Computational Soundness of Symbolic Zero-knowledge Proofs: Weaker Assumptions and Mechanized Verification

Michael Backes
Saarland University and MPI-SWS
backes@cs.uni-saarland.de

Fabian Bendun
Saarland University
bendun@cs.uni-saarland.de

Dominique Unruh
University of Tartu
dominique.unruh@ut.ee

***Abstract*—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 formalisms and at the cost of placing strong assumptions on the underlying cryptography, which leaves only highly inefficient realizations.**

In this paper, 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.

***Keywords*-Symbolic zero-knowledge proofs; computational soundness; weaker assumptions; mechanized proofs**

I. Introduction

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 [22], [23], [36], e.g., see [28], [41], [2], [33], [38], [15]. 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], [30], [12], [11], [31], [37], [20], [18], [42].

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 [19] or the Pseudo Trust protocol [34], or they can prove the reception of a certificate from a trusted server without revealing the actual content, as in the Direct Anonymous Attestation (DAA) protocol [17]. More recently, ZK proofs have been used to develop novel schemes for anonymous webs of trust [7] as well as privacy-aware proof-carrying authorization [35].

A symbolic abstraction of (non-interactive) ZK proofs has recently been put forward in [9]. The proposed abstraction is suitable for mechanized proofs [9], [6] 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 [13]. 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 [26]. Thus the vast number of recently proposed, far more efficient

[1]A zero-knowledge proof [24] 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 [29]), Diffie-Hellman-style exponentiation (e.g., in [1]), or private contract signatures (e.g., in [27]).

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 [13] casts symbolic ZK proofs within an ad-hoc formalism that is not accessible to existing formal proof tools.

A. Our Contribution

In this paper, we make the following two contributions to this problem space:

- 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 [13] 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) [39], as well as several efficient schemes that are known to satisfy simulation-sound extractability (and the remaining properties), e.g., [32], [25], [40]. 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 [5], 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.

B. Outline of the Paper

First, we introduce our symbolic abstraction of (non-interactive) ZK proofs within CoSP in Section II. Section III contains the weaker cryptographic assumptions that we show to be sufficient for achieving computational soundness of ZK proofs. Our main theorem is presented in Section IV for which we give a proof overview in Section V. We show how to use our result in the applied-π calculus in Section VI. Section VII concludes and outlines future work.

II. Symbolic Model for Zero-knowledge

In this section, we describe our symbolic abstraction of zero-knowledge proofs.

Terms and constructors. We model nonces, probabilistic public-key encryption and signatures, pairs, strings, and zero-knowledge proofs. Except for the latter, our modeling closely follows that of [5]. The following grammar describes the set $\mathbf{T}$ of all terms that may occur in the symbolic model:

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

Here N represents nonces and ranges over $\mathbf{N}_P \cup \mathbf{N}_E$, two disjoint infinite sets of nonces, the protocol nonces and adversary nonces, respectively. $\mathrm{ek}(N), \mathrm{dk}(N), \mathrm{vk}(N), \mathrm{sk}(N)$ represent encryption, decryption, verification, and signing keys. $\mathrm{enc}(\mathrm{ek}(N_1), t, N_2)$ represents an encryption under public key $\mathrm{ek}(N_1)$ of plaintext t using algorithmic randomness N_2. (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.) $\mathrm{sig}(\mathrm{sk}(N_1), t, N_2)$ is a signature of t under signing key $\mathrm{sk}(N_1)$ with algorithmic randomness N_2. Bitstrings can be expressed using terms matching the nonterminal S. $\mathrm{garbage}(N)$ represents invalid terms, $\mathrm{garbageEnc}(t, N)$ and $\mathrm{garbageSig}(t, N)$ represent invalid encryptions and signatures (but which at a first glance seem to be valid encryptions/signatures with public key t).

Zero-knowledge proofs. The interesting part are the zero-knowledge proofs. To understand the meaning of a term $\mathrm{ZK}(\mathrm{crs}(N), x, w, M)$, we first need to introduce the relation $R^{\mathrm{sym}}_{\mathrm{adv}}$. This relation is part of the symbolic modeling, but all our results are parametric in $R^{\mathrm{sym}}_{\mathrm{adv}}$. (I.e., our result holds for any choice of $R^{\mathrm{sym}}_{\mathrm{adv}}$, as long as $R^{\mathrm{sym}}_{\mathrm{adv}}$ satisfies certain constraints.) $R^{\mathrm{sym}}_{\mathrm{adv}}$ specifies what a valid witness for a particular statement would be. For example, if we wish to show that we know a decryption key w that decrypts a given ciphertext x, then we define $R^{\mathrm{sym}}_{\mathrm{adv}} := \{(x, w) : \exists N, M, t. x = \mathrm{enc}(\mathrm{ek}(N), t, M), w = \mathrm{dk}(N)\}$.[3]

[3]Notice that it is no restriction that statement and witness consist of only one argument: we can encode tuples using the pair-constructor. Also, it is no restriction that we use the same relation for all ZK-proofs: To encode multiple relations $R_1, \ldots, R_n$, we can define a relation $R := \{((a, x), w) : \exists i. a = a_i \wedge (x, w) \in R_i\}$ where a_i are distinct terms without names or variables.

The term $\mathrm{ZK}(\mathrm{crs}(N), x, w, M)$ then represents a zero-knowledge proof constructed with respect to a common reference string $\mathrm{crs}(N)$ with statement x and witness w and using algorithmic randomness M. A valid proof satisfies $(x, w) \in R^{\mathrm{sym}}_{\mathrm{adv}}$. Note that our symbolic model does not ensure that any term $\mathrm{ZK}(\mathrm{crs}(N), x, w, M)$ is a valid proof. Instead, we provide a destructor $\mathrm{verify}_{\mathrm{ZK}}$ below that allows to check the validity. As we will see below, the statement x can be extracted from a proof, but the witness w is hidden.

Destructors. Protocol operations on terms are described by a set of destructors. These are partial functions from $\mathbf{T}^n$ to $\mathbf{T}$ (where n depends on the destructor). The destructors are specified in Figure 1. Note that there are a number of destructors that do not modify their input ($\mathrm{isek}, \mathrm{iszk}, \mathrm{equals}, \mathrm{verify}_{\mathrm{sig}}, \dots$). 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. We only describe the destructors related to ZK proofs. $\mathrm{getPub}(t)$ returns the statement x proven by a ZK proof t. getPub does not check whether the proof is actually valid; for this, we have $\mathrm{verify}_{\mathrm{ZK}}(t_1, t_2)$ which checks whether t_2 is a valid proof with respect to the CRS t_1. If so, t_2 is returned (and can, e.g., be fed into getPub); otherwise $\perp$ is returned. $\mathrm{iscrs}(t)$ and $\mathrm{iszk}(t)$ allow us to test if t is a CRS or a (possibly invalid) zero-knowledge proof.

Protocols. We use the protocol model from the CoSP framework [5]. There, a protocol is modeled as a (possibly infinite) tree of nodes. Each node corresponds to a particular protocol action such as receiving a term from the adversary, sending a previously computed term to the adversary, applying a constructor or destructor to previously computed terms (and branching depending on whether the application is successful), or picking a nonce. We do not describe the protocol model in detail here, but it suffices to know that a protocol can freely apply constructors and destructors (computation nodes), branch depending on destructor success, and communicate with the adversary. Despite the simplicity of the model, it is powerful enough to embed powerful calculi such as the applied π-calculus (shown in [5]) or RCF, a core calculus for F# (shown in [10]). (In Section VI, we present our computational soundness result in the applied π-calculus.)

Protocol conditions. The protocols we consider are subject to a number of conditions, listed in the Appendix. The most interesting protocol condition is *valid proofs condition*: During the symbolic execution of the protocol, whenever the protocol constructs a ZK proof $\mathrm{ZK}(c, x, w, N)$ we have $(x, w) \in R^{\mathrm{sym}}_{\mathrm{honest}}$. Here $R^{\mathrm{sym}}_{\mathrm{honest}}$ is some fixed but arbitrary relation with $R^{\mathrm{sym}}_{\mathrm{honest}} \subseteq R^{\mathrm{sym}}_{\mathrm{adv}}$ (like in $R^{\mathrm{sym}}_{\mathrm{adv}}$, our results are parametric in $R^{\mathrm{sym}}_{\mathrm{honest}}$). In the simplest case, we would have $R^{\mathrm{sym}}_{\mathrm{honest}} := R^{\mathrm{sym}}_{\mathrm{adv}}$. Then the valid proofs condition simply requires that the protocol never tries to construct a ZK-proof with an invalid witness. (We only impose this condition on the honest protocol, not on the adversary.) 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 $(x, w) \in R^{\mathrm{sym}}_{\mathrm{adv}}$. We call $R^{\mathrm{sym}}_{\mathrm{honest}}$ the *usage restriction*.

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

$$\frac{m \in S}{S \vdash m} \qquad \frac{N \in \mathbf{N}_E}{S \vdash N}$$

$$\frac{\begin{array}{cc} S \vdash t_1, \dots, t_n & t_1, \dots, t_n \in \mathbf{T} \\ F \text{ constructor or destructor} & F(t_1, \dots, t_n) \in \mathbf{T} \end{array}}{S \vdash \mathrm{eval}_F(\bar{t})}$$

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 message, the protocol explicitly communicated through the adversary.

Protocol execution. Given a particular protocol Π (modeled as a tree), the set of possible protocol traces is defined by traversing the tree: in case of an input node the adversary nondeterministically picks a term t with $S \vdash t$ where S are the terms sent so far through output nodes; at computation nodes, a new term is computed by applying a constructor or destructor to terms computed/received at earlier nodes; then the left or right successor is taken depending on whether the destructor succeeded. The sequence of nodes we traverse in this fashion is called a *symbolic node trace* of the protocol. By specifying sets of node traces, we can specify trace properties for a given protocol. We refer to [5] for details on the protocol model and its semantics.

III. Computational implementation

We now describe how to implement the constructors and destructors from the preceding section computationally. Following [5], 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. In Appendix B we give the full list of implementation conditions that the computational implementation must fulfill. These are mostly simple syntactic conditions (such as $A_{\mathrm{fst}}(A_{\mathrm{pair}}(x, y)) = x$). Furthermore,

$$
\begin{aligned}
\mathrm{dec}(\mathrm{dk}(t_1), \mathrm{enc}(\mathrm{ek}(t_1), m, t_2)) &= m\\
\mathrm{verify}_{\mathrm{sig}}(\mathrm{vk}(t_1), \mathrm{sig}(\mathrm{sk}(t_1), t_2, t_3)) &= t_2\\
\mathrm{isek}(\mathrm{ek}(t)) &= \mathrm{ek}(t)\\
\mathrm{isvk}(\mathrm{vk}(t)) &= \mathrm{vk}(t)\\
\mathrm{isenc}(\mathrm{enc}(\mathrm{ek}(t_1), t_2, t_3)) &= \mathrm{enc}(\mathrm{ek}(t_1), t_2, t_3)\\
\mathrm{isenc}(\mathrm{garbageEnc}(t_1, t_2)) &= \mathrm{garbageEnc}(t_1, t_2)\\
\mathrm{issig}(\mathrm{sig}(\mathrm{sk}(t_1), t_2, t_3)) &= \mathrm{sig}(\mathrm{sk}(t_1), t_2, t_3)\\
\mathrm{issig}(\mathrm{garbageSig}(t_1, t_2)) &= \mathrm{garbageSig}(t_1, t_2)\\
\mathrm{iscrs}(\mathrm{crs}(t_1)) &= \mathrm{crs}(t_1)\\
\mathrm{iszk}(\mathrm{ZK}(t_1, t_2, t_3, t_4)) &= \mathrm{ZK}(t_1, t_2, t_3, t_4)\\
\mathrm{iszk}(\mathrm{garbageZK}(t_1, t_2, t_3)) &= \mathrm{garbageZK}(t_1, t_2, t_3)\\
\mathrm{ekof}(\mathrm{enc}(\mathrm{ek}(t_1), t_2, t_3)) &= \mathrm{ek}(t_1)\\
\mathrm{ekof}(\mathrm{garbageEnc}(t_1, t_2)) &= t_1
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{crsof}(\mathrm{ZK}(\mathrm{crs}(t_1), t_2, t_3, t_4)) &= \mathrm{crs}(t_1)\\
\mathrm{crsof}(\mathrm{garbageZK}(t_1, t_2, t_3)) &= t_1\\
\mathrm{vkof}(\mathrm{sig}(\mathrm{sk}(t_1), t_2, t_3)) &= \mathrm{vk}(t_1)\\
\mathrm{vkof}(\mathrm{garbageSig}(t_1, t_2)) &= t_1\\
\mathrm{fst}(\mathrm{pair}(t_1, t_2))) &= t_1\\
\mathrm{snd}(\mathrm{pair}(t_1, t_2)) &= t_2\\
\mathrm{unstring}_0(\mathrm{string}_0(s)) &= s\\
\mathrm{unstring}_1(\mathrm{string}_1(s)) &= s\\
\mathrm{getPub}(\mathrm{ZK}(t_1, t_2, t_3, t_4)) &= t_2\\
\mathrm{getPub}(\mathrm{garbageZK}(t_1, t_2, t_3)) &= t_2\\
\mathrm{equals}(x, x) &= x\\
\mathrm{verify}_{\mathrm{ZK}}(\mathrm{crs}(t_1), \mathrm{ZK}(\mathrm{crs}(t_1), t_2, t_3, t_4))&\\
= \mathrm{ZK}(\mathrm{crs}(t_1), t_2, t_3, t_4) \text{ if } (t_2, t_3) &\in R^{\mathrm{sym}}_{\mathrm{adv}}
\end{aligned}
$$

Figure 1. Definition of destructors. If no rule matches, a destructor returns $\bot$.

we require that A_{enc} and A_{sig} correspond to an IND-CCA secure encryption scheme and a strongly unforgeable signature scheme. These conditions are essentially the same as in [5]. Here, we will only discuss the cryptographic properties the implementation of ZK proofs should satisfy.

Properties of ZK proofs. In [14], it was shown that for getting computational soundness of (non-interactive) zero-knowledge proofs, we need at least the following properties:[4] *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 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* (see below). Furthermore, they required for convenience that the verification and the extraction algorithm are deterministic.

The interesting property is non-malleability: Intuitively, non-malleability means that given a proof for some statement x, it is not possible to derive a proof for some other statement x', even if x logically entails x'. (For example, given a proof that the ciphertext c contains a plaintext $i < 5$ it should not be possible to construct a proof that c contains $i < 6$.) There are several variants of non-malleability; [14] used 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 [26]). 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.

[4]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.

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.

In the symbolic model, 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 $R^{\mathrm{comp}}_{\mathrm{adv}} \supseteq R^{\mathrm{comp}}_{\mathrm{honest}}$. All conditions assume that honest participants use $(x, w) \in R^{\mathrm{comp}}_{\mathrm{honest}}$. ("Weakly" distinguishes our notion from that in [14] which requires extraction ZK.)

Definition 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 $\mathcal{O}_{\mathbf{P}}(x, w) := \bot$ otherwise, and let $\mathcal{O}_{\mathbf{S}}(x, w) := \mathbf{S}(x, \mathrm{crs}, \mathrm{simtd})$ if

$(x,w) \in R^{\text{comp}}_{\text{honest}}$ and $\mathcal{O}_{\mathbf{S}}(x,w) := \bot$ otherwise. Then

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

is negligible in η.

