N^m$ for some $m \in \{0,1\}^*$.

Proof of part 1 *of the lemma.*
By Definition 2.4.2[3], it suffices to show that if $(x, w) \in R^{\mathrm{sym}}_{\mathrm{honest}}$ then there is a consistent $\eta \in \mathcal{E}$ such that $(\mathrm{img}_\eta(x), \mathrm{img}_\eta(w)) = (\beta(x), \beta(w))$ since $\beta(x) \neq \bot \neq \beta(w)$. We show that the η defined above satisfies this criterium. Here, we prove the case for Sim_3. The proof for Sim is analogous with the only difference in the cases of ZK and crs. Here, the definition of η is done as for Enc and ek and the proof, as well.

For any term t that can occur in the execution of Sim_3 as annotation of a ZK node's statement or witness, we show that $\mathrm{img}_\eta(t) = \beta(t)$. This will be done by structural induction on the term t:

[3]The part we will use here says $(x, w) \in R^{\mathrm{sym}}_{\mathrm{honest}}$ and $\mathrm{img}_\eta(x) \neq \bot \neq \mathrm{img}_\eta(w)$ implies $(\mathrm{img}_\eta(x), \mathrm{img}_\eta(w)) \in R^{\mathrm{comp}}_{\mathrm{honest}}$.

- "$t = N$ with $N \in \mathbf{N}_P$".In this case $\beta(N) = r_N$. The nonce N may not occur as last argument of ZK or crs and inside x or w (protocol conditions 3 and 8). So N did not occur as last argument of ZK nor as argument of crs. Thus, it holds $\mathrm{img}_\eta(N) = \eta(N) = r_N$ by definition of η.

- "$t = N$ with $N \in \mathbf{N}_E$".In this case $N = N^n$ for some $n \in \{0,1\}^*$. Thus $\eta(N^n) = n = \beta(N^n)$.

- "$t \in \{\mathrm{ek}(u), \mathrm{dk}(u), \mathsf{vk}(u), \mathsf{sk}(u)\}$ with $u \in \mathbf{T}$".In this case, it holds that $u \in \mathbf{N}$. If $u \in \mathbf{N}_E$, i.e. $u = N^c$ for some $c \in \{0,1\}^*$, then $\beta(t) = c = \eta(t) = \mathrm{img}_\eta(t)$ by construction. So, consider $u \in \mathbf{N}_P$. Let $C \in \{\mathrm{ek}, \mathrm{dk}, \mathsf{vk}, \mathsf{sk}\}$ be the constructor such that $t = C(u)$. Then $\mathrm{img}_\eta(t) = A_C(\mathrm{img}_\eta(u)) \overset{(*)}{=} A_C(\beta(u)) = \beta(t)$. Since $u \in \mathbf{N}_P$ and occurs in $C(u)$, it follows that u does neither occur in $\mathrm{crs}(u)$ nor in $\mathrm{ZK}(c, x, w, u)$ for $c, x, w \in \mathbf{T}$ (protocol conditions forbid that these nonces are used more than once). Thus $\mathrm{img}_\eta(u) = r_u = \beta(u)$. Hence equality $(*)$ holds.

- "$t = \mathrm{crs}(N)$ with $N \in \mathbf{N}_P$".By definition $\beta(t)$ produces the crs using $\mathcal{O}^N_{\mathrm{ZK}}$ and $\mathrm{img}_\eta(\mathrm{crs}(N)) = \eta(\mathrm{crs}(N))$ which was defined as $\beta(t)$. Thus, it holds $\mathrm{img}_\eta(t) = \beta(t)$.

- "$t = \mathrm{crs}(N)$ with $N \in \mathbf{N}_E$".This case is analogue to the case $\mathrm{ek}(N)$ with $N \in \mathbf{N}_E$.

- "$t = \mathsf{Enc}(u_1, u_2, u_3)$".If $u_3 \in \mathbf{N}_E$, then this case is analogue to the case $t = \mathrm{ek}(u)$. So let $N := u_3 \in \mathbf{N}_P$. Then

  $$\beta(t) = A_{\mathsf{Enc}}(\beta(u_1), \beta(u_2), r_N) = A_{\mathsf{Enc}}(\mathrm{img}_\eta(u_1), \mathrm{img}_\eta(u_2), r_N)$$

  by induction hypothesis. The nonce N may only occur inside this encryption and as witness of the ZK-proof (protocol condition 3). Thus, by $r_N = \eta(N) = \mathrm{img}_\eta(N)$, it follows $\beta(t) = A_{\mathsf{Enc}}(\mathrm{img}_\eta(u_1), \mathrm{img}_\eta(u_2), \mathrm{img}_\eta(N)) = \mathrm{img}_\eta(t)$.

- "$t = \mathrm{sig}(u_1, u_2, u_3)$".If $u_3 \in \mathbf{N}_E$, then this case is analogue to the case $t = \mathrm{ek}(u)$. So let $N := u_3 \in \mathbf{N}_P$. By definition of τ, it follows that t was honestly generated. This means there was a sig-computation node that produced t. By protocol condition 7 this node is annotated by an sk-node. Since the protocol only uses its randomness (protocol condition 1), it follows that $u_1 = \mathsf{sk}(M)$ for some $M \in \mathbf{N}_P$. Then, it holds $\beta(t) = A_{\mathrm{sig}}(A_{\mathsf{sk}}(r_M), \beta(u_2), r_N)$. Again, $r_N = \mathrm{img}_\eta(N)$; the same holds for M. Since $\beta(\mathsf{sk}(M)) = A_{\mathsf{sk}}(M)$, it follows by induction hypothesis that $A_{\mathsf{sk}}(r_M) = \mathrm{img}_\eta(\mathsf{sk}(M))$. In total, it holds $\beta(t) = A_{\mathrm{sig}}(\mathrm{img}_\eta(\mathsf{sk}(M)), \mathrm{img}_\eta(u_2), \mathrm{img}_\eta(N)) = \mathrm{img}_\eta(t)$.

- "$t = \mathrm{pair}(u_1, u_2)$ where $u_1, u_2 \in \mathbf{T}$". By induction hypothesis, it follows $\beta(u_i) = \mathrm{img}_\eta(u_i)$. Thus, it holds

  $$\begin{aligned}\beta(\mathrm{pair}(u_1, u_2)) &= A_{\mathrm{pair}}(\beta(u_1), \beta(u_2)) \\ &= A_{\mathrm{pair}}(\mathrm{img}_\eta(u_1), \mathrm{img}_\eta(u_2)) = \mathrm{img}_\eta(\mathrm{pair}(u_1, u_2))\end{aligned}$$

- "$t \in \{\text{string}_0(u), \text{string}_1(u), \text{empty}\}$ with $u \in \mathbf{T}$".These cases are analogue to the case $t = \text{pair}(u_1, u_2)$.
- "$t \in \{\text{garbage}(u_1), \text{garbageSig}(u_1, u_2, u_3), \text{garbageEnc}(u_1, u_2), \text{garbageZK}(u_1, u_2, u_3, u_4)\}$ where $u_i \in \mathbf{T}$".By protocol condition 2 follows that t was generated by τ, i.e. the last argument of t has the form N^m for some $m \in \{0,1\}^*$. By definition of β, it holds that $\beta(t) = m$. On the other hand, by definition of img_η, it holds $\text{img}_\eta(t) = \eta(t) = m$, as well.

Proof of part 2 *of the lemma.*
It suffices to show that for each $m \in \{0,1\}^*$, that occurs with non-negligible probability in a hybrid execution of Sim_2, there is some η such that $m = \text{img}_\eta(\tau(m))$ holds. Then it follows by definition 2.4.2[4] $(m_x, m_w) \in R_{\text{adv}}^{\text{comp}} \implies (\text{img}_\eta(\tau(m_x)), \text{img}_\eta(\tau(m_w))) \in R_{\text{adv}}^{\text{comp}} \implies (\tau(m_x), \tau^*(m_w)) \in R_{\text{adv}}^{\text{sym}}$.

Take the same definition of η as in the case before. Note that this definition is canonical for an execution and does not depend on the term $\tau(m)$.

We will prove $m = \text{img}_\eta(\tau(m))$ by structural induction. Note that this suffices for τ^*, as well, since all cases for τ^* occur in τ.

- $\tau(m) = N$ for some $N \in \mathbf{N}_P$

  By construction of τ, it follows that $N \in \mathcal{N}$. Thus N was not argument of a crs or the last argument of a ZK node, by protocol conditions 1 and 3. Then $\text{img}_\eta(\tau(m)) = r_N = m$ where the last equality holds because of the definition of τ.

- $\tau(m) = N^m$

  Then by construction of η holds that $\text{img}_\eta(\tau(m)) = \text{img}_\eta(N^m) = \eta(N^m) = m$.

- $\tau(m) = \mathsf{Enc}(\text{ek}(M), t, N)$ for some $M \in \mathbf{N}, N \in \mathbf{N}_P$

  By definition of η holds that

  $$\text{img}_\eta(\tau(m)) = \text{img}_\eta(\mathsf{Enc}(\text{ek}(M), t, N)) = A_{\mathsf{Enc}}(A_{\text{ek}}(\eta(M)), \text{img}_\eta(t), \eta(N))$$

  By definition of τ follows that m was earlier output by β and thus evaluating t again using img_η gives the same bitstring m_t, r_N is the same argument as in the earlier call and $\text{ek}(M)$ is the same, too. By determinism of the implementations (implementation condition 1) follows that the output is m.