- Honest simulation-sound extractability: Let a polynomial-time oracle adversary $\mathcal{A}$ be given. Let $(\text{crs},\text{simtd},\text{extd}) \leftarrow \mathbf{K}(1^\eta)$. Let $\mathcal{O}(x,w) := \mathbf{S}(x,\text{crs},\text{simtd})$ if $(x,w) \in R^{\text{comp}}_{\text{honest}}$ and $\bot$ otherwise. Let $(x,\text{proof}) \leftarrow \mathcal{A}^{\mathcal{O}}(1^\eta,\text{crs})$. Let $w \leftarrow \mathbf{E}(x,\text{proof},\text{extd})$. Then with overwhelming probability, if $\mathbf{V}(x,\text{proof},\text{crs}) = 1$ and proof was not output by $\mathcal{O}$ then $(x,w) \in R^{\text{comp}}_{\text{adv}}$.
- Unpredictability: Let a polynomial-time adversary $\mathcal{A}$ be given. Let $(\text{crs},\text{simtd},\text{extd}) \leftarrow \mathbf{K}(1^\eta)$. Let $(x,w,\text{proof}') \leftarrow \mathcal{A}(1^\eta,\text{crs},\text{simtd},\text{extd})$. Then with overwhelming probability, it holds $\text{proof}' \neq \mathbf{P}(x,w,\text{crs})$ or $(x,w) \notin R^{\text{comp}}_{\text{honest}}$.
- Length-regularity: Let two witnesses w and w', and statements x and x' be given such that $|x| = |x'|$, and $|w| = |w'|$. Let $(\text{crs},\text{simtd},\text{extd}) \leftarrow \mathbf{K}(1^\eta)$. Then let $\text{proof} \leftarrow \mathbf{P}(x,w,\text{crs})$ and $\text{proof}' \leftarrow \mathbf{P}(x',w',\text{crs})$. Then we get $|\text{proof}| = |\text{proof}'|$ 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 then require that $A_{\text{crs}}, A_{\text{ZK}}, A_{\text{verify}_{\text{ZK}}}$ 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 $R^{\text{comp}}_{\text{honest}}, R^{\text{comp}}_{\text{adv}}$. We stress that using the construction in [39] on a length-regular and extractable NIZK leads to weakly symbolically-sound ZK proof system. The proof is analogous to the one in [39], see Appendix G.

The relations. It remains to specify what conditions we place on the relations $R^{\text{comp}}_{\text{honest}}, R^{\text{comp}}_{\text{adv}}$. Obviously, we cannot expect computational soundness if we allow arbitrary $R^{\text{comp}}_{\text{honest}}, R^{\text{comp}}_{\text{adv}}$. Instead, we need to formulate the fact that $R^{\text{comp}}_{\text{honest}}, R^{\text{comp}}_{\text{adv}}$ somehow correspond to the symbolic relations $R^{\text{sym}}_{\text{honest}}, R^{\text{sym}}_{\text{adv}}$. We thus give minimal requirements on the relationship between those relations. Essentially, we want that whenever $(x,w) \in R^{\text{sym}}_{\text{honest}}$ then for the corresponding computational bitstrings m_x, m_w we have $(m_x,m_w) \in R^{\text{comp}}_{\text{honest}}$; this guarantees that if symbolically, we respect the usage restriction $R^{\text{sym}}_{\text{honest}}$, then computationally we only use witnesses the honest protocol is allowed to use. And whenever $(m_x,m_w) \in R^{\text{comp}}_{\text{adv}}$ we have $(x,w) \in R^{\text{sym}}_{\text{adv}}$; 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, these conditions are used to show Lemmas 4 and 6 in the computational soundness proof below.) To model correspondence between the symbolic terms x, w and the bitstrings m_x, m_w, we define a function img_η that translates a term to a bitstring (essentially by applying A_F for each constructor F). 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. We use the definition of a *consistent environment* that lists various natural properties an environment will have (such as mapping ZK-terms to bitstrings of the right type); the definition of consistent environments is deferred to Appendix C. Given these notions, we can formalize the conditions $R^{\text{comp}}_{\text{honest}}, R^{\text{comp}}_{\text{adv}}$ should satisfy:

Definition 2 (Implementation of relations): A pair of relations $R^{\text{comp}}_{\text{honest}}, R^{\text{comp}}_{\text{adv}}$ on $\{0,1\}^*$ implement a relation $R^{\text{sym}}_{\text{adv}}$ on $\mathbf{T}$ with usage restriction $R^{\text{sym}}_{\text{honest}}$ if the following conditions hold for any consistent $\eta \in \mathcal{E}$:

(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 $R^{\text{comp}}_{\text{honest}} \subseteq R^{\text{comp}}_{\text{adv}}$ $\diamond$

We briefly give some examples for symbolic relations and their implementation. The detailed proofs that they indeed satisfy Definition 2 are postponed to Appendix H.

Valid encryptions. Several protocols require one party needs to show that a produced ciphertext is valid. This is basically done by showing, that there is some randomness, such that the encryption algorithm, applied to a public encryption key and the content, leads to the given ciphertext. Symbolically, this can be abstracted to the following relation:

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

Ability of decrypting. In section VI, we give a short example, how automated verification for the symbolic model can be done. We give a variation of the Needham-Schroeder protocol that uses a proof a party is able to decrypt a given ciphertext. Therefore, we use the following relations:

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

Summary of implementation conditions. Summarizing, we require that the functions A_F satisfy a list of implementation conditions. The most important condition is that $A_{\text{crs}}, A_{\text{ZK}}, A_{\text{verify}_{\text{ZK}}}$ correspond to a weakly symbolically-sound ZK proof system for some relations $R^{\text{comp}}_{\text{honest}}, R^{\text{comp}}_{\text{adv}}$ which implement $R^{\text{sym}}_{\text{adv}}$ with usage restriction $R^{\text{sym}}_{\text{honest}}$.

Protocol execution. The CoSP framework specifies semantics for executing a given protocol in the computational model given an computational implementation A_F. The execution is analogous to the symbolic execution (page 3), except that computation nodes apply functions A_F instead of constructors and destructors (with branching depending on $A_F(\dots) \overset{?}{=} \bot$). Input and output nodes receive and send bitstrings to a probabilistic polynomial-time adversary. This probabilistic process yields a trace of nodes, the *computational node trace*. Details are specified in [5].

IV. Computational soundness

Using the definitions from Section II and III, we can finally state our main result. A *trace property* is a prefix-closed, efficiently decidable set $\mathcal{P}$ of node traces. We say a protocol Π *symbolically satisfies* $\mathcal{P}$ if every symbolic node trace (see page 3) of Π is in $\mathcal{P}$. We say Π *computationally satisfies* $\mathcal{P}$ if the computational node trace (see page 6) is in $\mathcal{P}$ with overwhelming probability.

Theorem 1 (Computational soundness of ZK proofs): Let Π be a protocol satisfying the protocol conditions listed in the appendix. Let A_F be a computational implementation satisfying the implementation conditions from Section III. Then for any node trace $\mathcal{P}$, if Π symbolically satisfies $\mathcal{P}$, then Π computationally satisfies $\mathcal{P}$. $\diamond$

We describe the proof in Section V.

V. The proof

In this section, we describe our proof of computational soundness (Theorem 1). First, we describe how the computational soundness proof for encryptions and signatures is done in the CoSP framework (Section V-A). 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 [14] (Section V-B). It is instructive to compare their approach to ours. In Section V-C, we describe the idea underlying our proof (using simulation-sound extractability instead of extraction-soundness). Finally, in Section V-D we give an overview over our proof. The full proof is given in appendix E. The lemmas in this overview are simplified and informal.

A. Computational soundness proofs in CoSP

Remember that in the CoSP framework, a protocol is modeled as a tree whose nodes correspond to the steps of the protocol execution; security properties are expressed as sets of node traces. Computational soundness means that for any polynomial-time adversary A the trace in the computational execution is, except with negligible probability, also a possible node trace in the symbolic execution. The approach for showing this is to construct a so-called simulator Sim. The simulator is a machine that interacts with a symbolic execution of the protocol Π on the one hand, and with

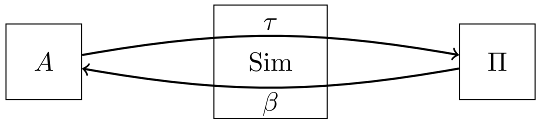


Figure 2. A typical CoSP simulator

the adversary A on the other hand; we call this a hybrid execution. (See Figure 2.) The simulator has to satisfy the following two properties:

- Indistinguishability: The node trace in the hybrid execution is computationally indistinguishable from that in the computational execution with adversary A.
- Dolev-Yaoness: The simulator Sim never (except for negligible probability) sends terms t to the protocol with $S \nvdash t$ where S is the list of terms Sim received from the protocol so far.

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.

How to construct a simulator? In [5], 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.) 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):

- $\tau(c) := \mathrm{enc}(\mathrm{ek}(M), t, N)$ if c has earlier been output by $\beta(\mathrm{enc}(\mathrm{ek}(M), t, N))$ for some $M \in \mathbf{N}, N \in \mathbf{N}_P$
- $\tau(c) := \mathrm{enc}(\mathrm{ek}(M), \tau(m), N^c)$ if c is of type ciphertext and $\tau(A_{\mathrm{ekof}}(c)) = \mathrm{ek}(M)$ for some $M \in \mathbf{N}_P$ and $m := A_{\mathrm{dec}}(A_{\mathrm{dk}}(r_M), c) \neq \bot$
- $\beta(\mathrm{enc}(\mathrm{ek}(N), t, M)) := A_{\mathrm{enc}}(A_{\mathrm{ek}}(r_N), \beta(t), r_M)$ if $M \in \mathbf{N}_P$
- $\beta(\mathrm{enc}(\mathrm{ek}(M), t, N^m)) := m$ 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 $t_1, \dots, t_n$, we have that $\beta(F(t_1, \dots, t_n)) = A_F(\beta(t_1), \dots, \beta(t_n))$. 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 [5] 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 $S \nvdash t_{bad}$) 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.

B. Computational soundness based on extraction ZK

We now describe how computational soundness for zero-knowledge proofs was shown in [14], based on the strong assumption of extraction zero-knowledge. Our presentation strongly deviates from the details of the proof in [14]; 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 described in Section V-A, 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 $\beta(\mathrm{ZK}(\dots, N^m)) := m$. 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 $\tau(z) := \mathrm{garbageZK}(\dots)$.). Finally, $\tau(z)$ returns $\mathrm{ZK}(\mathrm{crs}(N), \tau(x), \tau(w), N^z)$.

The proof of indistinguishability is analogous to that in Section V-A, 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 Section V-A, except that we introduce three more intermediate simulators Sim_1, Sim_2, and Sim_3. (See Figure 3.) 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 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 that in Section V-A. 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.

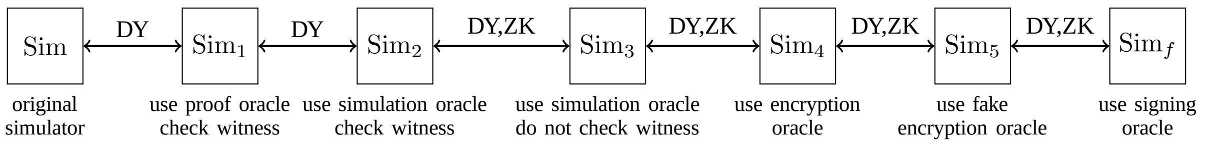
C. *Proof idea*

We now describe the idea of our approach that allows us to get rid of extraction ZK. As explained in Section V-B, 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 $x = \mathrm{enc}(\mathrm{ek}(N), t, M)$, then the witness is $\mathrm{dk}(N)$ which can just be read off x.) We stress that we do not claim that the witness can be deduced (in the sense of $\vdash$) from x, only that it can be efficiently computed.

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 received 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., $(x, w) \in R^{\mathrm{sym}}_{\mathrm{adv}}$) 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 $(x, w) \in R^{\mathrm{sym}}_{\mathrm{adv}}$ and $S \vdash w$). In this case, we can finish the proof analogously to that in Section V-B: Indistinguishability of Sim follows by carefully checking all cases, and the Dolev-Yaoness by the same sequence of



Sim
DY
Sim$_1$
DY
Sim$_2$
DY,ZK
Sim$_3$
DY,ZK
Sim$_4$
DY,ZK
Sim$_5$
DY,ZK
Sim$_f$
original simulator
use proof oracle check witness
use simulation oracle check witness
use simulation oracle do not check witness
use encryption oracle
use fake encryption oracle
use signing oracle

Figure 3. 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 V-D).

simulators as in Section V-B. 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 do we show that $\mathbf{SymbExtr}(S, x)$ always succeeds? Two things might go wrong. First, there might be no valid witness w with $(x, w) \in R^{\mathrm{sym}}_{\mathrm{adv}}$. Notice 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 necessarily 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, there 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.

D. 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 E.

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) $\beta(\mathrm{crs}(N)) := A_{\mathrm{crs}}(r_N)$ if $N \in \mathbf{N}_P$
7) $\beta(\mathrm{crs}(N^c)) := c$
8) $\beta(\mathrm{ZK}(\mathrm{crs}(N_1), t_1, t_2, N_2)) := A_{\mathrm{ZK}}(A_{\mathrm{crs}}(r_{N_1}), \beta(t_1), \beta(t_2), r_{N_2})$ if $N_1, N_2 \in \mathbf{N}_P$
9) $\beta(\mathrm{ZK}(\mathrm{crs}(t_0), t_1, t_2, N^s)) := s$
10) $\beta(\mathrm{garbageZK}(t_1, t_2, N^z)) := z$

Here $\mathbf{SymbExtr}(S, x)$ returns a witness w with $(x, w) \in R^{\mathrm{sym}}_{\mathrm{adv}}$ 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 $w := \tau^*(x)$ 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 $\mathrm{garbage}$ZK-term). Then the simulator aborts. If $(x, w) \notin R^{\mathrm{sym}}_{\mathrm{adv}}$, 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 $(x, w) \in R^{\mathrm{sym}}_{\mathrm{adv}}$ 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 V-B, we construct a sequence of simulators. The sequence is essentially the same (see Figure 3): Sim_1 differs from Sim by using a prover-oracle for constructing ZK-proofs in 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 8). 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 V-B. We even show something slightly stronger, namely that neither τ nor τ^* outputs an undeducable term:

Lemma 1 (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. $\diamond$

Now, as in Section V-B, we show Sim is Dolev-Yao iff Sim_f is Dolev-Yao. Later, we will also need preservation of the property that ZK-breaks do not occur.

Lemma 2 (Preservation of simulator-properties): • 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. $\diamond$

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 Lemma 2, 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 8). Thus, to conclude the proof of Lemma 2, 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 3 (No invalid symbolic witnesses): If Sim_3 is Dolev-Yao, then in rule 8), we have $(t_1, t_2) \in R^{\mathrm{sym}}_{\mathrm{honest}}$ with overwhelming probability. The same holds for Sim. $\diamond$

Lemma 4 (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. $\diamond$

Once we have these lemmas, Lemma 2 follows: We know from Lemma 1 that Sim_f is Dolev-Yao. We have already shown that this property carries over to Sim_3. Thus by Lemmas 3 and 4, $(\beta(t_1), \beta(t_2)) \in R^{\mathrm{comp}}_{\mathrm{honest}}$ in rule 8).