- $\tau(s) = \text{sig}(\mathsf{sk}(M), t, N)$ for some $M, N \in \mathbf{N}_P$

  This case is analogue to the one of $\mathsf{Enc}(\text{ek}(M), t, N)$ for some $M \in \mathbf{N}, N \in \mathbf{N}_P$.

- $\tau(m) = \text{ek}(N)$ for some $N \in \mathbf{N}_P$

  By definition of τ, it follows $m = A_{\text{ek}}(r_N)$. On the other hand, it holds that $\text{img}_\eta(\text{ek}(N)) = A_{\text{ek}}(\eta(N)) = A_{\text{ek}}(r_N) = m$ by construction.

[4] At this point we use $(\text{img}_\eta(x), \text{img}_\eta(w)) \in R_{\text{adv}}^{\text{comp}}$ implies $(x, w) \in R_{\text{adv}}^{\text{sym}}$.

- $\tau(m) \in \{ \mathsf{vk}(N), \mathsf{sk}(N), \mathrm{dk}(N)\}$ for some $N \in \mathbf{N}_P$

  The same as the case of $\mathrm{ek}(N)$.

- $\tau(m) = \mathrm{crs}(N)$ for some $N \in \mathbf{N}_P$

  By definition of τ follows that m was output after a call of β on $\mathrm{crs}(N)$. Thus m was output by the oracle and $\eta(N)$ is by definition the randomness used by the oracle to construct m. Thus $\mathrm{img}_\eta(\mathrm{crs}(N)) = A_{\mathrm{crs}}(\eta(N)) = m$ where the last equality holds because of the definition of $\eta(N)$.

- $\tau(m) = \mathrm{ZK}(\mathrm{crs}(N_1), t_1, t_2, N_2)$ for some $N_1, N_2 \in \mathbf{N}_P$

  By definition of η follows that

  $$\mathrm{img}_\eta(\mathrm{ZK}(\mathrm{crs}(N_1), t_1, t_2, N_2)) = \eta(\mathrm{ZK}(\mathrm{crs}(N_1), t_1, t_2, N_2)) = m$$

- $\tau(m) = \mathrm{pair}(t_1, t_2)$

  This case follows by the induction hypothesis and the determinism of the implementations.

- $\tau(m) \in \{\mathrm{string}_0(t_1), \mathrm{string}_1(t_1), \mathrm{empty}\}$

  The case of empty is trivial, since the implementation is deterministic. For the other cases holds that - by definition of τ - $t_1 = \tau(m')$ having $m' = A_{\mathrm{unstring_i}}(m)$ where $i \in \{0, 1\}$ and $\tau(m) = \mathrm{string_i}(t_1)$. Applying the induction hypothesis to t_1 leads to $\mathrm{img}_\eta(t_1) = m'$ and thus $\mathrm{img}_\eta(\tau(m)) = A_{\mathrm{string_i}}(\mathrm{img}_\eta(\tau(m'))) = A_{\mathrm{string_i}}(m') = A_{\mathrm{string_i}}(A_{\mathrm{unstring_i}}(m)) = m$. Here the last equality holds by implementation condition 17.

- $\tau(m) \in$

  $$\begin{aligned}\{&\mathsf{Enc}(\mathrm{ek}(M), t, N^m),\ \mathrm{ek}(N^m),\ \mathrm{dk}(N^m),\ \mathrm{garbageEnc}(t, N^m),\\ &\mathrm{sig}(\mathsf{sk}(M), t, N^m),\ \mathrm{garbageSig}(t, N^m),\ \mathsf{vk}(N^m),\ \mathsf{sk}(N^m),\\ &\mathrm{crs}(N^m),\ \mathrm{ZK}(\mathrm{crs}(M), x, w, N^m),\ \mathrm{garbageZK}(c, x, N^m),\ \mathrm{garbage}(N^m)\}\end{aligned}$$

  for some $M \in \mathbf{N}_P$

  All of these cases follow immediately by definition of η and definition 2.4.1.

The proof for Sim_2 is the same. Remind, the only difference between Sim_3 and Sim_2 is that Sim_3 does not check if $(x, w) \in R^{\mathrm{comp}}_{\mathrm{honest}}$ any more.

□

In the next step, we will show that ZK-breaks almost never occur in the execution of the simulator Sim_2. It is more convenient to show this for Sim and transfer it to Sim instead of showing it for Sim directly.

Lemma 2.4.6 (No ZK-breaks): In the hybrid execution of the simulator Sim_2, ZK-breaks occur only with negligible probability. ◇

Proof. We have to show that the case $(x, w) \notin R_{\text{adv}}^{\text{sym}}$ occurs with negligible probability. We do this by a case distinction on $(m_x, m_w) \notin R_{\text{adv}}^{\text{comp}}$.

First, consider that it holds $(m_x, m_w) \notin R_{\text{adv}}^{\text{comp}}$. The hybrid execution of Sim_2 is a valid adversary for the honest simulation-sound extractability game, because Sim_2 only sends a $(prove, x, w)$ query to $\mathcal{O}_{\text{sim}}$ if $(x, w) \in R_{\text{honest}}^{\text{comp}}$. In this case, the protocol sends a proof z such that an extraction-failure happens and the extraction extracts a m_w such that $(m_x, m_w) \notin R_{\text{adv}}^{\text{comp}}$ (where $m_x = A_{\text{getPub}}(z)$). Thus the adversary wins the honest simulation-sound extractability game which can only happen with negligible probability. Thus the case $(m_x, m_w) \notin R_{\text{adv}}^{\text{comp}}$ occurs with negligible probability. Therefore, in this case, it does not matter if $(x, w) \in R_{\text{adv}}^{\text{sym}}$ or not.

Now, consider the case that $(m_x, m_w) \in R_{\text{adv}}^{\text{comp}}$ holds. By Theorem A.1.4, it follows that $(\tau(m_x), \tau^*(m_w)) \in R_{\text{adv}}^{\text{sym}}$ with overwhelming probability. Since $x = \tau(m_x)$ and $w = \tau^*(m_w)$, it follows that $(x, w) \notin R_{\text{adv}}^{\text{sym}}$ can only occur with negligible probability.

Therefore a ZK-break in the execution of Sim_2 can only occur with negligible probability. □

Before we transfer the results to Sim, we prove a technical lemma that helps connecting Sim_2 and Sim_3. Especially, this lemma shows how it is possible to prove that protocol conditions, which are formulated for the symbolic execution, hold for the hybrid execution of some simulator, as well.

Lemma 2.4.4 (No invalid symbolic witnesses): Assume that Sim_3 is DY. Then, in the hybrid execution of Sim_3, for each ZK node with arguments t_1, t_2, t_3, t_4, it holds that $(t_2, t_3) \in R_{\text{honest}}^{\text{sym}}$ with overwhelming probability.

The same holds for Sim if Sim is DY. ◇

Proof. If Sim_3 is DY, then the hybrid execution of Sim_3 corresponds to a symbolic execution with overwhelming probability.

By definition of the hybrid execution, any hybrid execution is a valid symbolic execution, as long as the simulator does not send a term in the adversary's knowledge. Since Sim_3 is DY, this occurs only with negligible probability.

In the symbolic execution, the property $(t_2, t_3) \in R_{\text{honest}}^{\text{sym}}$ holds by protocol condition 10. Thus in the case that the hybrid execution corresponds to a symbolic one, it follows that $(t_2, t_3) \in R_{\text{honest}}^{\text{sym}}$ with overwhelming probability.

The same proof shows the statement for Sim. □

Now, we will formalize the connection of the simulators and transfer the results we have proven before. Thus, we achieve the following results: First, we show that ZK-breaks transfer to Sim_f. Second, we show that all simulators are DY-style. Thus, we may use all protocol conditions in all simulators, as we have shown in Lemma 2.4.4. Finally, we show that the node traces of all simulators are indistinguishable.

Lemma 2.4.3 (Preservation of simulator-properties):

(i) Let P_2 and P_f denote the probability of a ZK-break in the hybrid execution of the simulator Sim_2 and Sim_f, respectively. Then $|P_2 - P_f|$ is negligible.

(ii) Let P and P_f denote the probability of extraction failures in the hybrid execution of the simulator Sim and Sim_f, respectively. Then $|P - P_f|$ is negligible.

(iii) The simulator Sim is Dolev-Yao style if and only if the simulator Sim_f is.

◇

Proof. For $x \in \{1, \ldots, 5, f\}$ or x being the empty word, let $\mathrm{ZKBreak}_x$ denote the event that in the hybrid execution of the simulator Sim_x, a ZK-break occurs. The same way, we denote the event that a simulator Sim_x is DY in that execution by DY_x. We abbreviate $\text{H-Nodes}_{\mathbf{M},\Pi_p,\mathrm{Sim}_x}(k)$ as H-Nodes_x.

To show the lemma, we will show that

$$(\mathrm{DY}, \text{H-Nodes}) \overset{C}{\approx} (\mathrm{DY}_1, \text{H-Nodes}_1) \tag{A.1}$$

$$(\mathrm{DY}_1, \text{H-Nodes}_1) \overset{C}{\approx} (\mathrm{DY}_2, \text{H-Nodes}_2) \tag{A.2}$$

$$(\mathrm{ZKBreak}_2, \mathrm{DY}_2, \text{H-Nodes}_2) \overset{C}{\approx} (\mathrm{ZKBreak}_3, \mathrm{DY}_3, \text{H-Nodes}_3) \tag{A.3}$$

$$(\mathrm{ZKBreak}_3, \mathrm{DY}_3, \text{H-Nodes}_3) \overset{C}{\approx} (\mathrm{ZKBreak}_4, \mathrm{DY}_4, \text{H-Nodes}_4) \tag{A.4}$$

$$(\mathrm{ZKBreak}_4, \mathrm{DY}_4, \text{H-Nodes}_4) \overset{C}{\approx} (\mathrm{ZKBreak}_5, \mathrm{DY}_5, \text{H-Nodes}_5) \tag{A.5}$$

$$(\mathrm{ZKBreak}_5, \mathrm{DY}_5, \text{H-Nodes}_5) \overset{C}{\approx} (\mathrm{ZKBreak}_f, \mathrm{DY}_f, \text{H-Nodes}_f) \tag{A.6}$$

This will then imply statements (i)-(iii) from the lemma. It is obvious that Dolev-Yao-ness and ZK-breaks transfer as stated in the lemma by transitivity. But the extraction failures transfer because the in the presence of an extraction failure each simulator immediately stops. Thus if the extraction failures would not transfer as stated above, it would be possible to differentiate the node traces by their length.

We will show $(\mathrm{ZKBreak}_2, \mathrm{DY}_2, \text{H-Nodes}_2) \overset{C}{\approx} (\mathrm{ZKBreak}_3, \mathrm{DY}_3, \text{H-Nodes}_3)$ at the end, because we need the intermediate result to prove it.

- $(\mathrm{DY}, \text{H-Nodes}) \overset{C}{\approx} (\mathrm{DY}_1, \text{H-Nodes}_1)$

  Transforming Sim to Sim_1 is done by replacing invocations of the ZK algorithms by oracle-queries. We can replace $A_{\mathrm{crs}}(r_N)$ by a $(\mathtt{crs})$-query to $\mathcal{O}^N_{\mathrm{ZK}}$ because N is only used inside this crs (protocol condition 8) and the distributions of the implementation and the oracle are the same. Since $\tau(c)$ in Sim_1 not checks whether $c = A_{\mathrm{crs}}(r_N)$ but whether c is the result of some $(\mathtt{crs})$-query, the node traces have the same distribution.

  The same holds for the replacement of A_{ZK} by the $(prove, x, w)$ oracle query to $\mathcal{O}^N_{\mathrm{ZK}}$. The randomness – the fourth argument of the ZK proof – only occurs inside this proof and nowhere else (protocol condition 3), so we can replace it by the oracle's randomness as in the crs case.

By implementation condition 26, it holds that if $(x, w) \notin R^{\text{comp}}_{\text{honest}}$ the implementation, as well as the oracle, output $\perp$. So A_{ZK} and the $(\texttt{prove}, x, w)$-query return $\perp$ in the same cases. In the case that $(x, w) \in R^{\text{comp}}_{\text{honest}}$ both compute a proof of x using witness w. Thus, it holds $(\text{DY}, \text{H-Nodes}) \overset{C}{\approx} (\text{DY}_1, \text{H-Nodes}_1)$.

- $(\text{DY}_1, \text{H-Nodes}_1) \overset{C}{\approx} (\text{DY}_2, \text{H-Nodes}_2)$

  In this step we replace $\mathcal{O}_{\text{ZK}}$ by $\mathcal{O}_{\text{Sim}}$ which returns a simulated proof for x if for the input (x, w) it holds that $(x, w) \in R^{\text{comp}}_{\text{honest}}$.

  If we change both simulators to not extract the proof in case of an extraction failure, then the H-Nodes does not change. The simulator stops after handling extraction failures in any case. By definition of zero-knowledge these two modified cases are indistinguishable (using the fact that the simulator and prover are only invoked if $(x, w) \in R^{\text{comp}}_{\text{honest}}$). Thus $(\text{DY}_1, \text{H-Nodes}_1)$ and $(\text{DY}_2, \text{H-Nodes}_2)$ are indistinguishable, too.

- $(\text{ZKBreak}_3, \text{DY}_3, \text{H-Nodes}_3) \overset{C}{\approx} (\text{ZKBreak}_4, \text{DY}_4, \text{H-Nodes}_4)$

  In this step we replace encryptions, decryptions and key-generation by an encryption-oracle as we did for the zero-knowledge proofs in the step from Sim to Sim_1. Because Sim_3 does not compute witnesses of zero-knowledge proofs anymore, nonces of encryptions are only used once (by protocol condition 3). Nonces of keys were already only used once (by protocol condition 1). So replacing the implementation of encryptions, decryptions and the public key does not change the distribution of the node trace or ZK-Breaks (since we adapted τ accordingly, cf. the replacement of A_{crs} in Sim_1). In addition we can define $\beta(\text{dk}(N)) := \perp$ because decryption keys are not used as input to β (by protocol condition 4 and the use of an oracle for decrypting).

  We did neither change the bitstrings that are sent to the adversary nor the way they are parsed. So the property of DY did not change, either.

- $(\text{ZKBreak}_4, \text{DY}_4, \text{H-Nodes}_4) \overset{C}{\approx} (\text{ZKBreak}_5, \text{DY}_5, \text{H-Nodes}_5)$

  In the step from Sim_4 to Sim_5, the only change that is done is the replacement of the encryption oracle by a fake oracle that always encrypts $0^{|m|}$ instead of m. By construction of τ the protocol execution asks only for decryptions of ciphertexts which were not generated by the encryption oracle (since only β invokes the encryption oracle). So a run of the protocol is a valid adversary for the CCA property where the challenger is the encryption oracle. To get indistinguishability the adversary has to be able to use ZK-breaks, DYness and node traces to distinguish both executions. Obviously, it is possible to use DYness and the node traces. For the case of ZK-breaks, we have to require that $R^{\text{sym}}_{\text{adv}}$ is efficiently decidable.

  Thus replacing ENC by $\mathcal{O}_{\text{fake}}$ leads to an indistinguishable execution and hence $(\text{ZKBreak}_4, \text{DY}_4, \text{H-Nodes}_4)$ and $(\text{ZKBreak}_5, \text{DY}_5, \text{H-Nodes}_5)$ are computationally indistinguishable.

- $(\mathrm{ZKBreak}_5, \mathrm{DY}_5, \text{H-Nodes}_5) \overset{C}{\approx} (\mathrm{ZKBreak}_f, \mathrm{DY}_f, \text{H-Nodes}_f)$

  As in the case for Sim_3 and Sim_4, we have the case that after removing the witnesses in Sim_3 the nonces, used as randomness for signatures, are only used (by protocol condition 3) once for signing a message.

  The same holds for verification and signing keys (by protocol condition 1). Thus we can replace signing and computation of verification/signing keys by invocations of $\mathcal{O}_{\text{sig}}$ without changing the distribution of $(\mathrm{ZKBreak}, \mathrm{DY}, \text{H-Nodes})$ (since we adopted τ accordingly, cf. the replacement of A_{crs} in Sim_1). In a run of the protocol β is never applied to $\mathsf{sk}(N)$ (by protocol condition 5 and the use of an oracle for signing), so we can define $\beta(\mathsf{sk}(N)) := \bot$ without changing the distribution of $(\mathrm{ZKBreak}, \mathrm{DY}, \text{H-Nodes})$.

- $(\mathrm{ZKBreak}_2, \mathrm{DY}_2, \text{H-Nodes}_2) \overset{C}{\approx} (\mathrm{ZKBreak}_3, \mathrm{DY}_3, \text{H-Nodes}_3)$

  We have already proven that $(\mathrm{DY}_{\mathrm{f}}) \overset{C}{\approx} (\mathrm{DY}_3)$. Together with the fact that Sim_f is DY (Lemma A.1.3), it follows that Sim_3 is DY. By Lemma 2.4.4, it follows that $(t_2, t_3) \in R^{\text{sym}}_{\text{honest}}$ for all ZK-nodes with arguments $t_1, \ldots, t_4$ in a hybrid execution of Sim_3 (with overwhelming probability). Applying Lemma A.1.4 leads to $(\beta(t_2), \beta(t_3)) \in R^{\text{comp}}_{\text{honest}}$ for a hybrid execution of Sim_3 with overwhelming probability. The only difference between Sim_2 and Sim_3 is that Sim_2 checks whether $(\beta(t_2), \beta(t_3)) \in R^{\text{comp}}_{\text{honest}}$. Because this check would succeed with overwhelming probability in Sim_3, it actually succeeds in Sim_2.

  Thus the distribution of $(\mathrm{ZKBreak}, \mathrm{DY}, \text{H-Nodes})$ is the same in Sim_2 as in Sim_3.

□

Using the preceding lemma together with the generalized DY lemma, it is easy to prove that extraction failures during Sim_f's execution occur with negligible probability. Thus by the preceding lemma, it follows that the same holds for Sim.

Lemma 2.4.8 (No extraction failures): In a hybrid execution of the simulator Sim_f holds: An extraction failure can only occur with negligible probability. ◇

Proof. Assume an extraction-failure occurs with non-negligible probability. Then $\tau(z)$ is called for a bitstring z of type zero-knowledge proof such that the symbolic extraction fails.

So z was not generated by the protocol, i.e. it was not output of the simulation oracle, and the corresponding crs was generated by the protocol (otherwise τ would not invoke the symbolic extraction). Let $N \in \mathbf{N}_P$ be defined by $\mathrm{crs}(N) = \tau(A_{\text{crs}}(z))$. Let $m_x := A_{\text{getPub}}(z)$, $x := \tau(m_x)$, $m_w := \mathbf{E}(m_x, z, \mathrm{extd}_N))$,[5] $w := \tau(m_w)$. Let S denote the set of $\mathbf{T}$ that the protocol already sent to the simulator in this execution.

We have $\mathbf{SymbExtr}(S, x) = \bot$ by definition of extraction failures. Thus one of the following cases occurs with non-negligible probability.