To show Lemma 3, 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 8) 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 8). Lemma 3 follows. Lemma 4 follows because we required that $R^{\mathrm{comp}}_{\mathrm{honest}}, R^{\mathrm{comp}}_{\mathrm{adv}}$ implement $R^{\mathrm{sym}}_{\mathrm{adv}}$ with usage restriction $R^{\mathrm{sym}}_{\mathrm{honest}}$; Definition 2 was designed to make Lemma 4 true.

Thus, Lemmas 3 and 4 hold, thus Lemma 2 follows. Since Sim_f is Dolev-Yao by Lemma 1, it follows with Lemma 2 that Sim is Dolev-Yao. It remains to show that Sim is indistinguishable.

Indistinguishability of Sim. As described in Section V-A, to show indistinguishability of Sim, the main subproof is to show (a) that $\beta(F(t_1, \dots, t_n)) = A_F(\beta(t_1), \dots, \beta(t_n))$ when the protocol computes $F(t_1, \dots, t_n)$. 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 ($(\beta(x), \beta(w) \in R^{\mathrm{sym}}_{\mathrm{honest}})$. Fortunately, we have already derived all needed facts: By Lemmas 1 and 3, $(x, w) \in R^{\mathrm{sym}}_{\mathrm{honest}}$ with overwhelming probability. And then by Lemma 4, $(\beta(x), \beta(w) \in R^{\mathrm{sym}}_{\mathrm{honest}}$.

To show (b), we need to show that no extraction failures occur. The approach for this is a bit roundabout, we first analyze Sim_2:

Lemma 5 (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 Lemma 4:

Lemma 6 (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 Lemma 5 is shown. From this, with Lemma 2 we get that ZK-breaks occur with negligible probability also for Sim_f. Based on this fact, we can show the following lemma:

Lemma 7 (No extraction failures): In the hybrid execution of Sim_f, extraction failures occur with negligible probability. ⋄

To see this, we use that ZK-breaks do not occur 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 Lemma 1, 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 Lemma 7.

Finally, from Lemmas 5 and 2 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 do not occur with Sim is necessary: We cannot directly show Lemma 5 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 [5, Thm. 1] we then immediately get Theorem 1.

VI. Zero-knowledge in the applied π-calculus

In [5] 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 [5] 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 3 (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 $\tilde{D} ::= \tilde{M} \mid \mathrm{isek}(\tilde{D}) \mid \mathrm{isenc}(\tilde{D}) \mid \mathrm{dec}(x_d, \tilde{D}) \mid \mathrm{fst}(\tilde{D}) \mid \mathrm{snd}(\tilde{D}) \mid \mathrm{ekof}(\tilde{D}) \mid \mathrm{equals}(\tilde{D}, \tilde{D}) \mid \mathrm{isvk}(\tilde{D}) \mid \mathrm{issig}(\tilde{D}) \mid \mathrm{verify_{sig}}(\tilde{D}, \tilde{D}) \mid \mathrm{vkof}(\tilde{D}) \mid \mathrm{iscrs}(\tilde{D}) \mid \mathrm{crsof}(\tilde{D}) \mid \mathrm{verify_{ZK}}(x_c, \tilde{D}) \mid \mathrm{iszk}(\tilde{D}) \mid \mathrm{getPub}(\tilde{D}) \mid \mathit{unstring_0}(\tilde{D}) \mid \mathit{unstring_1}(\tilde{D})$ and

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

(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 $\tilde{P}|Q \to^* E[\mathit{event}\ zk.\mathit{let}\ x = \mathrm{ZK}(t_1, t_2, t_3, t_4)\ \mathit{in}\ P_1\ \mathit{else}\ P_2]$ 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 [5, Thm. 4], we obtain:

Theorem 2 (Computational soundness of ZK proofs): Let $\tilde{P}$ be a closed valid process and A_F a computational implementation satisfying the implementation conditions from Section III. Then for any π-trace property[5] $\wp$, if P symbolically satisfies $\wp$, then P computationally satisfies $\wp$.

Automated analysis in Proverif. To show the usability of our modeling, we have analyzed a toy protocol in ProVerif [16]. Our 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.

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

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-malleablity of our ZK proofs ensures that the adversary cannot change m' in an existing proof. In the Appendix H, we prove that this relation satisfies definition 2, thus the abstraction is sound.

We express this protocol in the applied π-calculus:

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

[5]A π-trace property is essentially a prefix-closed set of sequences of events that are allowed to occur. See [5] for a precise definition and for the definition of "symbolically/computationally satisfying" a π-trace property.

This protocol can be directly encoded in ProVerif.[6] The definition of the destructor $\mathrm{verify}_{\mathrm{ZK}}$ depends on the relation $R^{\mathrm{sym}}_{\mathrm{adv}}$, 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 $\mathit{end}_A \Rightarrow \mathit{begin}_B$ and $\mathit{end}_B \Rightarrow \mathit{begin}_A$. To show that P also computationally satisfies that trace property, we need to show that P is valid. P satisfies the syntactic condition (Definition 3(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 $\mathit{checkzk}$ is a destructor that checks if its arguments are in $R^{\mathrm{sym}}_{\mathrm{honest}}$:

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.

VII. Conclusions

In this paper, 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 several 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 [8] to reflect this behavior. Second, recent work has investigated the soundness of cryptographic implementations on the source code level, e.g., in F# [10]. Developing a computational soundness result for ZK implementations would allow to safely use existing libraries that offer ZK implementations to higher-level protocols. Finally, 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 [21] to establish a composable soundness result for ZK proofs.

[6]We provide the ProVerif input online at http://www.infsec.cs.uni-saarland.de/~bendun/zk-cosp/

References

[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*, pages 104–115, New York, NY, USA, 2001. ACM Press.

[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*, pages 36–47, 1997.

[3] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

[4] M. Backes, D. Hofheinz, and D. Unruh. Cosp: A general framework for computational soundness proofs. In *ACM CCS 2009*, pages 66–78, November 2009. Preprint on IACR ePrint 2009/080.

[5] M. Backes, D. Hofheinz, and D. Unruh. A general framework for computational soundness proofs - or - the computational soundness of the applied pi-calculus. IACR ePrint Archive 2009/080, 2009.

[6] M. Backes, C. Hriţcu, and M. Maffei. Type-checking zero-knowledge. In *15th ACM Conference on Computer and Communications Security (CCS 2008)*, pages 357–370. ACM Press, 2008.

[7] 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, pages 130–148, Berlin, Heidelberg, 2010. Springer-Verlag.

[8] M. Backes, M. Maffei, and E. Mohammadi. Computationally sound abstraction and verification of secure multi-party computations. In K. Lodaya and M. Mahajan, editors, *FSTTCS*, volume 8 of *LIPIcs*, pages 352–363. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

[9] 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*, pages 158–169, May 2008.

[10] M. Backes, M. Maffei, and D. Unruh. Computationally sound verification of source code. In *ACM CCS 2010*, pages 387–398. ACM Press, October 2010. Preprint on IACR ePrint 2010/416.

[11] M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *Proc. 17th IEEE Computer Security Foundations Workshop (CSFW)*, pages 204–218, 2004.

[12] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations (extended abstract). In *Proc. 10th ACM Conference on Computer and Communications Security*, pages 220–230, 2003. Full version in IACR Cryptology ePrint Archive 2003/015, Jan. 2003, http://eprint.iacr.org/2003/015.

[13] 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. To appear.

[14] M. Backes and D. Unruh. Computational soundness of symbolic zero-knowledge proofs. *Journal of Computer Security*, 18(6):1077–1155, 2010. Preprint on IACR ePrint 2008/152.

[15] D. Basin, S. Mödersheim, and L. Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 2004.

[16] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pages 82–96, Cape Breton, Nova Scotia, Canada, June 2001. IEEE Computer Society.

[17] E. F. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *Proc. 11th ACM Conference on Computer and Communications Security*, pages 132–145. ACM Press, 2004.

[18] R. Canetti and J. Herzog. Universally composable symbolic analysis of mutual authentication and key exchange protocols. In *Proc. 3rd Theory of Cryptography Conference (TCC)*, volume 3876 of *LNCS*, pages 380–403. Springer, 2006.

[19] 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*, pages 354–368, Washington, DC, USA, 2008. IEEE Computer Society.

[20] V. Cortier and B. Warinschi. Computationally sound, automated proofs for security protocols. In *Proc. 14th European Symposium on Programming (ESOP)*, pages 157–171, 2005.

[21] 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.

[22] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[23] 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)*, pages 34–39, 1983.

[24] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–207, 1989.

[25] J. Groth. Simulation-sound nizk proofs for a practical language and constant size group signatures. In *In proceedings of ASIACRYPT '06, LNCS series*, pages 444–459. Springer-Verlag, 2006.

[26] J. Groth and R. Ostrovsky. Cryptography in the multi-string model. In A. Menezes, editor, *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 323–341. Springer, 2007. 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.

[27] D. Kähler, K. Ralf, and T. Wilke. Deciding properties of contract-signing protocols. Technical Report IFI 0409, CAU Kiel, 2004.

[28] R. Kemmerer, C. Meadows, and J. Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, 1994.

[29] 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, pages 186–200. Springer-Verlag, 2005.

[30] P. Laud. Semantics and program analysis of computationally secure information flow. In *Proc. 10th European Symposium on Programming (ESOP)*, pages 77–91, 2001.

[31] P. Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *Proc. 25th IEEE Symposium on Security & Privacy*, pages 71–85, 2004.

[32] H. Li and B. Li. An unbounded simulation-sound non-interactive zero-knowledge proof system for np. In *CISC*, pages 210–220, 2005.

[33] 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)*, volume 1055 of *LNCS*, pages 147–166. Springer, 1996.

[34] L. Lu, J. Han, Y. Liu, L. Hu, J.-P. Huai, L. Ni, and J. Ma. Pseudo trust: Zero-knowledge authentication in anonymous p2ps. *IEEE Trans. Parallel Distrib. Syst.*, 19:1325–1337, October 2008.

[35] M. Maffei and K. Pecina. Position paper: Privacy-aware proof-carrying authorization. In *PLAS 2011*, 2011. To appear.

[36] M. Merritt. *Cryptographic Protocols*. PhD thesis, Georgia Institute of Technology, 1983.

[37] D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proc. 1st Theory of Cryptography Conference (TCC)*, volume 2951 of *LNCS*, pages 133–151. Springer, 2004.

[38] L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Cryptology*, 6(1):85–128, 1998.

[39] 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, pages 543–, Washington, DC, USA, 1999. IEEE Computer Society.

[40] A. Sahai. Simulation-sound non-interactive zero knowledge. Technical report, IBM RESEARCH REPORT RZ 3076, 2001.

[41] S. Schneider. Security properties and CSP. In *Proc. 17th IEEE Symposium on Security & Privacy*, pages 174–187, 1996.

[42] C. Sprenger, M. Backes, D. Basin, B. Pfitzmann, and M. Waidner. Cryptographically sound theorem proving. In *Proc. 19th IEEE Computer Security Foundations Workshop (CSFW)*, pages 153–166, 2006.

APPENDIX

A. Protocol conditions

In this section we formally state the protocol conditions used in the main result of the paper. Summarized the protocol conditions require that we do not sent secret keys and that all relations used in zero-knowledge proofs are reasonable.

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 in [5] or appendix D) 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 (which was generated according to any protocol satisfying the protocol conditions above), outputs a term N, such that $S \vdash N$ and $(N, M) \in R^{\mathrm{sym}}_{\mathrm{adv}}$ or outputs $\perp$ 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.

We will call a protocol satisfying these constraints a safe protocol. The class of safe protocols is the set of all protocols which are safe.

B. Implementation conditions

Essentially the implementation conditions say that the zero-knowledge proof system is weakly symbolically-sound, the encryption scheme is IND-CCA secure, the signature scheme is strongly existentially unforgeable and several trivial conditions as e.g. that $\mathrm{unstring_i}$ is the inverse of $\mathrm{string_i}$.[7]

1) The implementation is an implementation according to Definition 8 (see Section D).
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 $\{0,1\}^k$ and output the sampled value tagged as nonce (here k is the security parameter).
4) If $A_{\mathrm{dec}}(\mathrm{dk}_N, m) \neq \bot$ 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 $A_{\mathrm{vkof}}(e) \neq \bot$, otherwise $A_{\mathrm{vkof}}(e) = \bot$.
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 $(\mathrm{KeyGen_{enc}}, \mathrm{ENC}, \mathrm{DEC})$ which is IND-CCA secure.
9) The implementations A_{vk},A_{sk},A_{sig}, and $A_{\mathrm{verify_{sig}}}$ belong to a signature scheme $(\mathrm{KeyGen_{sig}}, \mathrm{SIG}, \mathrm{VER_{sig}})$ 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 $A_{\mathrm{ekof}}(c) = \bot$.
14) Let $\mathrm{vk}, \mathrm{sk} \in \{0,1\}^*$ be a keypair, i.e. $(\mathrm{vk}, \mathrm{sk})$ is in the image of $\mathrm{KeyGen_{sig}}$, then holds for all $m, n \in \{0,1\}^*$: $A_{\mathrm{vkof}}(A_{\mathrm{sig}}(\mathrm{sk}, m, n)) = \mathrm{vk}$.
15) $A_{\mathrm{verify_{sig}}}(A_{\mathrm{vk}}(r), A_{\mathrm{sig}}(A_{\mathrm{sk}}(r), m, r')) = m$ for all r, r' nonces.
16) For all $p, s \in \{0,1\}^*$ we have that $A_{\mathrm{verify_{sig}}}(p, s) \neq \bot$ 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 $A_{\mathrm{crs}}, A_{\mathrm{ZK}}, A_{\mathrm{verify_{ZK}}}$ 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 $A_{\mathrm{verify_{ZK}}}(\mathrm{crsof}(z), z) \in \{\bot, z\}$, where $A_{\mathrm{verify_{ZK}}}(\mathrm{crsof}(z), z) = z$ 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 $\mathrm{verify_{ZK}}(\mathrm{crsof}(z), z) = \bot$.
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 $A_{\mathrm{verify_{ZK}}}(A_{\mathrm{crsof}}(z), z) = 1$.
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.
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 holds for signing keys and verification keys.

[7] The implementation conditions up to 19 follow the ones in [4]. The remaining ones are used to show soundness w.r.t. zero-knowledge.

C. Consistent environments and img_η

Definition 4: 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 {ek, dk, vk, sk, enc, sig, crs, garbageZK, garbageSig, garbageEnc, 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: [8]

- 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, $\mathrm{garbage}$ 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 $\mathrm{img}_\eta(\mathrm{ek}(N)) = p(d)$[9], then it follows that $A_{\mathrm{dec}}(d, \mathrm{img}_\eta(\mathrm{enc}(\mathrm{ek}(N), t, M))) = \mathrm{img}_\eta(t)$.

As long as the η used in the proof of lemma 11 stays consistent, it is possible to add many more properties to the list. In fact, not all of them are used in the soundness result itself, but to prove that the example relations given satisfy definition 2.

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

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

D. CoSP Review

We start with the definitions that capture symbolic notions of protocols and executions and proceed with their computational counterpart. After that, we introduce computational soundness as well as sufficient conditions for achieving computational soundness in CoSP.

Symbolic Model. We first introduce the notion of a symbolic model, which comprises basic concepts such as constructors, destructors, and deduction relations.