1. $(x, w) \notin R^{\text{sym}}_{\text{adv}}$

[5] Here, extd_N is the extraction trapdoor that the simulator receives from the oracle $\mathcal{O}^N_{\text{ZK}}$ by querying (`extd`).

2. $S \nvdash w$

3. $(x, w) \in R_{\text{adv}}^{\text{sym}}$ and $S \vdash w$ and $\mathbf{SymbExtr}(S, x) = \bot$

We prove for each case that it occurs with negligible probability leading to a contradiction to the assumption that an extraction-failure occurs with non-negligible probability.

Case 1: "$(x, w) \notin R_{\text{adv}}^{\text{sym}}$"

This would be a ZK-break. Thus, this case only occurs with negligible probability because of Theorem 2.4.6 and Theorem 2.4.3 (i).

Case 2: "$S \nvdash w$"

By Lemma A.1.3 and since $w = \tau^*(m_w)$, this case can only occur with negligible probability.

Case 3: "$(x, w) \in R_{\text{adv}}^{\text{sym}}$ and $S \vdash w$ and $\mathbf{SymbExtr}(S, x) = \bot$"

By definition, $\mathbf{SymbExtr}$ returns only $\bot$ if there is no w such that $(x, w) \in R_{\text{adv}}^{\text{comp}}$ and $S \vdash w$. So this case cannot occur.

□

The only thing, that is missing to apply Theorem 2.1.1 is to show that Sim is indistinguishable from an computational execution.

Lemma A.1.5. Sim *is indistinguishable for* $\mathbf{M}, \Pi, A, E$ *and for every polynomial* p*.*

Proof. We will first show that when fixing the randomness of the adversary and the protocol, the node trace $\mathrm{Nodes}^p_{\mathbf{M},A,\Pi_p,E}$ in the computational execution and the node trace $\text{H-Trace}_{\mathbf{M},\Pi_p,\mathrm{Sim}}$ in the hybrid execution are equal. Hence, fix the variables r_N for all $N \in \mathbf{N}_P$, fix a random tape for the adversary, and for each non-deterministic node ν fix a choice e_ν of an outgoing edge.

We assume that the randomness is chosen such that all bitstrings r_N, $A_{\text{ek}}(r_N)$, $A_{\text{dk}}(r_N)$, $A_{\mathsf{vk}}(r_N)$, $A_{\mathsf{sk}}(r_N)$, $A_{\mathsf{Enc}}(e, m, r_N)$, $A_{\text{sig}}(s, m, r_N)$, $A_{\text{crs}}(r_N)$, and $A_{\text{ZK}}(c, x, w, r_N)$ are all pairwise distinct for all $N \in \mathcal{R}$ if they are well-formed.[6]

Note that this is the case with overwhelming probability: For terms of different types this follows from implementation condition 2. For keys, this follows from the fact that if two randomly chosen keys would be equal with non-negligible probability, the adversary could guess secret keys and thus break the IND-CCA property or the strong existential unforgeability (implementation conditions 8 and 9). For nonces, if two random nonces r_N, r_M would be equal with non-negligible probability, so would encryption keys $A_{\text{ek}}(r_N)$ and $A_{\text{ek}}(r_M)$. For encryptions, by implementation condition 6, the probability that $A_{\mathsf{Enc}}(e, m, r_N)$ for random r_N of type nonce matches any given string is negligible. Let e, e', m, m', r, r' be bitstrings. For

[6]That means that e is of type encryption key, s of type signing key, c of type common reference string and $x, w, m \in \{0, 1\}^*$ result from some evaluation of β in the execution.

$e \neq e'$, it holds that $A_{\mathsf{Enc}}(e, m, r_N) \neq A_{\mathsf{Enc}}(e', m', r'_N)$, because A_{ekof} returns in one case e and in the other e'. So if $A_{\mathsf{Enc}}(e, m, r_N) = A_{\mathsf{Enc}}(e', m', r'_N)$, it holds $e = e'$. Additionally, $m = m'$ because decryption, using e as argument, deterministically computes m. So the only case that can occur is $A_{\mathsf{Enc}}(e, m, r_N) = A_{\mathsf{Enc}}(e, m, r_{N'}$. Since by protocol condition 3, each $A_{\mathsf{Enc}}(e, m, r_N)$ computed by β uses a fresh nonce r_N, this case occurs with negligible probability. Analogously for signatures (implementation condition 7, protocol conditions 3 and 5) and for zero-knowledge proofs (implementation condition 20, protocol conditions 3 and 8).

Additionally, we assume that there is no extraction failure in the hybrid execution of Sim. By Lemma 2.4.8 extraction failures do not occur in the hybrid execution of Sim_f. Since the probability of an extraction failure in the hybrid execution of Sim and Sim_f differ only by a negligible function (Lemma 2.4.3 (ii)), extraction failures only occur with negligible probability in Sim. This is the only case in which the simulator aborts early.

In the following, we designate the values f_i and ν_i in the computational execution by f'_i and ν'_i, and in the hybrid execution by f_i^C and ν_i^C. Let s'_i denote the state of the adversary E in the computational model, and s_i^C the state of the simulated adversary in the hybrid model.

Claim 1: In the hybrid execution, for any $\forall m \in \{0,1\}^* : \beta(\tau(m)) = m$.

This claim follows by induction over the length of m and by distinguishing the cases in the definition of τ. A detailed proof is in the section A.1.1.

Claim 2: In the hybrid execution, for any term t stored at a node ν, $\beta(t) \neq \bot$.

By Definition of β, any term t with $\beta(t) = \bot$ has a subterm of the form $\mathrm{ZK}(t_1, t_2, t_3, t_4)$ with $t_4 \notin \mathbf{N}$, t_1 not of the form $\mathrm{crs}(N)$ with $N \in \mathbf{N}$, or $(\beta(t_2), \beta(t_3)) \notin R_{\text{adv}}^{\text{comp}}$ or a subterm with a similar form of encryption-, signature- or garbage-terms. These are never generated by τ nor by the protocol.

Claim 3: For all terms $t \notin \mathcal{R}$ that occur in the hybrid execution, $\tau(\beta(t)) = t$.

By induction on the structure of t and using the assumption that r_N, $A_{\text{ek}}(r_N)$, $A_{\text{dk}}(r_N)$, $A_{\mathsf{vk}}(r_N)$, $A_{\mathsf{sk}}(r_N)$, as well as all occurring encryptions and signatures are pairwise distinct for all $N \in \mathcal{R}$.

Claim 4: In the hybrid execution, at any computation node $\nu = \nu_i$ with constructor or destructor F and arguments $\bar{\nu}_1, \ldots, \bar{\nu}_n$ the following holds: Let t_j be the term stored at node $\bar{\nu}_j$ (i.e., $t_j = f'_i(\bar{\nu}_j)$). Then $\beta(\mathrm{eval}_F(\underline{t})) = A_F(\beta(t_1), \ldots, \beta(t_n))$. Here the left hand side is defined iff the right hand side is.

The proof for this claim is lengthy and thus postponed to the section A.1.2.

For given fixed randomness (see above), let ν'_i, f'_i be the nodes and functions as in the computational trace and ν_i^C, f_i^C be the ones in the hybrid trace. Let s'_i be the state of the adversary before execution of the i-th node and s_i^C the corresponding state of the adversary in the hybrid execution.

We will now show that $\mathrm{Nodes}^{p}_{\mathbf{M},A,\Pi_p,E} = \text{H-Nodes}_{\mathbf{M},\Pi_p,\mathrm{Sim}}(k)$.

To prove this, we show the following invariant: $f'_i = \beta \circ f^C_i$ and $\nu'_i = \nu^C_i$ and $s'_i = s^C_i$ for all $i \geq 0$ by induction on i

Base case $i = 0$. The adversary E is in its starting configuration,i.e. $s'_0 = s^C_0$, the node mapping function f is totally undefined, $f'_0 = f^C_0 =$ and the current node is the root of the protocol, $\nu'_0 = \nu^C_0$, so the invariant is satisfied for $i = 0$.

Induction hypothesis: For all $j \leq i$ holds $\nu_j = \nu^C_j$, $f_j = \beta \circ f^C_j$ and $s_j = s^C_j$.

Induction step: $i \to i + 1$:

We make a case distinction by the type of the nodes:

1. If $\nu'_i = \nu^C_i$ is a computation node annotated with constructor or destructor F, we have that $f'_{i+1}(\nu'_i) = A_F(f'_i(\bar{\nu}_1), \ldots, f'_i(\bar{\nu}_n)) = A_F(\beta(f^C_i(\bar{\nu}_1)), \ldots, \beta(f^C_i(\bar{\nu}_n)))$ for some nodes $\bar{\nu}_s$. And $f^C_{i+1}(\nu'_i) = f^C_{i+1}(\nu^C_i) = \mathrm{eval}_F(f^C_i(\bar{\nu}_1), \ldots, f^C_i(\bar{\nu}_n))$. From Claim 4 it follows that $\beta(f^C_{i+1}(\nu'_i)) = f'_{i+1}(\nu'_i)$ where the lhs is defined iff the rhs is. Hence $\beta \circ f^C_{i+1} = f'_{i+1}$.