Definition 5 (Symbolic model): 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 symbols 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 $\underline{t}$, we write $D(\underline{t}) = \bot$. A *deduction relation* $\vdash$ over a message type $\mathbf{T}$ is a relation between $2^{\mathbf{T}}$ and $\mathbf{T}$.

A *symbolic model* $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$ 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 $\mathbf{T}$.
To unify notation, 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.

Protocols in CoSP essentially constitute a tree with distinguished nodes for computations, input, output and branches.

Definition 6 (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.

Assigning each nondetrerministic node a probability distribution over its successors yields the notion of a *probabilistic*

CoSP protocol. A probabilistic CoSP protocol is called *efficient* if the lengths of all identifiers in N are polynomially bounded, and the labels of N can be computed in polynomial time.

The symbolic execution of a CoSP protocol for a given symbolic model consists of a sequence of triples (S, ν, f) where S represents the knowledge of the adversary, ν represents the current node identifier in the protocol, and f represents a partial function mapping already processed node identifiers to messages.

Definition 7 (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: $S_1 =, \nu_1$ 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 $m := \text{eval}_F(\underline{\tilde{t}}) \neq \perp$, ν' is the yes-successor of ν in Π_s, and $f' = f(\nu := m)$. If $m = \perp$, 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.

Computational Model. To define the corresponding computational execution of a CoSP protocol, we have to introduce computational implementations for a symbolic model.

Definition 8 (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.

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 9 (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 $\text{Nodes}^p_{\mathbf{M},A,\Pi_p,E}(k)$, 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 patrial 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. All cases are similar to the ones of the symbolic execution, but here we use the computational implementation instead of the symbols (constructors and destructors are executed, nonces are sampled once and their result is cached). A list of the full detailed cases can be found in [4].
  - 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.

Computational soundness. We finally introduce trace properties and computational soundness in CoSP.

Definition 10 (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 $\text{Nodes}^p_{\mathbf{M},A,\Pi_p,E}(k) \in \mathcal{P}$.

Definition 11 (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 introduce the concept of a simulator. Simulators with specific properties have been shown to constitute a sufficient condition for achieving computational soundness. We will exploit this in our main soundness theorem.

Definition 12 (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 thus constitutes a technique to map symbolic executions onto computational executions by translating the symbols to bitstrings and vice versa. This is realized by an hybrid execution as follows.

Definition 13 (Hybrid execution): Let Π_p be a probabilistic CoSP protocol, and let Sim be a simulator. We define a probability distribution $\text{H-Trace}_{\mathbf{M},\Pi_p,\mathrm{Sim}}(k)$ 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. The computation nodes are treated as in the symbolic execution and for the input, output and control nodes we use the simulator. Nondeterministic nodes are sampled according to the annotated probability distribution (Full details of the cases can be found in [4]).
  - 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 $\text{H-Nodes}_{\mathbf{M},\Pi_p,\mathrm{Sim}}(k)$. We call this distribution the hybrid node trace.

The existence of a simulator that fulfills two distinguished properties, DY-style and indistinguishable, has been shown sufficient for needs to fulfill to establish computational soundness. DY-style means that the adversary should not be able to sent terms which cannot be deduced from the adversary's knowledge. Indistinguishable means that an adversary should not be able to distinguish a hybrid execution (which involves the simulator and the symbolic protocol) from an computational execution. We speak of a *good* simulator if both properties are fulfilled.

Definition 14 (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.

Theorem 3 (Good simulator implies soundness [4]): 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.

E. Detailed Soundness Proof

In this section, we give the complete soundness proof. We enumerated the lemmas as in the body of the paper. The goal is to prove the following Theorem:

Theorem 1 (Computational soundness of ZK proofs): Every good implementation A is a computationally sound implementation of the symbolic model $\mathbf{M}$ (defined in the appendix A) for the class of safe protocols. $\diamond$

To prove the Theorem, we will use Theorem 3. 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 3 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*:*

- $\beta(N) := r_N$ if $N \in \mathbf{N}_P$
- $\beta(N^m) := m$
- $\beta(\mathrm{enc}(\mathrm{ek}(N), t, M)) := A_{\mathrm{enc}}(A_{\mathrm{ek}}(r_N), \beta(t), r_M)$ if $M \in \mathbf{N}_P$
- $\beta(\mathrm{enc}(\mathrm{ek}(M), t, N^m)) := m$ if $M \in \mathbf{N}_P$
- $\beta(\mathrm{ek}(N)) := A_{\mathrm{ek}}(r_N)$ if $N \in \mathbf{N}_P$
- $\beta(\mathrm{ek}(N^m)) := m$
- $\beta(\mathrm{dk}(N)) := A_{\mathrm{dk}}(r_N)$ if $N \in \mathbf{N}_P$
- $\beta(\mathrm{dk}(N^m)) := d$ such that $\tau(d) = \mathrm{dk}(N^m)$ was computed earlier
- $\beta(\mathrm{sig}(\mathrm{sk}(N), t, M)) := A_{\mathrm{sig}}(A_{\mathrm{sk}}(r_N), \beta(t), r_M)$ if $N, M \in \mathbf{N}_P$
- $\beta(\mathrm{sig}(\mathrm{sk}(M), t, N^s)) := s$
- $\beta(\mathrm{vk}(N)) := A_{\mathrm{vk}}(r_N)$ if $N \in \mathbf{N}_P$
- $\beta(\mathrm{vk}(N^m)) := m$
- $\beta(\mathrm{sk}(N)) := A_{\mathrm{sk}}(r_N)$ if $N \in \mathbf{N}_P$
- $\beta(\mathrm{sk}(N^m)) := s$ such that $\tau(s) = \mathrm{sk}(N^m)$ was computed earlier
- $\beta(\mathrm{crs}(N)) := A_{\mathrm{crs}}(r_N)$ if $N \in \mathbf{N}_P$
- $\beta(\mathrm{crs}(N^c)) := c$
- $\beta(\mathrm{ZK}(\mathrm{crs}(N_1), t_1, t_2, N_2)) := A_{\mathrm{ZK}}(A_{\mathrm{crs}}(r_{N_1}), \beta(t_1), \beta(t_2), r_{N_2})$ if $N_1, N_2 \in \mathbf{N}_P$
- $\beta(\mathrm{ZK}(\mathrm{crs}(t_0), t_1, t_2, N^s)) := s$
- $\beta(\mathrm{pair}(t_1, t_2)) := A_{\mathrm{pair}}(\beta(t_1), \beta(t_2))$
- $\beta(\mathrm{string}_0(t)) := A_{\mathrm{string}_0}(\beta(t))$
- $\beta(\mathrm{string}_1(t)) := A_{\mathrm{string}_1}(\beta(t))$
- $\beta(\mathrm{empty}) := A_{\mathrm{empty}}()$
- $\beta(\mathrm{garbage}(N^c)) := c$
- $\beta(\mathrm{garbageEnc}(t, N^c)) := c$
- $\beta(\mathrm{garbageSig}(t, N^s)) := s$
- $\beta(\mathrm{garbageZK}(t_1, t_2, N^z)) := z$
- $\beta(t) := \bot$ 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)

- $\tau(r) := N$ if $r = r_N$ for some $N \in \mathcal{N}$ and N occurred in a term sent from Π^C
- $\tau(r) := N^r$ if r is of type nonce
- $\tau(c) := \mathrm{enc}(\mathrm{ek}(M), t, N)$ if c has earlier been output by $\beta(\mathrm{enc}(\mathrm{ek}(M), t, N))$ for some $M \in \mathbf{N}, N \in \mathbf{N}_P$
- $\tau(c) := \mathrm{enc}(\mathrm{ek}(M), \tau(m), N^c)$ if c is of type ciphertext and $\tau(A_{\mathrm{ekof}}(c)) = \mathrm{ek}(M)$ for some $M \in \mathbf{N}_P$ and $m := A_{\mathrm{dec}}(A_{\mathrm{dk}}(r_N), c) \neq \bot$
- $\tau(c) := \mathrm{garbageEnc}(\tau(A_{\mathrm{ekof}}(c)), N^c)$ if c is of type ciphertext
- $\tau(c) := \mathrm{ek}(N)$ 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
- $\tau(s) := \mathrm{sig}(\mathrm{sk}(M), t, N)$ if s has earlier been output by $\beta(\mathrm{sig}(\mathrm{sk}(M), t, N))$ for some $M, N \in \mathbf{N}_P$
- $\tau(s) := \mathrm{sig}(\mathrm{sk}(M), \tau(m), N^s)$ if s is of type signature and $\tau(A_{\mathrm{vkof}}(s)) = \mathrm{vk}(M)$ 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
- $\tau(s) := \mathrm{vk}(N)$ if $s = A_{\mathrm{vk}}(r_N)$ for some N that occurred in a subterm of the form $\mathrm{vk}(N)$ or $\mathrm{sk}(N)$ before
- $\tau(s) := \mathrm{vk}(N^s)$ if s is of type verification key
- $\tau(s) := \mathrm{sk}(N)$ if $s = A_{\mathrm{sk}}(r_N)$ for some N that occurred in a subterm of the form $\mathrm{vk}(N)$ or $\mathrm{sk}(N)$ before
- $\tau(s) := \mathrm{sk}(N^c)$ 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.

- $\tau(z) := \text{garbageZK}(c, x, N^z)$ if z is of type zero-knowledge proof, $c := \tau(A_{\text{crsof}}(z))$ and $x := \tau(A_{\text{getPub}}(z))$.
- $\tau(m) := \text{pair}(\tau(A_{\text{fst}}(m)), \tau(A_{\text{snd}}(m)))$ if m is of type pair
- $\tau(m) := \text{string}_0(\tau(m'))$ if m is of type payload-string and $m' := A_{\text{unstring}_0}(m) \neq \perp$
- $\tau(m) := \text{string}_1(\tau(m'))$ if m is of type payload-string and $m' := A_{\text{unstring}_1}(m) \neq \perp$
- $\tau(m) := \text{empty}$ if m is of type payload-string and $m = A_{\text{empty}}()$
- $\tau(m) := \text{garbage}(N^m)$ otherwise

When an extraction-failure on (N, z, m_x) occurs (i.e., when in the computation of τ, $\mathbf{SymbExtr}(S, x)$ returns $w = \perp$), the simulator computes $(\text{crs}, \text{simtd}, \text{extd}) \leftarrow \mathbf{K}(1^\eta; r_N)$ to get the extraction trapdoor extd corresponding to $\text{crs} = A_{\text{crs}}(r_N)$. Then the simulator invokes $m_w := \mathbf{E}(m_x, z, \text{extd})$ and computes $x := \tau(m_x)$ as well as $w := \tau^*(m_w)$. If $(x, w) \notin R^{\text{sym}}_{\text{adv}}$, 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 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.

1) 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 $\mathcal{O}_{\text{ZK}}$ that internally picks $(\text{crs}, \text{simtd}, \text{extd}) \leftarrow \mathbf{K}(1^\eta)$ and that responds to three kinds of queries: Upon a $(\texttt{crs})$-query, it returns crs, and upon a $(\texttt{prove}, x, w)$-query, it returns $\mathbf{P}(x, w, \text{crs})$ if $(x, w) \in R^{\text{comp}}_{\text{honest}}$ and $\perp$ otherwise. Upon a $(\texttt{extd})$-query, it returns extd. For each $N \in \mathbf{N}_P$, Sim_1 maintains an instance $\mathcal{O}^N_{\text{ZK}}$ of $\mathcal{O}_{\text{ZK}}$. Then Sim_1 computes $\beta(\text{crs}(N))$ with $N \in \mathbf{N}_P$ as $\beta(\text{crs}(N)) := \mathcal{O}^N_{\text{ZK}}(\texttt{crs})$, and Sim_1 computes $\beta(\text{ZK}(\text{crs}(N_1), t_1, t_2, N_2))$ with $N_1, N_2 \in \mathbf{N}_P$ as $\beta(\text{ZK}(\text{crs}(N_1), t_1, t_2, N_2)) := \mathcal{O}^N_{\text{ZK}}(\texttt{prove}, \beta(t_1), \beta(t_2))$. In case of an extraction-failure, Sim_1 performs a $(\texttt{extd})$-query to get extd. (Here and in the descriptions of $\text{Sim}_2, \dots, \text{Sim}_5, \text{Sim}_f$, 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 $\tau(z) = \text{crs}(N)$ for $N \in \mathbf{N}_P$, instead of checking $z = A_{\text{crs}}(r_N)$, Sim_1 checks whether z is equal to the $(\texttt{crs})$-query outcomes for all oracles $\mathcal{O}^N_{\text{ZK}}$ which have been used so far.
- We define Sim_2 like Sim_1, except that we replace the oracle $\mathcal{O}_{\text{ZK}}$ by an oracle $\mathcal{O}_{\text{sim}}$. That oracle behaves like $\mathcal{O}_{\text{ZK}}$, except that upon a $(\texttt{prove}, x, w)$-query, it returns $\mathbf{S}(x, \text{crs}, \text{simtd})$ if $(x, w) \in R^{\text{comp}}_{\text{honest}}$ and $\perp$ otherwise.
- We define Sim_3 like Sim_2, except that we replace the oracle $\mathcal{O}_{\text{sim}}$ by an oracle $\mathcal{O}'_{\text{sim}}$. That oracle behaves like $\mathcal{O}_{\text{sim}}$, except that upon a $(\texttt{prove}, x, w)$-query, it returns $\mathbf{S}(x, \text{crs}, \text{simtd})$ (even if $(x, w) \notin R^{\text{comp}}_{\text{honest}}$). Therefore the simulator only queries $(\textit{prove}, x)$ 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 $\mathcal{O}_{\text{enc}}$ that internally picks $(\text{ek}, \text{dk}) \leftarrow \text{KeyGen}_{\text{enc}}(1^\eta)$ and that responds to three kinds of queries: Upon an $(\texttt{ek})$-query, it returns ek. Upon a $(\texttt{enc}, m)$-query, it returns $\text{ENC}(\text{ek}, m)$. Upon a $(\texttt{dec}, c)$-query, it returns $\text{DEC}(\text{dk}, c)$. Sim_4 maintains an instance $\mathcal{O}^N_{\text{enc}}$ for each $N \in \mathbf{N}_P$. Then Sim_4 computes $\beta(\text{ek}(N))$ with $N \in \mathbf{N}_P$ as $\beta(\text{ek}(N)) := \mathcal{O}^N_{\text{enc}}(\texttt{ek})$. And it computes $\beta(\text{enc}(\text{ek}(N), t, M))$ with $N, M \in \mathbf{N}_P$ as $\beta(\text{enc}(\text{ek}(N), t, M)) := \mathcal{O}^N_{\text{enc}}(\texttt{enc}, \beta(t))$. And it computes $\beta(\text{dk}(N)) := \perp$. And in the computation of $\tau(c)$ for c of type ciphertext, the computation of $A_{\text{dec}}(A_{\text{dk}}(r_N), c)$ is replaced by $\mathcal{O}^N_{\text{enc}}(\texttt{dec}, c)$.
  In the definition of $\tau(c) = \text{ek}(N)$ and $\tau(c) = \text{dk}(N)$ for $N \in \mathbf{N}_P$, instead of checking $c = A_{\text{ek}}(r_N)$ and $c = A_{\text{dk}}(r_N)$, Sim_4 checks whether c is equal to the corresponding query outcomes for all oracles $\mathcal{O}^N_{\text{enc}}$ which have been used so far.
- We define Sim_5 like Sim_4, except that we replace the oracle $\mathcal{O}_{\text{enc}}$ by an oracle $\mathcal{O}_{\text{fake}}$. That oracle behaves like $\mathcal{O}_{\text{enc}}$, except that upon an $(\texttt{enc}, x)$-query, it returns $\text{ENC}(\text{ek}, 0^{|x|})$.
- We define Sim_f like Sim_5, but we change β to use signing oracles instead of computing $A_{\text{vk}}, A_{\text{sk}}, A_{\text{sig}}$. More precisely, we assume an oracle $\mathcal{O}_{\text{sig}}$ that internally picks $(\text{vk}, \text{sk}) \leftarrow \text{KeyGen}_{\text{sig}}(1^\eta)$ and that responds to two kinds of queries: Upon a $(\texttt{vk})$-request, it returns vk, and upon a $(\texttt{sig}, m)$-request, it returns $\text{SIG}(\text{sk}, m)$. Sim_f maintains an instance $\mathcal{O}^N_{\text{sig}}$ for each $N \in \mathbf{N}_P$. Then Sim_f computes $\beta(\text{vk}(N))$ with $N \in \mathbf{N}_P$ as $\beta(\text{vk}(N)) := \mathcal{O}^N_{\text{sig}}(\texttt{vk})$. And $\beta(\text{sk}(N))$ with $N \in \mathbf{N}_P$ as $\beta(\text{sk}(N)) := \perp$. And $\beta(\text{sig}(\text{sk}(N), t, M))$ with $N, M \in \mathbf{N}_P$ as $\beta(\text{sig}(\text{sk}(N), t, M)) := \mathcal{O}^N_{\text{sig}}(\texttt{sig}, \beta(t))$.
  In the definition of $\tau(c) = \text{vk}(N)$ and $\tau(c) = \text{sk}(N)$

for $N \in \mathbf{N}_P$, instead of checking $c = A_{\text{vk}}(r_N)$ and $c = A_{\text{sk}}(r_N)$, Sim_f checks whether c is equal to the corresponding query outcomes for all oracles $\mathcal{O}^N_{\text{sig}}$ which have been used so far.

2) 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 8: 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 $S \nvdash u$.

Then there is a term t_{bad} and a context D such that D can be obtained by the following grammar:

$$\begin{aligned} D ::= &\ \square \mid \text{pair}(t, D) \mid \text{pair}(D, t) \mid \text{enc}(\text{ek}(N), D, M) \\ &\mid \text{enc}(D, t, M) \mid \text{sig}(\text{sk}(M), D, M') \\ &\mid \text{ZK}(t, D, t', M) \mid \text{ZK}(D, t, t', M) \\ &\mid \text{garbageEnc}(D, M) \mid \text{garbageSig}(D, M) \\ &\mid \text{garbageZK}(D, t, M) \mid \text{garbageZK}(t, D, M) \\ &\quad \text{with } M, M' \in \mathbf{N}_E, t, t' \in \mathbf{T} \end{aligned}$$

with $u = D[t_{bad}]$ such that $S \nvdash t_{bad}$ and such that one of the following holds:

1) $t_{bad} \in \mathcal{N}$
2) $t_{bad} = \text{enc}(p, m, N)$ with $N \in \mathbf{N}_P$
3) $t_{bad} = \text{sig}(k, m, N)$ with $N \in \mathbf{N}_P$
4) $t_{bad} = \text{ZK}(\text{crs}(M), x, w, N)$ with $M, N \in \mathbf{N}_P$
5) $t_{bad} = \text{sig}(\text{sk}(N), m, M)$ with $N \in \mathbf{N}_P$, $M \in \mathbf{N}_E$
6) $t_{bad} = \text{crs}(N)$ with $N \in \mathbf{N}_P$
7) $t_{bad} = \text{ek}(N)$ with $N \in \mathbf{N}_P$
8) $t_{bad} = \text{vk}(N)$ with $N \in \mathbf{N}_P$
9) $t_{bad} = \text{sk}(N)$ with $N \in \mathbf{N}_P$
10) $t_{bad} = \text{dk}(N)$ 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: "$u \in \{\text{ek}(N), \text{vk}(N), \text{crs}(N), \text{dk}(N), \text{sk}(N)\}$ with $N \notin \mathbf{N}_P$"
Let $u = C(N)$ for $C \in \{\text{ek}, \text{vk}, \text{crs}, \text{dk}, \text{sk}\}$. 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: "$u \in \{\text{ek}(N), \text{vk}(N), \text{crs}(N), \text{dk}(N), \text{sk}(N)\}$ with $N \in \mathbf{N}_P$"
Then the claim is fulfilled with $D := \square$ and $t_{bad} = u$.

Case 3: "$u = \text{garbage}(u_1)$"
By protocol condition 2 no garbage term is generated by the protocol. Therefore there is a $c \in \{0,1\}^*$ such that $\tau(c) = \text{garbage}(N^c) = u$. But this means that $S \vdash u$, contradicting the premise of the lemma.

Case 4: "$u = \text{garbageEnc}(u_1, u_2)$ or $u = \text{garbageSig}(u_1, u_2)$"
By protocol condition 2 no garbage term is generated by the protocol. So there exists a $c \in \{0,1\}^*$ with $\tau(c) = \text{garbageEnc}(u_1, N^c)$ or $\tau(c) = \text{garbageSig}(u_1, N^c)$. 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 $\text{garbageEnc}(D', N^c)$, respectively $\text{garbageSig}(D', N^c)$, shows the claim.

Case 5: "$u = \text{garbageZK}(u_1, u_2, u_3)$"
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 $\text{garbageZK}(D', u_2, u_3)$ to satisfy the lemma. In the other case we apply the induction hypothesis to u_2 leading to context $\text{garbageZK}(u_1, D', u_3)$ and t_{bad}.

Case 6: "$u = \text{pair}(u_1, u_2)$"
Since $S \nvdash u$ there is an $i \in \{1, 2\}$ such that $S \nvdash u_i$. Let D be the context and t_{bad} the term given by applying the induction hypothesis to u_i. Then $D_1 := \text{pair}(D, M)$ or $D_2 := \text{pair}(M, D)$ is the context for the term u depending on i with the same term t_{bad}.

Case 7: "$u = \text{empty}$"
This case cannot happen because $S \vdash \text{empty}$, so the premise of the lemma is not fulfilled.

Case 8: "$u = \text{string}_0(u_1)$ or $u = \text{string}_1(u_1)$"
Again the premise is not fulfilled since inductively $S \vdash u_1$ with base case $u_1 = \text{empty}$ and therefore $S \vdash \text{string}_i(u_1)$ 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 $D := \square$ 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: "$u = \text{enc}(u_1, u_2, N)$ with $N \in \mathbf{N}_P$"
The lemma is satisfied by $t_{bad} = u$ and $D = \square$.

Case 13: "$u = \text{enc}(u_1, u_2, u_3)$ with $u_3 \notin \mathbf{N}_P$ and $S \nvdash u_1$"

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 $\tau(c) = \mathrm{enc}(\mathrm{ek}(M), u_2, N^c) = u$ for some $M \in \mathbf{N}_P$. Apply the induction hypothesis to u_1 getting t_{bad} and context D we can define $D' := \mathrm{enc}(D, u_2, N^c)$ fulfilling the claim of the lemma with t_{bad}.

Case 14: "$u = \mathrm{enc}(u_1, u_2, u_3)$ 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 $\tau(c) = \mathrm{enc}(\mathrm{ek}(M), u_2, N^c) = u$ for some $M \in \mathbf{N}_P$. Since $S \vdash u_1$, $S \vdash N^c$, and $S \nvdash u$, it follows that $S \nvdash u_2$. Let D be the context and t_{bad} be the term resulting by the induction hypothesis applied to u_2. Then $D' := \mathrm{enc}(\mathrm{ek}(M), D, N^c)$ 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 $D := \square$ and $t_{bad} = u$.

Case 16: "$u = \mathrm{sig}(\mathrm{sk}(N), u_1, u_3)$ 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 $D := \square$ 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 $\mathrm{sk}(N)$ 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 $\tau(s) = \mathrm{sig}(\mathrm{sk}(M), u_2, N^s) = u$ with $M \in \mathbf{N}$. Because u_1 has not the form $\mathrm{sk}(N)$ for any $N \in \mathbf{N}_P$ follows that $M \in \mathbf{N}_E$, so $S \vdash M$ and therefore $S \vdash \mathrm{sk}(M)$. 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 $D' := \mathrm{sig}(\mathrm{sk}(M), D, N^s)$ 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 $D := \square$ 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 $D := \mathrm{ZK}(\square, u_1, u_2, N)$ 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 9: 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) $t' = \mathrm{sig}(\mathrm{sk}(N), t, M)$ 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) $t' = \mathrm{enc}(\mathrm{ek}(N^e), t, M)$ 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 $t' = \mathrm{enc}(\mathrm{ek}(N), t, M)$ 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: $S \vdash \mathrm{sig}(\mathrm{sk}(N), t, M) = t'$. 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: $S \vdash \mathrm{enc}(\mathrm{ek}(N^e), t, M)$. 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 10 (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 8 with context $C = \square$ 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 9 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: "$t_{bad} = \mathrm{enc}(p, m, N)$ 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 9 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 $t_{bad} = \mathrm{enc}(p, m, N)$ 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 9 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 9, 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: "$t_{bad} = \mathrm{sig}(\mathrm{sk}(N), m', M)$ 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. [10]

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 $\mathcal{O}^N_{\mathrm{enc}}$ 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 $\mathrm{enc}(\mathrm{ek}(N), \cdot, \cdot)$. 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 9 that either $S \vdash \mathrm{ek}(N)$ or $S \vdash \mathrm{enc}(\mathrm{ek}(N), \cdot, \cdot)$. In the latter case $S \vdash \mathrm{ek}(N)$ using destructor ekof. So $S \not\vdash t_{bad}$ is impossible.

Case 8: "$t_{bad} = \mathrm{vk}(N)$ 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 $\mathrm{vk}(N)$, $\mathrm{sk}(N)$, or $\mathrm{sig}(\mathrm{sk}(N), \cdot, \cdot)$. The case $\mathrm{sk}(N)$ corresponds to $\mathrm{dk}(N)$ and thus it is impossible. In the remaining cases, it follows that $S \vdash \mathrm{vk}(N)$ (using vkof constructor on the signature). So $S \not\vdash t_{bad}$ is impossible.

Case 9: "$t_{bad} = \mathrm{sk}(N)$ with $N \in \mathbf{N}_P$".
If $t_{bad} = \mathrm{sk}(N)$ 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 $S \not\vdash t_{bad}$ can only be the case with negligible probability.

Hence, $S \nvdash \tau(m)$ happens only with negligible probability. ■

3) 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.

[10]Note that an adversary against this property is allowed to use the extraction trapdoor. The same holds the property of CCA.

Lemma 11 (Relating the relations): Let $R^{\text{comp}}_{\text{honest}}, R^{\text{comp}}_{\text{adv}}$ be relations implementing $R^{\text{sym}}_{\text{adv}}$ with usage restriction $R^{\text{sym}}_{\text{honest}}$.

1) In the hybrid execution of Sim and Sim_3 it holds with overwhelming probability: If $(x, w) \in R^{\text{sym}}_{\text{honest}}$ and x, w occur as node annotation of a ZK node in the execution, then it holds $(\beta(x), \beta(w)) \in R^{\text{comp}}_{\text{honest}}$.
2) In the hybrid execution of Sim_2 it holds with overwhelming probability: If $(m_x, m_w) \in R^{\text{comp}}_{\text{adv}}$ for some bitstrings m_x, m_w, then it holds $(\tau(m_x), \tau^*(m_w)) \in R^{\text{sym}}_{\text{adv}}$.

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 $t_1, \dots, t_n$ 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 $t = C(t_1, \dots, t_n, N^m)$ define $\eta(t) := m$ for all C as stated in definition 4.[11]
- For $t = \text{crs}(N)$ with $N \in \mathbf{N}_P$ define $\eta(t)$ to be the crs produced by the oracle $\mathcal{O}^N_{\text{ZK}}$.
- For $t = \text{ZK}(\text{crs}(N), x, w, M)$ with $N, M \in \mathbf{N}_P$ define $\eta(t)$ to be proof produced by $\mathcal{O}^N_{\text{ZK}}$ 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 $\text{crs}(t)$ nor in $\text{ZK}(c, x, w, t)$ 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 $\text{crs}(N)$. 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 $\eta(\text{dk}(N)) = \eta(\text{dk}(M))$ for $N \in \mathbf{N}_P, M \in \mathbf{N}_E$. By protocol condition 2, it follows that $\text{dk}(M) = \text{dk}(N^d)$ for some $d \in \{0,1\}^*$, $\text{dk}(N^d)$ was subterm of an output of τ, and d was not output of β earlier (otherwise d would have been parsed to $\text{dk}(N)$). So the adversary used either no input or only encryptions plus the encryption key to compute $\text{dk}(N)$. 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 $\eta(\text{ek}(N)) = \eta(\text{ek}(M))$ for $N \in \mathbf{N}_P, M \in \mathbf{N}_E$. By protocol condition 2, it follows that $\text{ek}(M) = \text{ek}(N^e)$ for some $e \in \{0,1\}^*$. But then τ parsed e to $\text{ek}(N^e)$, so neither $\text{ek}(N)$ nor $\text{dk}(N)$ 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 bistring. 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 $\text{garbage}(t)$ 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[12], it suffices to show that if $(x, w) \in R^{\text{sym}}_{\text{honest}}$ then there is a consistent $\eta \in \mathcal{E}$ such that $(\text{img}_\eta(x), \text{img}_\eta(w)) = (\beta(x), \beta(w))$ 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 analoguous 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.

[11]They are {ek, dk, vk, sk, enc, sig, crs, garbageZK, garbageSig, garbageEnc, garbage}

[12]The part we will use here says $(x, w) \in R^{\text{sym}}_{\text{honest}}$ and $\text{img}_\eta(x) \neq \bot \neq \text{img}_\eta(w)$ implies $(\text{img}_\eta(x), \text{img}_\eta(w)) \in R^{\text{comp}}_{\text{honest}}$.