   By Claim 2, $\beta(f^C_{i+1}(\nu^C_i))$ is defined if $f^C_{i+1}(\nu^C_i)$ is. Hence $f^C_{i+1}(\nu^C_i)$ is defined iff $f'_{i+1}(\nu'_i) = f'_{i+1}(\beta(f^C_{i+1}(\nu^C_i)))$ is. If $f^C_{i+1}(\nu^C_i)$ is defined, then ν^C_{i+1} is the yes-successor of ν^C_i and the no-successor otherwise. If $f'_{i+1}(\nu'_i)$ is defined, then ν'_{i+1} is the yes-successor of $\nu'_i = \nu^C_i$ and the no-successor otherwise. Thus $\nu^C_{i+1} = \nu'_{i+1}$.

   The adversary E is not invoked, hence $s'_{i+1} = s^C_{i+1}$. So the invariant holds for $i + 1$ if ν'_i is a computation node with a constructor or destructor.

2. If $\nu'_i = \nu^C_i$ is a computation node annotated with nonce $N \in \mathbf{N}_P$, we have that $f'_{i+1}(\nu'_i) = r_N = \beta(N) = \beta(f^C_{i+1}(\nu'_i))$. Hence $\beta \circ f^C_{i+1} = f'_{i+1}$. By Definition 2.1.6, ν'_{i+1} is the yes-successor of ν'_i. Since $N \in \mathbf{T}$, ν^C_{i+1} is the yes-successor of $\nu^C_i = \nu'_i$. Thus $\nu'_{i+1} = \nu^C_{i+1}$. The adversary E is not invoked, hence $s'_{i+1} = s^C_{i+1}$. So the invariant holds for $i + 1$ if ν'_i is a computation node with a nonce.

3. If $\nu'_i = \nu^C_i$ is an input node, the adversary E in the computational execution and the simulator in the hybrid execution is asked for a bitstring m' or bitstring t^C, respectively. The simulator produces this string by asking the simulated adversary E for a bitstring m^C and setting $t^C := \tau(m^C)$. Since $s'_i = s^C_i$, we have $m' = m^C$. Then by definition of the computational and hybrid executions, $f'_{i+1}(\nu'_i) = m'$ and $f^C_{i+1}(\nu'_i) = t^C = \tau(m')$. Thus $f'_{i+1}(\nu'_i) = m' \overset{(*)}{=} \beta(\tau(m')) = \beta(f^C_{i+1}(\nu'_i))$ where $(*)$ follows from Claim 1. Since $f'_{i+1} = f'_i$ and $f^C_{i+1} = f^C$ everywhere else, we have $f'_{i+1} = \beta \circ f^C_{i+1}$. Furthermore, since input nodes have only one successor, $\nu'_{i+1} = \nu^C_{i+1}$. Since we fixed the random choices of the execution, the adversaries state is $s'_i = s^C_i$, and since $m' = m^C$, it follows that $s'_{i+1} = s^C_{i+1}$. Thus the invariant holds for $i + 1$ in the case of an input node.

4. If $\nu'_i = \nu^C_i$ is an output node, the adversary E in the computational execution gets $m' := f'_i(\bar{\nu}_1)$ where the node $\bar{\nu}_1$ depends on the label of ν'_i. In the hybrid execution, the simulator gets $t^C := f^C_i(\bar{\nu}_1)$ and sends $m^C := \beta(t^C)$ to the simulated adversary E. By induction hypothesis we then have $m' = m^C$, so the adversary gets the same input in both

executions. Thus $s'_{i+1} = s^C_{i+1}$. Furthermore, since output nodes have only one successor, we have $\nu'_{i+1} = \nu^C_{i+1}$. And $f'_{i+1} = f'_i$ and $f^C_{i+1} = f^C_i$, so $f'_{i+1} = \beta \circ f^C_{i+1}$. Thus the invariant holds for $i+1$ in the case of an output node.

5. If $\nu'_i = \nu^C_i$ is a control node, the adversary E in the computational execution and the simulator in the hybrid execution get the out-metadata l of the node ν'_i or ν^C_i, respectively. The simulator passes l on to the simulated adversary. Thus, since $s'_i = s^C_i$, we have that $s'_{i+1} = s^C_{i+1}$, and in the computational and the hybrid execution, E answers with the same in-metadata l'. Thus $\nu'_{i+1} = \nu^C_{i+1}$. Since a control node does not modify f we have $f'_{i+1} = f'_i = \beta \circ f^C_i = \beta \circ f^C_{i+1}$. Hence the invariant holds for $i+1$ if ν'_i is a control node.

6. If $\nu'_i = \nu^C_i$ is a nondeterministic node, $\nu'_{i+1} = \nu^C_{i+1}$ is determined by $e_{\nu'_i} = e_{\nu^C_i}$. Since a nondeterministic node does not modify f and the adversary is not activated, $f'_{i+1} = f'_i = \beta \circ f^C_i = \beta \circ f^C_{i+1}$ and $s_{i+1} = s'_{i+1}$. Hence the invariant holds for $i+1$ if ν'_i is a nondeterministic node.

From the invariant it follows that the node trace is the same in both executions.

Since random choices with all nonces, keys, encryptions, and signatures being pairwise distinct occur with overwhelming probability (as discussed above), the node traces of the real and the hybrid execution are indistinguishable. □

Final Soundness Proof.. Having the preceding lemmas, we prove the computational soundness (Theorem 2.4.1).

Proof of Theorem 2.4.1. By lemma A.1.5 we get that Sim is indistinguishable for $\mathbf{M}, \Pi, A, E$ and for every polynomial p. By Lemma A.1.3, Sim_f is DY which transfers to Sim by lemma 2.4.3 (iii). So Sim is a good simulator. By Theorem 2.1.1 we finally conclude that the implementation A is sound for every protocol as specified in the Theorem. □

A.1.1 Proof of Claim 1

In this section we present a proof of claim 1 of the claims used in the indistinguishability proof. The proofs of all other claims are similar done by structural induction.

We want to show that - in the hybrid execution of Sim - holds $\forall m \in \{0,1\}^* : \beta(\tau(m)) = m$.

Proof. By structural induction on $\tau(c)$.

- $\tau(m) = N$ for some $N \in \mathbf{N}_P$

  Then $m = r_N$ and $\beta(\tau(m)) = \beta(N) = r_N = m$.

- $\tau(m) = N^m$

  Then $\beta(\tau(m)) = \beta(N^m) = m$

- $\tau(c) = \mathsf{Enc}(\mathrm{ek}(M), t, N)$ and c was output by $\beta(\mathsf{Enc}(M), t, N))$

  Then $\beta(\tau(c)) = \beta(\mathsf{Enc}(\mathrm{ek}(M), t, N)) = A_{\mathsf{Enc}}(\beta(\mathrm{ek}(M)), \beta(t), r_M)$. This is equal to c since the arguments are equal (randomness of Enc is the third argument) and by implementation condition 1 we know that A_{Enc} is deterministic.

- $\tau(c) = \mathsf{Enc}(\mathrm{ek}(M), t, N^c)$

  Then $\beta(\tau(c)) = \beta(\mathsf{Enc}(\mathrm{ek}(M), t, N^c) = c$.

- $\tau(c) = \mathrm{garbageEnc}(t, N^c)$

  Then $\beta(\tau(c)) = \beta(\mathrm{garbageEnc}(t, N^c) = c$.

- $\tau(c) = \mathrm{ek}(N)$ for some $N \in \mathbf{N}_P$.

  Then by definition of τ, it holds $c = A_{\mathrm{ek}}(r_N) = \beta(\mathrm{ek}(N)) = \beta(\tau(c))$.

- $\tau(c) = \mathrm{ek}(N^c)$.

  Then $\beta(\tau(c)) = \beta(\mathrm{ek}(N^c)) = c$.

- $\tau(c) = \mathrm{dk}(N)$ for some $N \in \mathbf{N}_P$.

  Then by definition of τ, it holds $c = A_{\mathrm{dk}}(r_N) = \beta(\mathrm{dk}(N)) = \beta(\tau(c))$.

- $\tau(c) = \mathrm{dk}(N^c)$.

  Then $\beta(\tau(c)) = \beta(\mathrm{dk}(N^c)) = c$.

- $\tau(c) = \mathrm{sig}(\mathsf{sk}(M), t, N)$ with $N, M \in \mathbf{N}_P$, earlier output by $\beta(\mathrm{sig}(\mathsf{sk}(M), t, N))$.

  Then holds $\beta(\tau(c)) = \beta(\mathrm{sig}(\mathsf{sk}(M), t, N)) = c$ As in the case of encryption we have the same arguments and the a deterministic function, so the result has to be c again.

- $\tau(c) = \mathrm{sig}(\mathsf{sk}(M), t, N^c)$

  Then holds $\beta(\tau(c)) = \beta(\mathrm{sig}(\mathsf{sk}(M), t, N^c)) = c$.

- $\tau(c) = \mathrm{garbageSig}(\mathsf{sk}(M), N^c)$

  Then holds $\beta(\tau(c)) = \beta(\mathrm{garbageSig}(\mathsf{sk}(M), N^c)) = c$.

- $\tau(c) = \mathsf{vk}(N)$

  Then by definition of τ, it holds $c = A_{\mathsf{vk}}(r_N) = \beta(\mathsf{vk}(N)) = \beta(\tau(c))$.

- $\tau(c) = \mathsf{vk}(N^c)$

  Then holds $\beta(\tau(c)) = \beta(\mathsf{vk}(N^c)) = c$.

- $\tau(c) = \mathsf{sk}(N)$

  Then by definition of τ, it holds $c = A_{\mathsf{sk}}(r_N) = \beta(\mathsf{sk}(N)) = \beta(\tau(c))$.

- $\tau(c) = \mathsf{sk}(N^c)$

  Then holds $\beta(\tau(c)) = \beta(\mathsf{sk}(N^c)) = c$.