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:

- "$t = N$ 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 $\eta(N^n) = n = \beta(N^n)$.
- "$t \in \{\mathrm{ek}(u), \mathrm{dk}(u), \mathrm{vk}(u), \mathrm{sk}(u)\}$ 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 $C \in \{\mathrm{ek}, \mathrm{dk}, \mathrm{vk}, \mathrm{sk}\}$ 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 $\mathcal{O}_{\mathrm{ZK}}^N$ 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 $N \in \mathbf{N}_E$.
- "$t = \mathrm{enc}(u_1, u_2, u_3)$". 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 $\beta(t) = A_{\mathrm{enc}}(\beta(u_1), \beta(u_2), r_N) = A_{\mathrm{enc}}(\mathrm{img}_\eta(u_1), \mathrm{img}_\eta(u_2), r_N)$ 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 $\beta(t) = A_{\mathrm{enc}}(\mathrm{img}_\eta(u_1), \mathrm{img}_\eta(u_2), \mathrm{img}_\eta(N)) = \mathrm{img}_\eta(t)$.
- "$t = \mathrm{sig}(u_1, u_2, u_3)$" 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 $u_1 = \mathrm{sk}(M)$ for some $M \in \mathbf{N}_P$. Then, it holds $\beta(t) = A_{\mathrm{sig}}(A_{\mathrm{sk}}(r_M), \beta(u_2), r_N)$. Again, $r_N = \mathrm{img}_\eta(N)$; the same holds for M. Since $\beta(\mathrm{sk}(M)) = A_{\mathrm{sk}}(M)$, it follows by induction hypothesis that $A_{\mathrm{sk}}(r_M) = \mathrm{img}_\eta(\mathrm{sk}(M))$. In total, it holds $\beta(t) = A_{\mathrm{sig}}(\mathrm{img}_\eta(\mathrm{sk}(M)), \mathrm{img}_\eta(u_2), \mathrm{img}_\eta(N)) = \mathrm{img}_\eta(t)$.
- "$t = \mathrm{pair}(u_1, u_2)$ where $u_1, u_2 \in \mathbf{T}$". By induction hypothesis, it follows $\beta(u_i) = \mathrm{img}_\eta(u_i)$. Thus, it holds $\beta(\mathrm{pair}(u_1, u_2)) = A_{\mathrm{pair}}(\beta(u_1), \beta(u_2)) = A_{\mathrm{pair}}(\mathrm{img}_\eta(u_1), \mathrm{img}_\eta(u_2)) = \mathrm{img}_\eta(\mathrm{pair}(u_1, u_2))$.
- "$t \in \{\mathrm{string}_0(u), \mathrm{string}_1(u), \mathrm{empty}\}$ with $u \in \mathbf{T}$". These cases are analogue to the case $t = \mathrm{pair}(u_1, u_2)$.
- "$t \in \{\mathrm{garbage}(u_1), \mathrm{garbageSig}(u_1, u_2, u_3), \mathrm{garbageEnc}(u_1, u_2), \mathrm{garbageZK}(u_1, u_2, u_3, u_4)\}$ 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 $\mathrm{img}_\eta(t) = \eta(t) = m$, 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 $m = \mathrm{img}_\eta(\tau(m))$ holds. Then it follows by definition 2[13] $(m_x, m_w) \in R_{\mathrm{adv}}^{\mathrm{comp}} \implies (\mathrm{img}_\eta(\tau(m_x)), \mathrm{img}_\eta(\tau(m_w))) \in R_{\mathrm{adv}}^{\mathrm{comp}} \implies (\tau(m_x), \tau^*(m_w)) \in R_{\mathrm{adv}}^{\mathrm{sym}}$.

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 $m = \mathrm{img}_\eta(\tau(m))$ 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 $\mathrm{img}_\eta(\tau(m)) = r_N = m$ where the last equality holds because of the definition of τ.
- $\tau(m) = N^m$
  Then by construction of η holds that $\mathrm{img}_\eta(\tau(m)) = \mathrm{img}_\eta(N^m) = \eta(N^m) = m$.
- $\tau(m) = \mathrm{enc}(\mathrm{ek}(M), t, N)$ for some $M \in \mathbf{N}, N \in \mathbf{N}_P$
  By definition of η holds that $\mathrm{img}_\eta(\tau(m)) = \mathrm{img}_\eta(\mathrm{enc}(\mathrm{ek}(M), t, N)) = A_{\mathrm{enc}}(A_{\mathrm{ek}}(\eta(M)), \mathrm{img}_\eta(t), \eta(N))$. 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 $\mathrm{ek}(M)$ is the same, too. By determinism of the implementations (implementation condition 1) follows that the output is m.
- $\tau(s) = \mathrm{sig}(\mathrm{sk}(M), t, N)$ for some $M, N \in \mathbf{N}_P$
  This case is analogue to the one of $\mathrm{enc}(\mathrm{ek}(M), t, N)$ for some $M \in \mathbf{N}, N \in \mathbf{N}_P$.
- $\tau(m) = \mathrm{ek}(N)$ for some $N \in \mathbf{N}_P$
  By definition of τ, it follows $m = A_{\mathrm{ek}}(r_N)$. On the other hand, it holds that $\mathrm{img}_\eta(\mathrm{ek}(N)) = A_{\mathrm{ek}}(\eta(N)) = A_{\mathrm{ek}}(r_N) = m$ by construction.
- $\tau(m) \in \{\ \mathrm{vk}(N), \mathrm{sk}(N), \mathrm{dk}(N)\}$ for some $N \in \mathbf{N}_P$

[13]At this point we use $(\mathrm{img}_\eta(x), \mathrm{img}_\eta(w)) \in R_{\mathrm{adv}}^{\mathrm{comp}}$ implies $(x, w) \in R_{\mathrm{adv}}^{\mathrm{sym}}$.

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.
- $\tau(m) \in \{\mathrm{enc}(\mathrm{ek}(M), t, N^m),\ \mathrm{ek}(N^m),\ \mathrm{dk}(N^m),\ \mathrm{garbageEnc}(t, N^m),\ \mathrm{sig}(\mathrm{sk}(M), t, N^m),\ \mathrm{garbageSig}(t, N^m),\ \mathrm{vk}(N^m),\ \mathrm{sk}(N^m),\ \mathrm{crs}(N^m),\ \mathrm{ZK}(\mathrm{crs}(M), x, w, N^m),\ \mathrm{garbageZK}(c, x, N^m),\ \mathrm{garbage}(N^m)\}$ for some $M \in \mathbf{N}_P$
  All of these cases follow immediately by definition of η and definition 4.

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 5 (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 $(x, w) \notin R^{\mathrm{sym}}_{\mathrm{adv}}$ occurs with negligible probability. We do this by a case distinction on $(m_x, m_w) \notin R^{\mathrm{comp}}_{\mathrm{adv}}$.

First, consider that it holds $(m_x, m_w) \notin R^{\mathrm{comp}}_{\mathrm{adv}}$. 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 $\mathcal{O}_{\mathrm{sim}}$ if $(x, w) \in R^{\mathrm{comp}}_{\mathrm{honest}}$. In this case, the protocol sends a proof z such that an extraction-failure happens and the extraction extracts a m_w such that $(m_x, m_w) \notin R^{\mathrm{comp}}_{\mathrm{adv}}$ (where $m_x = A_{\mathrm{getPub}}(z)$). Thus the adversary wins the honest simulation-sound extractability game which can only happen with negligible probability. Thus the case $(m_x, m_w) \notin R^{\mathrm{comp}}_{\mathrm{adv}}$ occurs with negligible probability. Therefore, in this case, it does not matter if $(x, w) \in R^{\mathrm{sym}}_{\mathrm{adv}}$ or not.

Now, consider the case that $(m_x, m_w) \in R^{\mathrm{comp}}_{\mathrm{adv}}$ holds. By Lemma 11, it follows that $(\tau(m_x), \tau^*(m_w)) \in R^{\mathrm{sym}}_{\mathrm{adv}}$ with overwhelming probability. Since $x = \tau(m_x)$ and $w = \tau^*(m_w)$, it follows that $(x, w) \notin R^{\mathrm{sym}}_{\mathrm{adv}}$ 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 3 (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 $(t_2, t_3) \in R^{\mathrm{sym}}_{\mathrm{honest}}$ 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 $(t_2, t_3) \in R^{\mathrm{sym}}_{\mathrm{honest}}$ holds by protocol condition 10. Thus in the case that the hybrid execution corresponds to a symbolic one, it follows that $(t_2, t_3) \in R^{\mathrm{sym}}_{\mathrm{honest}}$ 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 3. Finally, we show that the node traces of all simulators are indistinguishable.

Lemma 2 (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

$$(\mathrm{DY}, \text{H-Nodes}) \overset{C}{\approx} (\mathrm{DY}_1, \text{H-Nodes}_1) \qquad (1)$$

$$(\mathrm{DY}_1, \text{H-Nodes}_1) \overset{C}{\approx} (\mathrm{DY}_2, \text{H-Nodes}_2) \qquad (2)$$

$$(\mathrm{ZKBreak}_2, \mathrm{DY}_2, \text{H-Nodes}_2) \overset{C}{\approx} (\mathrm{ZKBreak}_3, \mathrm{DY}_3, \text{H-Nodes}_3) \qquad (3)$$

$$(\mathrm{ZKBreak}_3, \mathrm{DY}_3, \text{H-Nodes}_3) \overset{C}{\approx} (\mathrm{ZKBreak}_4, \mathrm{DY}_4, \text{H-Nodes}_4) \qquad (4)$$

$$(\mathrm{ZKBreak}_4, \mathrm{DY}_4, \text{H-Nodes}_4) \overset{C}{\approx} (\mathrm{ZKBreak}_5, \mathrm{DY}_5, \text{H-Nodes}_5) \qquad (5)$$

$$(\mathrm{ZKBreak}_5, \mathrm{DY}_5, \text{H-Nodes}_5) \overset{C}{\approx} (\mathrm{ZKBreak}_f, \mathrm{DY}_f, \text{H-Nodes}_f) \qquad (6)$$

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 $\mathcal{O}_{\mathrm{ZK}}^N$ 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 $\mathcal{O}_{\mathrm{ZK}}^N$. 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 $(x, w) \notin R_{\mathrm{honest}}^{\mathrm{comp}}$ the implementation, as well as the oracle, output $\perp$. So A_{ZK} and the $(\mathtt{prove}, x, w)$-query return $\perp$ in the same cases. In the case that $(x, w) \in R_{\mathrm{honest}}^{\mathrm{comp}}$ both compute a proof of x using witness w. Thus, it holds $(\mathrm{DY}, \text{H-Nodes}) \overset{C}{\approx} (\mathrm{DY}_1, \text{H-Nodes}_1)$.
- $(\mathrm{DY}_1, \text{H-Nodes}_1) \overset{C}{\approx} (\mathrm{DY}_2, \text{H-Nodes}_2)$
  In this step we replace $\mathcal{O}_{\mathrm{ZK}}$ by $\mathcal{O}_{\mathrm{Sim}}$ which returns a simulated proof for x if for the input (x, w) it holds that $(x, w) \in R_{\mathrm{honest}}^{\mathrm{comp}}$.
  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 $(x, w) \in R_{\mathrm{honest}}^{\mathrm{comp}}$). Thus $(\mathrm{DY}_1, \text{H-Nodes}_1)$ and $(\mathrm{DY}_2, \text{H-Nodes}_2)$ are indistinguishable, too.
- $(\mathrm{ZKBreak}_3, \mathrm{DY}_3, \text{H-Nodes}_3) \overset{C}{\approx} (\mathrm{ZKBreak}_4, \mathrm{DY}_4, \text{H-Nodes}_4)$
  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 $\beta(\mathrm{dk}(N)) := \perp$ 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.
- $(\mathrm{ZKBreak}_4, \mathrm{DY}_4, \text{H-Nodes}_4) \overset{C}{\approx} (\mathrm{ZKBreak}_5, \mathrm{DY}_5, \text{H-Nodes}_5)$
  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 $R_{\mathrm{adv}}^{\mathrm{sym}}$ is efficiently decidable.
  Thus replacing ENC by $\mathcal{O}_{\mathrm{fake}}$ leads to an indistinguishable execution and hence $(\mathrm{ZKBreak}_4, \mathrm{DY}_4, \text{H-Nodes}_4)$ and $(\mathrm{ZKBreak}_5, \mathrm{DY}_5, \text{H-Nodes}_5)$ 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 $\mathcal{O}_{\rm sig}$ without changing the distribution of $(\text{ZKBreak}, \text{DY}, \text{H-Nodes})$ (since we adopted τ accordingly, cf. the replacement of $A_{\rm crs}$ in Sim_1). In a run of the protocol β is never applied to $\text{sk}(N)$ (by protocol condition 5 and the use of an oracle for signing), so we can define $\beta(\text{sk}(N)) := \perp$ without changing the distribution of $(\text{ZKBreak}, \text{DY}, \text{H-Nodes})$.

- $(\text{ZKBreak}_2, \text{DY}_2, \text{H-Nodes}_2) \overset{C}{\approx} (\text{ZKBreak}_3, \text{DY}_3, \text{H-Nodes}_3)$

  We have already proven that $(\text{DY}_\text{f}) \overset{C}{\approx} (\text{DY}_3)$. Together with the fact that Sim_f is DY (Lemma 10), it follows that Sim_3 is DY. By Lemma 3, it follows that $(t_2, t_3) \in R^{\rm sym}_{\rm honest}$ for all ZK-nodes with arguments $t_1, \dots, t_4$ in a hybrid execution of Sim_3 (with overwhelming probability). Applying Lemma 11 leads to $(\beta(t_2), \beta(t_3)) \in R^{\rm comp}_{\rm honest}$ 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 $(\beta(t_2), \beta(t_3)) \in R^{\rm comp}_{\rm honest}$. Because this check would succeed with overwhelming probability in Sim_3, it actually succeeds in Sim_2.

  Thus the distribution of $(\text{ZKBreak}, \text{DY}, \text{H-Nodes})$ 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 7 (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 $\text{crs}(N) = \tau(A_{\rm crs}(z))$. Let $m_x := A_{\rm getPub}(z)$, $x := \tau(m_x)$, $m_w := \mathbf{E}(m_x, z, \text{extd}_N))$,[14] $w := \tau(m_w)$. Let S denote the set of $\mathbf{T}$ that the protocol already sent to the simulator in this execution.

We have $\mathbf{SymbExtr}(S, x) = \perp$ by definition of extraction failures. Thus one of the following cases occurs with non-negligible probability.

1) $(x, w) \notin R^{\rm sym}_{\rm adv}$
2) $S \not\vdash w$
3) $(x, w) \in R^{\rm sym}_{\rm adv}$ and $S \vdash w$ and $\mathbf{SymbExtr}(S, x) = \perp$

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: "$(x, w) \notin R^{\rm sym}_{\rm adv}$"
This would be a ZK-break. Thus, this case only occurs with negligible probability because of Lemma 5 and Lemma 2 (i).

Case 2: "$S \not\vdash w$"
By Lemma 10 and since $w = \tau^*(m_w)$, this case can only occur with negligible probability.

Case 3: "$(x, w) \in R^{\rm sym}_{\rm adv}$ and $S \vdash w$ and $\mathbf{SymbExtr}(S, x) = \perp$"
By definition, $\mathbf{SymbExtr}$ returns only $\perp$ if there is no w such that $(x, w) \in R^{\rm comp}_{\rm adv}$ and $S \vdash w$. So this case cannot occur.

■

The only thing, that is missing to apply Theorem 3 is to show that Sim is indistinguishable from an computational execution.

Lemma 12: 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 $\text{Nodes}^p_{\mathbf{M},A,\Pi_p,E}$ in the computational execution and the node trace $\text{H-Trace}_{\mathbf{M},\Pi_p,\text{Sim}}$ 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, $A_{\rm ek}(r_N)$, $A_{\rm dk}(r_N)$, $A_{\rm vk}(r_N)$, $A_{\rm sk}(r_N)$, $A_{\rm enc}(e, m, r_N)$, $A_{\rm sig}(s, m, r_N)$, $A_{\rm crs}(r_N)$, and $A_{\rm ZK}(c, x, w, r_N)$ are all pairwise distinct for all $N \in \mathcal{R}$ if they are well-formed.[15]

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 $A_{\rm ek}(r_N)$ and $A_{\rm ek}(r_M)$. For encryptions, by implementation condition 6, the probability that $A_{\rm enc}(e, m, r_N)$ for random r_N of type nonce matches any given string is negligible. Let e, e', m, m', r, r' be bitstrings. For $e \neq e'$, it holds that $A_{\rm enc}(e, m, r_N) \neq A_{\rm enc}(e', m', r'_N)$, because $A_{\rm ekof}$ returns in one case e and in the other e'. So if

[14]Here, extd_N is the extraction trapdoor that the simulator receives from the oracle $\mathcal{O}^N_{\rm ZK}$ by querying $(\texttt{extd})$.