- $\tau(c) = \mathrm{ZK}(\mathrm{crs}(t_1), t_2, t_3, N)$ with $N \in \mathbf{N}_P$, earlier output by $\beta(\mathrm{ZK}(\mathrm{crs}(t_1), t_2, t_3, N))$.

  This case holds because of the determinism of the implementation A_{ZK}.

- $\tau(c) = \mathrm{ZK}(\mathrm{crs}(t_1), t_2, t_3, N^c)$

  Then holds $\beta(\tau(c)) = \beta(\mathrm{ZK}(\mathrm{crs}(t_1), t_2, t_3, N^c)) = c$ by definition of β.

- $\tau(c) = \mathrm{crs}(N)$ for $N \in \mathbf{N}_P$, earlier been output by $\beta(\mathrm{crs}(N))$

  Then holds by determinism of A_{crs} that $\beta(\tau(c)) = \beta(\mathrm{crs}(N)) = A_{\mathrm{crs}}(r_N) = c$.

- $\tau(c) = \mathrm{crs}(N^c)$

  Then holds $\beta(\tau(c)) = \beta(\mathrm{crs}(N^c)) = c$.

- $\tau(c) = \mathrm{garbageZK}(t_1, t_2, N^c)$

  Then holds $\beta(\tau(c)) = \beta(\mathrm{garbageZK}(t_1, t_2, N^c)) = c$

- $\tau(c) = \mathrm{pair}(t_1, t_2)$

  By construction of τ follows that $t_1 = \tau(A_{\mathrm{fst}}(c))$ and $t_2 = \tau(A_{\mathrm{snd}}(c))$. By induction hypothesis follows for $c_1 \coloneqq A_{\mathrm{fst}}(c)$ that $\beta(\tau(c_1)) = c_1$. The same holds for $c_2 \coloneqq A_{\mathrm{snd}}(c)$. Therefore we get

  $$\begin{aligned}\beta(\mathrm{pair}(t_1, t_2)) &= A_{\mathrm{pair}}(\beta(t_1), \beta(t_2)) = A_{\mathrm{pair}}(\beta(\tau(A_{\mathrm{fst}}(c))), \beta(\tau(A_{\mathrm{snd}}(c))))\\ &= A_{\mathrm{pair}}(A_{\mathrm{fst}}(c), A_{\mathrm{snd}}(c)) = c\end{aligned}$$

  where the last equality holds because of implementation conditions 11 and 1.

- $\tau(c) = \mathrm{string}_0(t)$

  By definition of τ follows that $t = \tau(c')$ with $c' = A_{\mathrm{unstring}_0}(c)$ and $c' \neq \bot$. The induction hypothesis implies that $\beta(\tau(c')) = c'$. So $\tau(\mathrm{string}_0(t)) = A_{\mathrm{string}_0}(\beta(t)) = A_{\mathrm{string}_0}(c') = A_{\mathrm{string}_0}(A_{\mathrm{unstring}_0}(c)) = c$ The last equality holds because of implementation conditions 17 and 1.

- $\tau(c) = \mathrm{string}_1(t)$

  Analogue to the case of $\tau(c) = \mathrm{string}_0(t)$.

- $\tau(c) = \mathrm{empty}$

  Then $c = A_{\mathrm{empty}}() = \beta(\mathrm{empty}) = \beta(\tau(c))$.

- $\tau(c) = \mathrm{garbage}(N^c)$

  Then $\beta(\mathrm{garbage}(N^c)) = c$.

□

A.1.2 Proof of Claim 4

Proof. The proof is done by induction on the trace length with a case distinction on all constructors and destructors F.

1. "$F = \mathrm{crs}$"

   By protocol condition 1 the first argument of this node is a nonce computation node, i.e. $t_1 = N$ for some $N \in \mathbf{N}_P$. Therefore holds $\beta(\mathrm{eval}_{\mathrm{crs}}(t_1)) = \beta(\mathrm{crs}(t_1)) = A_{\mathrm{crs}}(r_N) = A_{\mathrm{crs}}(\beta(N))$.

2. "$F \in \{\mathrm{ek}, \mathrm{dk}, \mathsf{vk}, \mathsf{sk}\}$"

   Analogous to the case $F = \mathrm{crs}$.

3. "$F = \mathrm{ZK}$"

   A node annotated with ZK has as $t_1 = \mathrm{crs}(N_1)$ for some $N_1 \in \mathbf{N}_P$ and $t_4 = N_2$ for $N_2 \in \mathbf{N}_P$ (protocol conditions 8 and 3). Thus, we have:

   $$\begin{aligned} A_{\mathrm{ZK}}(\beta(t_1), \beta(t_2), \beta(t_3), \beta(t_4)) &= A_{\mathrm{ZK}}(A_{\mathrm{crs}}(r_{N_1}), \beta(t_2), \beta(t_3), r_{N_2}) \\ &= \beta(\mathrm{eval}_{\mathrm{ZK}}(t_1, t_2, t_3, t_4)) \end{aligned}$$

4. "$F = \mathrm{getPub}$"

   If the argument t is neither of the form $\mathrm{ZK}(t_1, t_2, t_3, t_4)$ nor $\mathrm{garbageZK}(t_1, t_2, t_3)$ then $\beta(\mathrm{getPub}(t)) = \bot$ and $A_{\mathrm{getPub}}(\beta(t)) = \bot$, too. So first consider $t = \mathrm{ZK}(t_1, t_2, t_3, t_4)$. Then, by protocol conditions 3 and 8 and by definition of τ, it follows that t_1 has the form $\mathrm{crs}(u_1)$ with $u_1 \in \mathbf{N}$ and $t_4 \in \mathbf{N}$.

   Thus we have $\beta(\mathrm{eval}_{\mathrm{getPub}}(\mathrm{ZK}(\mathrm{crs}(u_1), t_2, t_3, t_4))) = \beta(t_2)$.

   Case 1: $t_4 = N \in \mathbf{N}_P$

   Then it holds that $A_{\mathrm{getPub}}(\beta(\mathrm{ZK}(\mathrm{crs}(u_1), t_2, t_3, N))) = A_{\mathrm{getPub}}(A_{\mathrm{ZK}}(A_{\mathrm{crs}}(r_{t_1}), \beta(t_2), \beta(t_3), r_N)) = \beta(t_2)$ where the last equality holds because of implementation condition 27.

   Case 2: $t_4 = N^m \in \mathbf{N}_E$

   Then we have $A_{\mathrm{getPub}}(\beta(\mathrm{ZK}(\mathrm{crs}(t_1), t_2, t_3, N^m))) = A_{\mathrm{getPub}}(m)$ where $\tau(m) = \mathrm{ZK}(\mathrm{crs}(t_1), t_2, t_3, N^m)$ and $\tau(A_{\mathrm{getPub}}(m)) = t_2$ by definition of τ. By applying β on both sides, it follows that $\beta(t_2) = \beta(\tau(A_{\mathrm{getPub}}(m))) = A_{\mathrm{getPub}}(m)$ where the last equality holds because of Claim 1.

   Now, consider the case that $t = \mathrm{garbageZK}(t_1, t_2, t_3)$. By protocol condition 2, it follows that t was generated via τ. Thus, there is a $z \in \{0,1\}^*$ such that $t_3 = N^z$ and $t_2 = \tau(A_{\mathrm{getPub}}(z))$. Therefore, it holds that $A_{\mathrm{getPub}}(\beta(\mathrm{garbageZK}(t_1, t_2, t_3))) = A_{\mathrm{getPub}}(z) \overset{(*)}{=} \beta(\tau(A_{\mathrm{getPub}}(z))) = \beta(t_2)$. Here, the equality $(*)$ holds because of Claim 1.

5. "$F = \mathrm{verify}_{\mathrm{ZK}}$"

   If $\beta(t_2)$ has not the type zero-knowledge proof, then the left hand side is $\perp$ by definition of β, and the right hand side is $\perp$ by implementation condition 22.

   Therefore consider t_2 to be of the form $\mathrm{ZK}(u_1, u_2, u_3, u_4)$ or $\mathrm{garbageZK}(u_1, u_2, u_3)$. Additionally has to hold that $t_1 = u_1$ and that by protocol condition 9 t_1 is of the form $\mathrm{crs}(N_1)$ for some $N_1 \in \mathbf{N}_P$. Consider the following subcases:

   (a) $t_2 = \mathrm{ZK}(\mathrm{crs}(N_1), u_2, u_3, u_4)$ with $u_4 \in \mathbf{N}_P$.

   Then u_4 has the form N_2 for $N_2 \in \mathbf{N}_P$ by protocol conditions 8 and 3. By Theorem 2.4.4 and the fact that Sim is DY (Lemma A.1.3 and 2.4.3 (i)), it holds that the proof is, valid, more precisely $(u_2, u_3) \in R^{\mathrm{sym}}_{\mathrm{honest}}$. Therefore, it follows $\mathrm{eval}_{\mathrm{verify}_{\mathrm{ZK}}}(t_1, t_2) = t_2$. Thus, it holds $\beta(t_2) = A_{\mathrm{ZK}}(A_{\mathrm{crs}}(r_{N_1}), \beta(u_2), \beta(u_3), r_{N_2})$. By Theorem A.1.4 follows $(\beta(u_2), \beta(u_3)) \in R^{\mathrm{comp}}_{\mathrm{honest}}$, therefore – by completeness of the zero-knowledge proof system – this gives a correct proof. Thus verification succeeds, and therefore by implementation condition 21 $A_{\mathrm{verify}_{\mathrm{ZK}}}(\beta(t_1), \beta(t_2)) = \beta(t_2)$.

   (b) $t_2 = \mathrm{ZK}(\mathrm{crs}(N_1), u_2, u_3, u_4)$ with $u_4 \in \mathbf{N}_E$.

   Then $u_4 = N^z$ with $\tau(z) = t_2$ and by definition of τ holds $z = A_{\mathrm{verify}_{\mathrm{ZK}}}(A_{\mathrm{crsof}}(z), z) \overset{*}{=} A_{\mathrm{verify}_{\mathrm{ZK}}}(\beta(\tau(A_{\mathrm{crsof}}(z))), \beta(\tau(z))) = A_{\mathrm{verify}_{\mathrm{ZK}}}(\beta(\mathrm{crs}(N_1)), \beta(t_2)) = A_{\mathrm{verify}_{\mathrm{ZK}}}(\beta(t_1)), \beta(t_2))$, on the other hand $\beta(\mathrm{eval}_{\mathrm{verify}_{\mathrm{ZK}}}(t_1, t_2)) \overset{**}{=} \beta(t_2) \overset{*}{=} \beta(\tau(z)) = z$, where in both cases $(*)$ hold because of claim 1. The equality $(**)$ requires that $(u_2, u_3) \in R^{\mathrm{sym}}_{\mathrm{adv}}$. This holds because t_2 was constructed by τ and therefore u_2 was constructed by symbolic extraction (if an extraction failure has occurred, we would already have stopped earlier) s.t. $(u_2, u_3) \in R^{\mathrm{sym}}_{\mathrm{adv}}$.

   (c) $t_2 = \mathrm{garbageZK}(\mathrm{crs}(N_1), u_2, u_3)$.

   By protocol condition 2 holds that t_2 was produced by τ. Thus $u_3 = N^z$ with $\tau(z) = t_2$. Because $u_1 = t_1 = \mathrm{crs}(N_1)$ for $N_1 \in \mathbf{N}_P$ follows by definition of τ that $\perp = A_{\mathrm{verify}_{\mathrm{ZK}}}(A_{\mathrm{crsof}}(z), z) = A_{\mathrm{verify}_{\mathrm{ZK}}}(\beta(t_1), \beta(t_2))$ (by implementation condition 21). By definition of $\mathrm{verify}_{\mathrm{ZK}}$ follows that $\mathrm{eval}_{\mathrm{verify}_{\mathrm{ZK}}}(t_1, t_2) = \perp$ and therefore $\beta(\mathrm{eval}_{\mathrm{verify}_{\mathrm{ZK}}}(t_1, t_2)) = \perp$, too.

6. "$F = \mathrm{iszk}$"

   If t_1 is not of the form $\mathrm{ZK}(\mathrm{crs}(N_1), u_1, u_2, N_2)$ or $\mathrm{garbageZK}(u_1, u_2, N_1)$ with $N_1, N_2 \in \mathbf{N}$ then $\beta(t_1)$ is not of type zero-knowledge proof. Therefore $A_{\mathrm{iszk}}(\beta(t_1)) = \perp$ by implementation condition 18. On the other hand holds $\beta(\mathrm{eval}_{\mathrm{iszk}}(t_1)) = \beta(\perp) = \perp$.

   So let t_1 be of the form $\mathrm{ZK}(\mathrm{crs}(N_1), u_1, u_2, N_2)$ or $\mathrm{garbageZK}(u_1, u_2, N_1)$ with $N_1, N_2 \in \mathbf{N}$. Then $\beta(\mathrm{eval}(_{\mathrm{iszk}}(t_1))) = \beta(t_1)$ and $A_{\mathrm{iszk}}(\beta(t_1)) = \beta(t_1)$ by implementation condition 18 because $\beta(t_1)$ has type zero-knowledge proof.

7. "$F \in \{\text{isenc}, \text{issig}, \text{isek}, \text{isvk}, \text{iscrs}\}$"

   Analogue to the case $F = \text{iszk}$.

8. "$F = \text{crsof}$"

   If t_1 is not of the form $\text{ZK}(u_1, u_2, u_3, N)$ or $\text{garbageZK}(u_1, u_2, N)$ with $N \in \mathbf{N}$. Then $\text{eval}_{\text{crsof}}(t_1) = \bot$ and $\beta(t_1)$ is not of type zero-knowledge proof, therefore by implementation condition 24 holds $A_{\text{crsof}}(\beta(t_1)) = \bot$.

   In the other both cases holds $\beta(\text{eval}_{\text{crsof}}(t_1)) = \beta(u_1)$. Consider the following subcases:

   (a) $t_1 = \text{ZK}(u_1, u_2, u_3, N)$ with $N \in \mathbf{N}_P$.
   So the term was generated by the protocol, therefore - by protocol condition 8 - holds that $u_1 = \text{crs}(M)$ for some $M \in \mathbf{N}_P$. Thus holds $A_{\text{crsof}}(\beta(t_1)) = A_{\text{crsof}}(A_{\text{ZK}}(A_{\text{crs}}(r_M), \beta(u_2), \beta(u_3), r_N)) \overset{*}{=} A_{\text{crs}}(r_M) = \beta(\text{crs}(M)) = \beta(\text{eval}_{\text{crsof}}(t_1))$ where (*) holds by implementation condition 23.

   (b) $t_1 = \text{garbageZK}(u_1, u_2, N)$.
   By protocol condition 2 follows that t_1 was constructed by τ, i.e. $t_1 = \text{garbageZK}(u_1, u_2, N^z)$ for some $z \in \{0,1\}^*$ of type zero-knowledge and $u_1 = \tau(A_{\text{crsof}}(z))$. Thus we have: $\beta(u_1) = \beta(\tau(A_{\text{crsof}}(z))) \overset{*}{=} A_{\text{crsof}}(z) = A_{\text{crsof}}(\beta(t_1))$ where the last equality holds by definition of β and (*) holds by claim 1.

   (c) $t_1 = \text{ZK}(u_1, u_2, u_3, N^z)$ with $N^z \in \mathbf{N}_E$.
   This case is analogue to the case $t_1 = \text{garbageZK}$.

9. "$F \in \{\text{ekof}, \text{vkof}\}$"

   Analogue to the case $F = \text{crsof}$.

10. "$F = \mathsf{Enc}$"

    By protocol condition 3 holds that $t_3 = N$ for $N \in \mathbf{N}_P$. If t_1 is of the form $\text{ek}(u)$, then $\beta(\mathsf{Enc}(t_1, t_2, t_3)) = A_{\mathsf{Enc}}(\beta(t_1), \beta(t_2), r_N) = A_{\mathsf{Enc}}(\beta(t_1), \beta(t_2), \beta(t_3))$, because $\beta(N) = r_N$.

    So let t_1 be not of the form $\text{ek}(u)$. Thus $\beta(\mathsf{Enc}(t_1, t_2, t_3)) = \bot$ and $A_{\mathsf{Enc}}(\beta(t_1), \beta(t_2), \beta(t_3)) = \bot$, because $\beta(t_1)$ is not of type encryption key and implementation condition 19.

11. "$F = \mathsf{Dec}$"

    By protocol condition 6, $t_1 = \text{dk}(N)$ with $N \in \mathbf{N}_P$. We distinguish the following cases for t_2:

    (a) $t_2 = \mathsf{Enc}(\text{ek}(N), u_2, M)$ with $M \in \mathbf{N}_P$
    Then $A_{\mathsf{Dec}}(\beta(t_1), \beta(t_2)) = A_{\mathsf{Dec}}(A_{\text{dk}}(r_N), A_{\mathsf{Enc}}(A_{\text{ek}}(N), \beta(u_2), r_M)) = \beta(u_2)$ by implementation condition 12. Furthermore $\beta(\mathsf{Dec}(t_1, t_2)) = \beta(u_2)$ by definition of Dec.

(b) $t_2 = \mathsf{Enc}(\mathrm{ek}(N), u_2, N^c)$

Then t_2 was produced by τ and hence c is of type ciphertext and $\tau(A_{\mathsf{Dec}}(A_{\mathrm{dk}}(r_N), c)) = u_2$. Then by Claim 1, $A_{\mathsf{Dec}}(A_{\mathrm{dk}}(r_N), c) = \beta(u_2)$ and hence $A_{\mathsf{Dec}}(\beta(t_1), \beta(t_2)) = A_{\mathsf{Dec}}(A_{\mathrm{dk}}(r_N), c) = \beta(u_2) = \beta(\mathsf{Dec}(t_1, t_2))$.

(c) $t_2 = \mathsf{Enc}(u_1, u_2, u_3)$ with $u_1 \neq \mathrm{ek}(N)$

As shown above (case $F = \mathrm{ekof}$), $A_{\mathrm{ekof}}(\beta(\mathsf{Enc}(u_1, u_2, u_3)) = \beta(\mathrm{ekof}(\mathsf{Enc}(u_1, u_2, u_3)) = \beta(u_1)$. Moreover, from Claim 3, $A_{\mathrm{ekof}}(\beta(\mathsf{Enc}(u_1, u_2, u_3)) = \beta(u_1) \neq \beta(\mathrm{ek}(N)) = A_{\mathrm{ek}}(r_N)$. Thus by implementation condition 4, $A_{\mathsf{Dec}}(\beta(t_1), \beta(t_2)) = A_{\mathsf{Dec}}(A_{\mathrm{dk}}(r_N), \beta(\mathsf{Enc}(u_1, u_2, u_3))) = \bot$. Furthermore, $\mathsf{Dec}(t_1, t_2) = \bot$ and thus $\beta(\mathsf{Dec}(t_1, t_2)) = \bot$.