[15]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.

$A_{\text{enc}}(e, m, r_N) = A_{\text{enc}}(e', m', r'_N)$, it holds $e = e'$. Additionally, $m = m'$ because decryption, using e as argument, deterministically computes m. So the only case that can occur is $A_{\text{enc}}(e, m, r_N) = A_{\text{enc}}(e, m, r_{N'}$. Since by protocol condition 3, each $A_{\text{enc}}(e, m, r_N)$ 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 7 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 (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^C_i and ν^C_i. Let s'_i denote the state of the adversary E in the computational model, and s^C_i 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 F.

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 $\text{ZK}(t_1, t_2, t_3, t_4)$ with $t_4 \notin \mathbf{N}$, t_1 not of the form $\text{crs}(N)$ with $N \in \mathbf{N}$, or $(\beta(t_2), \beta(t_3)) \notin R^{\text{comp}}_{\text{adv}}$ 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, $A_{\text{ek}}(r_N)$, $A_{\text{dk}}(r_N)$, $A_{\text{vk}}(r_N)$, $A_{\text{sk}}(r_N)$, as well as all occuring 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 $\bar{\nu}_1, \dots, \bar{\nu}_n$ 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 $\beta(\text{eval}_F(\underline{t})) = A_F(\beta(t_1), \dots, \beta(t_n))$. 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 F.

For given fixed randomness (see above), let ν'_i, f'_i be the nodes and functions as in the computational trace and ν^C_i, f^C_i 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^C_i the corresponding state of the adversary in the hybrid execution.

We will now show that $\text{Nodes}^p_{\mathbf{M},A,\Pi_p,E} = \text{H-Nodes}_{\mathbf{M},\Pi_p,\text{Sim}}(k)$.

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 $f'_{i+1}(\nu'_i) = A_F(f'_i(\bar{\nu}_1), \dots, f'_i(\bar{\nu}_n)) = A_F(\beta(f^C_i(\bar{\nu}_1)), \dots, \beta(f^C_i(\bar{\nu}_n)))$ for some nodes $\bar{\nu}_s$. And $f^C_{i+1}(\nu'_i) = f^C_{i+1}(\nu^C_i) = \text{eval}_F(f^C_i(\bar{\nu}_1), \dots, f^C_i(\bar{\nu}_n))$. 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 9, 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 1).

Proof of Theorem 1: By lemma 12 we get that Sim is indistinguishable for $\mathbf{M}, \Pi, A, E$ and for every polynomial p. By Lemma 10, Sim_f is DY which transfers to Sim by lemma 2 (iii). So Sim is a good simulator. By Theorem 3 we finally conclude that the implementation A is sound for every protocol as specified in the Theorem. ■

F. Proof of Claims

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 $\beta(\tau(m)) = \beta(N^m) = m$
- $\tau(c) = \mathrm{enc}(\mathrm{ek}(M), t, N)$ and c was output by $\beta(\mathrm{enc}(M), t, N))$
Then $\beta(\tau(c)) = \beta(\mathrm{enc}(\mathrm{ek}(M), t, N)) = A_{\mathrm{enc}}(\beta(\mathrm{ek}(M)), \beta(t), r_M)$. 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.
- $\tau(c) = \mathrm{enc}(\mathrm{ek}(M), t, N^c)$
Then $\beta(\tau(c)) = \beta(\mathrm{enc}(\mathrm{ek}(M), t, N^c) = c$.
- $\tau(c) = \mathrm{garbageEnc}(t, N^c)$
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 $\beta(\tau(c)) = \beta(\mathrm{dk}(N^c)) = c$.
- $\tau(c) = \mathrm{sig}(\mathrm{sk}(M), t, N)$ with $N, M \in \mathbf{N}_P$, earlier output by $\beta(\mathrm{sig}(\mathrm{sk}(M), t, N))$.
Then holds $\beta(\tau(c)) = \beta(\mathrm{sig}(\mathrm{sk}(M), t, N)) = c$ As in the case of encryption we have the same arguments and the a deterministic function, so the result has to be c again.
- $\tau(c) = \mathrm{sig}(\mathrm{sk}(M), t, N^c)$
Then holds $\beta(\tau(c)) = \beta(\mathrm{sig}(\mathrm{sk}(M), t, N^c)) = c$.
- $\tau(c) = \mathrm{garbageSig}(\mathrm{sk}(M), N^c)$
Then holds $\beta(\tau(c)) = \beta(\mathrm{garbageSig}(\mathrm{sk}(M), N^c)) = c$.
- $\tau(c) = \mathrm{vk}(N)$
Then by definition of τ, it holds $c = A_{\mathrm{vk}}(r_N) = \beta(\mathrm{vk}(N)) = \beta(\tau(c))$.
- $\tau(c) = \mathrm{vk}(N^c)$
Then holds $\beta(\tau(c)) = \beta(\mathrm{vk}(N^c)) = c$.
- $\tau(c) = \mathrm{sk}(N)$
Then by definition of τ, it holds $c = A_{\mathrm{sk}}(r_N) = \beta(\mathrm{sk}(N)) = \beta(\tau(c))$.
- $\tau(c) = \mathrm{sk}(N^c)$
Then holds $\beta(\tau(c)) = \beta(\mathrm{sk}(N^c)) = c$.
- $\tau(c) = \mathrm{ZK}(\mathrm{crs}(t_1), t_2, t_3, N)$ 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}.

- $\tau(c) = \text{ZK}(\text{crs}(t_1), t_2, t_3, N^c)$
  Then holds $\beta(\tau(c)) = \beta(\text{ZK}(\text{crs}(t_1), t_2, t_3, N^c)) = c$ by definition of β.
- $\tau(c) = \text{crs}(N)$ for $N \in \mathbf{N}_P$, earlier been output by $\beta(\text{crs}(N))$
  Then holds by determinism of A_{crs} that $\beta(\tau(c)) = \beta(\text{crs}(N)) = A_{\text{crs}}(r_N) = c$.
- $\tau(c) = \text{crs}(N^c)$
  Then holds $\beta(\tau(c)) = \beta(\text{crs}(N^c)) = c$.
- $\tau(c) = \text{garbageZK}(t_1, t_2, N^c)$
  Then holds $\beta(\tau(c)) = \beta(\text{garbageZK}(t_1, t_2, N^c)) = c$
- $\tau(c) = \text{pair}(t_1, t_2)$
  By construction of τ follows that $t_1 = \tau(A_{\text{fst}}(c))$ and $t_2 = \tau(A_{\text{snd}}(c))$. By induction hypothesis follows for $c_1 := A_{\text{fst}}(c)$ that $\beta(\tau(c_1)) = c_1$. The same holds for $c_2 := A_{\text{snd}}(c)$. Therefore we get $\beta(\text{pair}(t_1, t_2)) = A_{\text{pair}}(\beta(t_1), \beta(t_2)) = A_{\text{pair}}(\beta(\tau(A_{\text{fst}}(c))), \beta(\tau(A_{\text{snd}}(c)))) = A_{\text{pair}}(A_{\text{fst}}(c), A_{\text{snd}}(c)) = c$ where the last equality holds because of implementation conditions 11 and 1.
- $\tau(c) = \text{string}_0(t)$
  By definition of τ follows that $t = \tau(c')$ with $c' = A_{\text{unstring}_0}(c)$ and $c' \neq \bot$. The induction hypothesis implies that $\beta(\tau(c')) = c'$. So $\tau(\text{string}_0(t)) = A_{\text{string}_0}(\beta(t)) = A_{\text{string}_0}(c') = A_{\text{string}_0}(A_{\text{unstring}_0}(c)) = c$ The last equality holds because of implementation conditions 17 and 1.
- $\tau(c) = \text{string}_1(t)$
  Analogue to the case of $\tau(c) = \text{string}_0(t)$.
- $\tau(c) = \text{empty}$
  Then $c = A_{\text{empty}}() = \beta(\text{empty}) = \beta(\tau(c))$.
- $\tau(c) = \text{garbage}(N^c)$
  Then $\beta(\text{garbage}(N^c)) = c$.

■

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) "$F = \text{crs}$"
   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 $\beta(\text{eval}_{\text{crs}}(t_1)) = \beta(\text{crs}(t_1)) = A_{\text{crs}}(r_N) = A_{\text{crs}}(\beta(N))$.
2) "$F \in \{\text{ek}, \text{dk}, \text{vk}, \text{sk}\}$"
   Analogous to the case $F = \text{crs}$.
3) "$F = \text{ZK}$"
   A node annotated with ZK has as $t_1 = \text{crs}(N_1)$ 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:
   $A_{\text{ZK}}(\beta(t_1), \beta(t_2), \beta(t_3), \beta(t_4)) = A_{\text{ZK}}(A_{\text{crs}}(r_{N_1}), \beta(t_2), \beta(t_3), r_{N_2}) = \beta(\text{eval}_{\text{ZK}}(t_1, t_2, t_3, t_4))$.
4) "$F = \text{getPub}$"
   If the argument t is neither of the form $\text{ZK}(t_1, t_2, t_3, t_4)$ nor $\text{garbageZK}(t_1, t_2, t_3)$ then $\beta(\text{getPub}(t)) = \bot$ and $A_{\text{getPub}}(\beta(t)) = \bot$, too. So first consider $t = \text{ZK}(t_1, t_2, t_3, t_4)$. Then, by protocol conditions 3 and 8 and by definition of τ, it follows that t_1 has the form $\text{crs}(u_1)$ with $u_1 \in \mathbf{N}$ and $t_4 \in \mathbf{N}$.
   Thus we have $\beta(\text{eval}_{\text{getPub}}(\text{ZK}(\text{crs}(u_1), t_2, t_3, t_4))) = \beta(t_2)$.
   Case 1: $t_4 = N \in \mathbf{N}_P$
   Then it holds that $A_{\text{getPub}}(\beta(\text{ZK}(\text{crs}(u_1), t_2, t_3, N))) = A_{\text{getPub}}(A_{\text{ZK}}(A_{\text{crs}}(r_{t_1}), \beta(t_2), \beta(t_3), r_N)) = \beta(t_2)$ where the last equality holds because of implementation condition 27.
   Case 2: $t_4 = N^m \in \mathbf{N}_E$
   Then we have $A_{\text{getPub}}(\beta(\text{ZK}(\text{crs}(t_1), t_2, t_3, N^m))) = A_{\text{getPub}}(m)$ where $\tau(m) = \text{ZK}(\text{crs}(t_1), t_2, t_3, N^m)$ and $\tau(A_{\text{getPub}}(m)) = t_2$ by definition of τ. By applying β on both sides, it follows that $\beta(t_2) = \beta(\tau(A_{\text{getPub}}(m))) = A_{\text{getPub}}(m)$ where the last equality holds because of Claim 1.
   Now, consider the case that $t = \text{garbageZK}(t_1, t_2, t_3)$. 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 $t_2 = \tau(A_{\text{getPub}}(z))$. Therefore, it holds that $A_{\text{getPub}}(\beta(\text{garbageZK}(t_1, t_2, t_3))) = A_{\text{getPub}}(z) \overset{(*)}{=} \beta(\tau(A_{\text{getPub}}(z))) = \beta(t_2)$. Here, the equality $(*)$ holds because of Claim 1.
5) "$F = \text{verify}_{\text{ZK}}$"
   If $\beta(t_2)$ has not the type zero-knowledge proof, then the left hand side is $\bot$ by definition of β, and the right hand side is $\bot$ by implementation condition 22.
   Therefore consider t_2 to be of the form $\text{ZK}(u_1, u_2, u_3, u_4)$ or $\text{garbageZK}(u_1, u_2, u_3)$. Additionally has to hold that $t_1 = u_1$ and that by protocol condition 9 t_1 is of the form $\text{crs}(N_1)$ for some $N_1 \in \mathbf{N}_P$. Consider the following subcases:
   a) $t_2 = \text{ZK}(\text{crs}(N_1), u_2, u_3, u_4)$ 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 Lemma 3 and the fact that Sim is DY (Lemma 10 and 2 (i)), it holds that the proof is, valid, more precisely $(u_2, u_3) \in R^{\text{sym}}_{\text{honest}}$.
      Therefore, it follows $\text{eval}_{\text{verify}_{\text{ZK}}}(t_1, t_2) = t_2$. Thus, it holds $\beta(t_2) = A_{\text{ZK}}(A_{\text{crs}}(r_{N_1}), \beta(u_2), \beta(u_3), r_{N_2})$.
      By Lemma 11 follows $(\beta(u_2), \beta(u_3)) \in R^{\text{comp}}_{\text{honest}}$, therefore – by completeness of the zero-knowledge proof system – this gives a correct proof. Thus verification succeeds, and therefore by implementation condition 21 $A_{\text{verify}_{\text{ZK}}}(\beta(t_1), \beta(t_2)) = \beta(t_2)$.
   b) $t_2 = \text{ZK}(\text{crs}(N_1), u_2, u_3, u_4)$ 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 $\bot = A_{\mathrm{verify}_{\mathrm{ZK}}}(A_{\mathrm{crsof}}(z), z) = A_{\mathrm{verify}_{\mathrm{ZK}}}(\beta(t_1), \beta(t_2))$ (by implementation condition 21). By definition of $\mathrm{verify}_{\mathrm{ZK}}$ follows that $\mathrm{eval}_{\mathrm{verify}_{\mathrm{ZK}}}(t_1, t_2) = \bot$ and therefore $\beta(\mathrm{eval}_{\mathrm{verify}_{\mathrm{ZK}}}(t_1, t_2)) = \bot$, 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 $A_{\mathrm{iszk}}(\beta(t_1)) = \bot$ by implementation condition 18. On the other hand holds $\beta(\mathrm{eval}_{\mathrm{iszk}}(t_1)) = \beta(\bot) = \bot$.
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) "$F \in \{\mathrm{isenc}, \mathrm{issig}, \mathrm{isek}, \mathrm{isvk}, \mathrm{iscrs}\}$"
Analogue to the case $F = \mathrm{iszk}$.

8) "$F = \mathrm{crsof}$"
If t_1 is not of the form $\mathrm{ZK}(u_1, u_2, u_3, N)$ or $\mathrm{garbageZK}(u_1, u_2, N)$ with $N \in \mathbf{N}$. Then $\mathrm{eval}_{\mathrm{crsof}}(t_1) = \bot$ and $\beta(t_1)$ is not of type zero-knowledge proof, therefore by implementation condition 24 holds $A_{\mathrm{crsof}}(\beta(t_1)) = \bot$.
In the other both cases holds $\beta(\mathrm{eval}_{\mathrm{crsof}}(t_1)) = \beta(u_1)$. Consider the following subcases:

a) $t_1 = \mathrm{ZK}(u_1, u_2, u_3, N)$ with $N \in \mathbf{N}_P$.
So the term was generated by the protocol, therefore - by protocol condition 8 - holds that $u_1 = \mathrm{crs}(M)$ for some $M \in \mathbf{N}_P$. Thus holds $A_{\mathrm{crsof}}(\beta(t_1)) = A_{\mathrm{crsof}}(A_{\mathrm{ZK}}(A_{\mathrm{crs}}(r_M), \beta(u_2), \beta(u_3), r_N)) \overset{*}{=} A_{\mathrm{crs}}(r_M) = \beta(\mathrm{crs}(M)) = \beta(\mathrm{eval}_{\mathrm{crsof}}(t_1))$ where (*) holds by implementation condition 23.

b) $t_1 = \mathrm{garbageZK}(u_1, u_2, N)$.
By protocol condition 2 follows that t_1 was constructed by τ, i.e. $t_1 = \mathrm{garbageZK}(u_1, u_2, N^z)$ for some $z \in \{0,1\}^*$ of type zero-knowledge and $u_1 = \tau(A_{\mathrm{crsof}}(z))$. Thus we have: $\beta(u_1) = \beta(\tau(A_{\mathrm{crsof}}(z))) \overset{*}{=} A_{\mathrm{crsof}}(z) = A_{\mathrm{crsof}}(\beta(t_1))$ where the last equality holds by definition of β and (*) holds by claim 1.

c) $t_1 = \mathrm{ZK}(u_1, u_2, u_3, N^z)$ with $N^z \in \mathbf{N}_E$.
This case is analogue to the case $t_1 = \mathrm{garbageZK}$.