(d) $t_2 = \mathrm{garbageEnc}(u_1, N^c)$

Assume that $m := A_{\mathsf{Dec}}(\beta(t_1), \beta(t_2)) = A_{\mathsf{Dec}}(A_{\mathrm{dk}}(r_N), c) \neq \bot$. By implementation condition 13 this implies $A_{\mathrm{ekof}}(c) = A_{\mathrm{ek}}(r_N)$ and thus $\tau(A_{\mathrm{ekof}}(c)) = \tau(A_{\mathrm{ek}}(r_N)) = \mathrm{ek}(N)$. By protocol condition 2, t_2 has been produced by τ, i.e., $t_2 = \tau(c)$. Hence c is of type ciphertext. Then, however, we would have $\tau(c) = \mathsf{Enc}(\mathrm{ek}(N), \tau(m), N^c) \neq t_2$. This is a contradiction to $t_2 = \tau(c)$, so the assumption that $A_{\mathsf{Dec}}(\beta(t_1), \beta(t_2)) \neq \bot$ was false. So $A_{\mathsf{Dec}}(\beta(t_1), \beta(t_2)) = \bot = \beta(\bot) = \beta(\mathsf{Dec}(t_1, \mathrm{garbageEnc}(u_1, N^c)))$.

(e) All other cases

Then $\beta(t_2)$ is not of type ciphertext. By implementation condition 13, $A_{\mathrm{ekof}}(\beta(t_2)) = \bot$. Hence $A_{\mathrm{ekof}}(\beta(t_2)) \neq A_{\mathrm{ek}}(r_N)$ and by implementation condition 4, $A_{\mathsf{Dec}}(\beta(t_1), \beta(t_2)) = A_{\mathsf{Dec}}(A_{\mathrm{dk}}(r_N), \beta(t_2)) = \bot = \beta(\mathsf{Dec}(t_1, t_2))$.

12. "$F = \mathrm{sig}$"

By protocol conditions 3 and 7 we have that $t_1 = \mathsf{sk}(N)$ and $t_3 = M$ for $N, M \in \mathbf{N}_P$. Then $\beta(\mathrm{sig}(t_1, t_2, t_3)) = A_{\mathrm{sig}}(A_{\mathsf{sk}}(r_N), \beta(t_2), r_M) = A_{\mathrm{sig}}(\beta(\mathsf{sk}(N)), \beta(t_2), \beta(M)) = A_{\mathrm{sig}}(\beta(t_1), \beta(t_2), \beta(t_3))$.

13. "$F = \mathrm{verify}_{\mathrm{sig}}$"

We distinguish the following subcases:

(a) "$t_1 = \mathsf{vk}(N)$ and $t_2 = \mathrm{sig}(\mathsf{sk}(N), u_2, M)$ with $N, M \in \mathbf{N}_P$"

Then $A_{\mathrm{verify}_{\mathrm{sig}}}(\beta(t_1), \beta(t_2)) = A_{\mathrm{verify}_{\mathrm{sig}}}(A_{\mathsf{vk}}(r_N), A_{\mathrm{sig}}(A_{\mathsf{sk}}(r_N), \beta(u_2), r_M)) \overset{*}{=} \beta(u_2) = \beta(\mathrm{verify}_{\mathrm{sig}}(\underline{t}))$ where $(*)$ uses implementation condition 15.

(b) "$t_2 = \mathrm{sig}(\mathsf{sk}(N), u_2, M)$ and $t_1 \neq \mathsf{vk}(N)$ with $N, M \in \mathbf{N}_P$"

By Claim 3, $\beta(t_1) \neq \beta(\mathsf{vk}(N))$ Furthermore $A_{\mathrm{verify}_{\mathrm{sig}}}(\beta(\mathsf{vk}(N)), \beta(t_2)) = A_{\mathrm{verify}_{\mathrm{sig}}}(\beta(t_1), A_{\mathrm{sig}}(A_{\mathsf{sk}}(r_N), \beta(u_2), r_M)) \overset{*}{=} \beta(u_2) \neq \bot$. Hence with implementation condition 16, $A_{\mathrm{verify}_{\mathrm{sig}}}(\beta(t_1), \beta(t_2)) = \bot = \beta(\bot) = \mathrm{verify}_{\mathrm{sig}}(t_1, t_2)$.

(c) "$t_1 = \mathsf{vk}(N)$ and $t_2 = \mathrm{sig}(\mathsf{sk}(N), u_2, M^s)$"

Then t_2 was produced by τ and hence s is of type signature with $\tau(A_{\text{vkof}}(s)) = \mathsf{vk}(N)$ and $m := A_{\text{verify}_{\text{sig}}}(A_{\text{vkof}}(s), s) \neq \bot$ and $u_2 = \tau(m)$. Hence with Claim 1 we have $m = \beta(\tau(m)) = \beta(u_2)$ and $\beta(t_1) = \beta(\mathsf{vk}(N)) = \beta(\tau(A_{\text{vkof}}(s))) = A_{\text{vkof}}(s)$. Thus $A_{\text{verify}_{\text{sig}}}(\beta(t_1), \beta(t_2)) = A_{\text{verify}_{\text{sig}}}(A_{\text{vkof}}(s), s) = m = \beta(u_2)$. And $\beta(\text{verify}_{\text{sig}}(t_1, t_2)) = \beta(\text{verify}_{\text{sig}}(\mathsf{vk}(N), \text{sig}(\mathsf{sk}(N), u_2, M^s))) = \beta(u_2)$.

(d) "$t_2 = \text{sig}(\mathsf{sk}(N), u_2, M^s)$ and $t_1 \neq \mathsf{vk}(N)$"

As in the previous case, $A_{\text{verify}_{\text{sig}}}(A_{\text{vkof}}(s), s) \neq \bot$ and $\beta(\mathsf{vk}(N)) = A_{\text{vkof}}(s)$. Since $t_1 \neq \mathsf{vk}(N)$, by Claim 3, $\beta(t_1) \neq \beta(\mathsf{vk}(N)) = A_{\text{vkof}}(s)$. From implementation condition 16 and $A_{\text{verify}_{\text{sig}}}(A_{\text{vkof}}(s), s) \neq \bot$, we have $A_{\text{verify}_{\text{sig}}}(\beta(t_1), \beta(t_2)) = A_{\text{verify}_{\text{sig}}}(\beta(t_1), s) = \bot = \beta(\bot) = \beta(\text{verify}_{\text{sig}}(t_1, t_2))$.

(e) "$t_2 = \text{garbageSig}(u_1, N^s)$"

Then t_2 was produced by τ and hence s is of type signature and either $A_{\text{verify}_{\text{sig}}}(A_{\text{vkof}}(s), s) = \bot$ or $\tau(A_{\text{vkof}}(s))$ is not of the form $\mathsf{vk}(\dots)$. The latter case only occurs if $A_{\text{vkof}}(s) = \bot$ as otherwise $A_{\text{vkof}}(s)$ is of type verification key and hence $\tau(A_{\text{vkof}}(s)) = \mathsf{vk}(\dots)$. Hence in both cases $A_{\text{verify}_{\text{sig}}}(A_{\text{vkof}}(s), s) = \bot$. If $\beta(t_1) = A_{\text{vkof}}(s)$ then $A_{\text{verify}_{\text{sig}}}(\beta(t_1), \beta(t_2)) = A_{\text{verify}_{\text{sig}}}(A_{\text{vkof}}(s), s) = \bot = \beta(\text{verify}_{\text{sig}}(t_1, t_2))$. If $\beta(t_1) \neq A_{\text{vkof}}(s)$ then by implementation condition 16, $A_{\text{verify}_{\text{sig}}}(\beta(t_1), \beta(t_2)) = A_{\text{verify}_{\text{sig}}}(\beta(t_1), s) = \bot$. Thus in both cases, with $\text{verify}_{\text{sig}}(t_1, t_2) = \bot$ we have $A_{\text{verify}_{\text{sig}}}(\beta(t_1), \beta(t_2)) = \bot = \beta(\text{verify}_{\text{sig}}(t_1, t_2))$.

(f) All other cases

Then $\beta(t_2)$ is not of type signature, hence by implementation condition 5, $A_{\text{vkof}}(\beta(t_2)) = \bot$, hence $\beta(t_1) \neq A_{\text{vkof}}(\beta(t_2))$, and by implementation condition 16 we have $A_{\text{verify}_{\text{sig}}}(\beta(t_1), \beta(t_2)) = \bot = \beta(\text{verify}_{\text{sig}}(t_1, t_2))$.

14. "$F \in \{\text{pair}, \text{fst}, \text{snd}, \text{string}_0, \text{unstring}_0, \text{string}_1, \text{unstring}_1, \text{empty}\}$" The claim follows directly from the definition of β.

15. "$F = \text{equals}$"

If $t_1 = t_2$ then holds $\beta(\text{equals}(t_1, t_2)) = \beta(t_1) = A_{\text{equals}}(\beta(t_1), \beta(t_1)) = A_{\text{equals}}(\beta(t_1), \beta(t_2))$. So let $t_1 \neq t_2$. By Claim 3 holds $t_i = \tau(\beta(t_i))$, so $\beta(t_1) \neq \beta(t_2)$, because otherwise $t_1 = t_2$. But then holds $A_{\text{equals}}(t_1, t_2) = \bot = \beta(\text{equals}(t_1, t_2))$

16. "$F \in \{\text{garbage}, \text{garbageEnc}, \text{garbageSig}, \text{garbageZK}\} \cup \mathbf{N}_E$"

By protocol condition 2, the constructor F does not occur in the protocol.

□