9) "$F \in \{\mathrm{ekof}, \mathrm{vkof}\}$"
Analogue to the case $F = \mathrm{crsof}$.

10) "$F = \mathrm{enc}$"
By protocol condition 3 holds that $t_3 = N$ for $N \in \mathbf{N}_P$. If t_1 is of the form $\mathrm{ek}(u)$, then $\beta(\mathrm{enc}(t_1, t_2, t_3)) = A_{\mathrm{enc}}(\beta(t_1), \beta(t_2), r_N) = A_{\mathrm{enc}}(\beta(t_1), \beta(t_2), \beta(t_3))$, because $\beta(N) = r_N$.
So let t_1 be not of the form $\mathrm{ek}(u)$. Thus $\beta(\mathrm{enc}(t_1, t_2, t_3)) = \bot$ and $A_{\mathrm{enc}}(\beta(t_1), \beta(t_2), \beta(t_3)) = \bot$, because $\beta(t_1)$ is not of type encryption key and implementation condition 19.

11) "$F = \mathrm{dec}$"
By protocol condition 6, $t_1 = \mathrm{dk}(N)$ with $N \in \mathbf{N}_P$. We distinguish the following cases for t_2:

a) $t_2 = \mathrm{enc}(\mathrm{ek}(N), u_2, M)$ with $M \in \mathbf{N}_P$
Then $A_{\mathrm{dec}}(\beta(t_1), \beta(t_2)) = A_{\mathrm{dec}}(A_{\mathrm{dk}}(r_N), A_{\mathrm{enc}}(A_{\mathrm{ek}}(N), \beta(u_2), r_M)) = \beta(u_2)$ by implementation condition 12. Furthermore $\beta(\mathrm{dec}(t_1, t_2)) = \beta(u_2)$ by definition of dec.

b) $t_2 = \mathrm{enc}(\mathrm{ek}(N), u_2, N^c)$
Then t_2 was produced by τ and hence c is of type ciphertext and $\tau(A_{\mathrm{dec}}(A_{\mathrm{dk}}(r_N), c)) = u_2$. Then by Claim 1, $A_{\mathrm{dec}}(A_{\mathrm{dk}}(r_N), c) = \beta(u_2)$ and hence $A_{\mathrm{dec}}(\beta(t_1), \beta(t_2)) = A_{\mathrm{dec}}(A_{\mathrm{dk}}(r_N), c) = \beta(u_2) = \beta(\mathrm{dec}(t_1, t_2))$.

c) $t_2 = \mathrm{enc}(u_1, u_2, u_3)$ with $u_1 \neq \mathrm{ek}(N)$
As shown above (case F = ekof), $A_{\mathrm{ekof}}(\beta(\mathrm{enc}(u_1, u_2, u_3)) = \beta(\mathrm{ekof}(\mathrm{enc}(u_1, u_2, u_3)) = \beta(u_1)$. Moreover, from Claim 3, $A_{\mathrm{ekof}}(\beta(\mathrm{enc}(u_1, u_2, u_3)) = \beta(u_1) \neq \beta(\mathrm{ek}(N)) = A_{\mathrm{ek}}(r_N)$. Thus by implementation condition 4, $A_{\mathrm{dec}}(\beta(t_1), \beta(t_2)) = A_{\mathrm{dec}}(A_{\mathrm{dk}}(r_N), \beta(\mathrm{enc}(u_1, u_2, u_3))) = \bot$. Furthermore, $\mathrm{dec}(t_1, t_2) = \bot$ and thus $\beta(\mathrm{dec}(t_1, t_2)) = \bot$.

d) $t_2 = \mathrm{garbageEnc}(u_1, N^c)$
Assume that $m := A_{\mathrm{dec}}(\beta(t_1), \beta(t_2)) = A_{\mathrm{dec}}(A_{\mathrm{dk}}(r_N), c) \neq \bot$. 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 $\tau(c) = \mathrm{enc}(\mathrm{ek}(N), \tau(m), N^c) \neq t_2$. This is a contradiction to $t_2 = \tau(c)$, so the assumption that $A_{\mathrm{dec}}(\beta(t_1), \beta(t_2)) \neq \bot$ was false. So $A_{\mathrm{dec}}(\beta(t_1), \beta(t_2)) = \bot = \beta(\bot) = \beta(\mathrm{dec}(t_1, \mathrm{garbageEnc}(u_1, N^c)))$.

e) All other cases
Then $\beta(t_2)$ is not of type ciphertext. By implementation condition 13, $A_{\text{ekof}}(\beta(t_2)) = \bot$. Hence $A_{\text{ekof}}(\beta(t_2)) \neq A_{\text{ek}}(r_N)$ and by implementation condition 4, $A_{\text{dec}}(\beta(t_1), \beta(t_2)) = A_{\text{dec}}(A_{\text{dk}}(r_N), \beta(t_2)) = \bot = \beta(\text{dec}(t_1, t_2))$.

12) "$F = \text{sig}$"
By protocol conditions 3 and 7 we have that $t_1 = \text{sk}(N)$ and $t_3 = M$ for $N, M \in \mathbf{N}_P$. Then $\beta(\text{sig}(t_1, t_2, t_3)) = A_{\text{sig}}(A_{\text{sk}}(r_N), \beta(t_2), r_M) = A_{\text{sig}}(\beta(\text{sk}(N)), \beta(t_2), \beta(M)) = A_{\text{sig}}(\beta(t_1), \beta(t_2), \beta(t_3))$.

13) "$F = \text{verify}_{\text{sig}}$"
We distinguish the following subcases:

a) "$t_1 = \text{vk}(N)$ and $t_2 = \text{sig}(\text{sk}(N), u_2, M)$ with $N, M \in \mathbf{N}_P$"
Then $A_{\text{verify}_{\text{sig}}}(\beta(t_1), \beta(t_2)) = A_{\text{verify}_{\text{sig}}}(A_{\text{vk}}(r_N), A_{\text{sig}}(A_{\text{sk}}(r_N), \beta(u_2), r_M)) \overset{*}{=} \beta(u_2) = \beta(\text{verify}_{\text{sig}}(\underline{t}))$ where $(*)$ uses implementation condition 15.

b) "$t_2 = \text{sig}(\text{sk}(N), u_2, M)$ and $t_1 \neq \text{vk}(N)$ with $N, M \in \mathbf{N}_P$"
By Claim 3, $\beta(t_1) \neq \beta(\text{vk}(N))$ Furthermore $A_{\text{verify}_{\text{sig}}}(\beta(\text{vk}(N)), \beta(t_2)) = A_{\text{verify}_{\text{sig}}}(\beta(t_1), A_{\text{sig}}(A_{\text{sk}}(r_N), \beta(u_2), r_M)) \overset{*}{=} \beta(u_2) \neq \bot$. Hence with implementation condition 16, $A_{\text{verify}_{\text{sig}}}(\beta(t_1), \beta(t_2)) = \bot = \beta(\bot) = \text{verify}_{\text{sig}}(t_1, t_2)$.

c) "$t_1 = \text{vk}(N)$ and $t_2 = \text{sig}(\text{sk}(N), u_2, M^s)$"
Then t_2 was produced by τ and hence s is of type signature with $\tau(A_{\text{vkof}}(s)) = \text{vk}(N)$ 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 $\beta(t_1) = \beta(\text{vk}(N)) = \beta(\tau(A_{\text{vkof}}(s))) = A_{\text{vkof}}(s)$. 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 $\beta(\text{verify}_{\text{sig}}(t_1, t_2)) = \beta(\text{verify}_{\text{sig}}(\text{vk}(N), \text{sig}(\text{sk}(N), u_2, M^s))) = \beta(u_2)$.

d) "$t_2 = \text{sig}(\text{sk}(N), u_2, M^s)$ and $t_1 \neq \text{vk}(N)$"
As in the previous case, $A_{\text{verify}_{\text{sig}}}(A_{\text{vkof}}(s), s) \neq \bot$ and $\beta(\text{vk}(N)) = A_{\text{vkof}}(s)$. Since $t_1 \neq \text{vk}(N)$, by Claim 3, $\beta(t_1) \neq \beta(\text{vk}(N)) = A_{\text{vkof}}(s)$. 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 $\text{vk}(\dots)$. The latter case only occurs if $A_{\text{vkof}}(s) = \bot$ as otherwise $A_{\text{vkof}}(s)$ is of type verification key and hence $\tau(A_{\text{vkof}}(s)) = \text{vk}(\dots)$. 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) "$F \in \{\text{pair, fst, snd, string}_0\text{, unstring}_0\text{, string}_1\text{, unstring}_1\text{, empty}\}$" 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) "$F \in \{\text{garbage, garbageEnc, garbageSig, garbageZK}\} \cup \mathbf{N}_E$"
By protocol condition 2, the constructor F does not occur in the protocol.

■

G. *Generic Construction of WSSZK-Proof Systems*

The construction in [39] 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 4: Let π be a length regular, extractable non-interactive zero-knowledge proof system and assume that one way functions exist. Then the construction in [39] leads to a weakly symbolically-sound zero-knowledge proof system Π.

In paper [39], they proved that the construction satisfies all properties listed in definition 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 [39] uses a strong one-time signature scheme $(\mathbf{K}, \text{sig}, \text{verify}_{\text{sig}})$ 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 $\{1, \dots, q'(k)\}$. 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 $m^1, \dots, m^{t(k)}$ 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 [39].

We show that the construction given in [39] 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 $\Sigma = \sigma_0 \circ \sigma_1 \circ \dots \circ \sigma_{q'(k)}$. 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 $P := p_1 \circ \dots \circ p_{q'(k)}$.
(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 $P := p_1 \circ \dots \circ p_{g'(k)}$.
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 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 [39].
2) Simulation-Extractability:
Let $S_1, \dots, S_n$ 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 $j \in \{1, \dots, q'(k)\}$ 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.

H. Example relations

In this section we prove that the relations shown in section III satisfy definition 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:

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

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 13: If, in addition to the implementation conditions, it holds:

1) For all $k, k', m, m', r \in \{0,1\}^*$ $A_{\text{enc}}(k,m,r) = A_{\text{enc}}(k,m',r)$ implies $m = m'$ and $A_{\text{enc}}(k,m,r) = A_{\text{enc}}(k',m,r)$ 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 $R^{\text{comp}}_{\text{honest}}, R^{\text{comp}}_{\text{adv}}$ implement $R^{\text{sym}}_{\text{adv}}$ with usage restriction $R^{\text{sym}}_{\text{honest}}$.

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

We first show that if $(x,w) \in R^{\text{sym}}_{\text{honest}}$ and $\text{img}_\eta(x) \neq \bot \neq \text{img}_\eta(w)$, then $(\text{img}_\eta(x), \text{img}_\eta(w)) \in R^{\text{comp}}_{\text{honest}}$. Thus, fix $x = (\text{enc}(k,m,r),k,m)$ and $w = r \in \mathbf{N}_P$. Then $\text{img}_\eta(\text{enc}(k,m,r)) = A_{\text{enc}}(\text{img}_\eta(k), \text{img}_\eta(m), \text{img}_\eta(r)) =: A_{\text{enc}}(k',m',r')$ Here it is crucial that $\text{img}_\eta(x) \neq \bot$ and hence $A_{\text{enc}}(\text{img}_\eta(k), \text{img}_\eta(m), \text{img}_\eta(r)) \neq \bot$. So, it follows $(\text{img}_\eta(x), \text{img}_\eta(w)) = ((A_{\text{enc}}(k',m',r'),m',r'),r') \in R^{\text{sym}}_{\text{honest}}$.

Now we show that if $(\text{img}_\eta(x), \text{img}_\eta(w)) \in R^{\text{comp}}_{\text{adv}}$, then $(x,w) \in R^{\text{sym}}_{\text{adv}}$. Fix some $x, w \in \mathbf{T}$ with $(\text{img}_\eta(x), \text{img}_\eta(w)) \in R^{\text{comp}}_{\text{adv}}$. Then $\text{img}_\eta(x) = (c',k',m') := (\text{img}_\eta(c), \text{img}_\eta(k), \text{img}_\eta(m))$ with $c' = A_{\text{enc}}(k',m',r')$ for some $c,k,m \in \mathbf{T}, r' \in \{0,1\}^*$. Since $\text{img}_\eta(x) = (c',k',m')$, we have $x = (c,k,m)$ by definition of img_η and injectivity of A_{pair}. By implementation condition 2, $c' = \text{img}_\eta(c)$ has type ciphertext. Thus, by definition of consistent environments, $c = \text{enc}(k_1,m_1,r_1)$ 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 $c' = \text{img}_\eta(c) = A_{\text{enc}}(\text{img}_\eta(k_1), \text{img}_\eta(m_1), \text{img}(r_1))$. Since A_{enc} is injective in its first two arguments (follows by the assumption (1) of the lemma), we have that $\text{img}_\eta(k_1) = \text{img}_\eta(k)$, $\text{img}_\eta(m_1) = \text{img}_\eta(m)$. Thus $(x,w) = ((\text{enc}(k,m,r_1),k,m),w) \in R^{\text{sym}}_{\text{adv}}$.

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

■

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.

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

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

Lemma 14: If, in addition to the implementation conditions, it holds: For all $d, d', c \in \{0,1\}^*$ it holds: If $A_{\text{dec}}(d,c) \neq \bot \neq A_{\text{dec}}(d',c)$ then $d' = d$. Then follows that $R^{\text{comp}}_{\text{honest}}, R^{\text{comp}}_{\text{adv}}$ implement $R^{\text{sym}}_{\text{adv}}$ with usage restriction $R^{\text{sym}}_{\text{honest}}$.

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 $A_{\text{dec}}(d,m) = \bot$.

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

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

Now we show that if $(\text{img}_\eta(x), \text{img}_\eta(w)) \in R^{\text{comp}}_{\text{adv}}$, then $(x,w) \in R^{\text{sym}}_{\text{adv}}$. By definition $\text{img}_\eta(x) = (m',m'_1)$ and $\text{img}_\eta(w) = d'$ such that $A_{\text{dec}}(d',m'_1) \neq \bot$. 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 $A_{\text{dec}}(d',m'_1) \neq \bot$. Hence, by consistency of η it follows that m_1 has the form $\text{enc}(\text{ek}(N),t,M)$ 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 $(x, w) \in R^{\mathrm{sym}}_{\mathrm{adv}}$.

■