# Symbolic Malleable Zero-knowledge Proofs

**Abstract**

Zero-knowledge (ZK) proofs have become a central building block for a variety of modern security protocols, e.g., as ZK-SNARKs in Pinocchio (IEEE S&P 2013) and ADSNARK (IEEE S&P 2015). One of the reasons is that modern ZK constructions, such as the Groth-Sahai proof system, offer novel types of cryptographic flexibility: a participant is able to re-randomize existing ZK proofs to achieve, for instance, message unlinkability in anonymity protocols; she can hide public parts of a ZK proof statement to meet her specific privacy requirements; and she can logically compose ZK proofs in order to construct new proof statements. ZK proof systems that permit these transformations are called *malleable*. However, since these transformations are accessible also to the adversary, analyzing the security of these protocols requires one to cope with a much more comprehensive attacker model – a challenge that automated protocol analysis thus far has not been capable of dealing with.

In this work, we introduce the first symbolic abstraction of malleable ZK proofs. We further prove the computational soundness of our abstraction with respect to observational equivalence, which enables the computationally sound verification of privacy properties. Finally, we show that our symbolic abstraction is suitable for ProVerif, a state-of-the-art cryptographic protocol verifier, by verifying an improved version of the anonymous webs of trust protocol.

## 1 Introduction

Proving security and privacy properties of protocols that rely on cryptographic operations constitutes a highly complex and error-prone task. As a consequence, research has strived for the automation of such proofs soon after the first protocols were developed [DY83, EG83, Mer83]. To tame the inherent complexity of such proofs, the cryptographic operations were abstracted as symbolic terms that obey simple cancelation rules, so-called Dolev-Yao models [DY83].

While Dolev-Yao models in the beginning included only basic cryptographic primitives such as encryption and digital signatures, the Dolev-Yao models have been extended over the years to on the one hand more basic primitives (e.g., DH exponentiation, bilinear pairings, and AC-operators [SSCB14]), requiring more sophisticated verification techniques, and on the other hand more complex primitives (e.g., secure multi-party computation [BMM10]). In this work, we focus on zero-knowledge proofs [GMR89], which have rapidly become a central building block for a variety of modern privacy protocols, such as verifiable computation [PHGR13, BBFR15], e-voting systems [CCM08], anonymous credentials [BL13], group signatures, and many others. A zero-knowledge proof 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 other than the sole fact that $x$ constitutes a valid statement.

In addition to these core properties, commonly used ZK proof schemes, such as the Groth-Sahai proof system [GS08] and partly the ZK-SNARKs used in Pinocchio [PHGR13] or ADSNARK [BBFR15], offer additional cryptographic functionalities. First, a participant is able to re-randomize existing ZK proofs, which is fundamental for achieving unlinkability in anonymity protocols. Second, in order to adhere to individual privacy requirements, a participant can hide public parts of a ZK proof statement to selectively hide information of third-party proofs (e.g., this enables the design of privacy-preserving credentials for open-ended systems [MP11, BMP12, MPR13]). Third, a participant can logically compose ZK proofs in order to construct new proof statements.

Existing symbolic abstractions, however, are restricted to non-malleable ZK proofs, which are modelled as monolithic building blocks that cannot be further transformed [BMU08, BHM08, BHM12]. There have even been two corresponding computational soundness results for these abstractions [BBU13, BU08]. However,

these computational soundness results only hold for trace properties, neglecting the privacy properties that zero-knowledge proofs offer, and these abstractions do not model malleable proofs, that can either be re-randomized or composed and existentially quantified without the knowledge of secret values.

Designing a symbolic model for malleable ZK proofs is hard given the number and complexity of the cryptographic functionalities to be considered, even more so automating security proofs in such a setting. Furthermore, the symbolic models of non-malleable ZK proofs have been justified by computational soundness results, i.e., a successful symbolic analysis carries over to the corresponding cryptographic ZK realizations [BU08, BBU13]. These results, however, are limited to trace properties, and therefore do not apply to privacy properties, which are the primary reason for deploying zero-knowledge proofs.

### 1.1 Our Contribution

First, we provide a symbolic abstraction of malleable ZK (MZK) proofs by means of an equational theory.[1] The main conceptual challenge we faced when devising this abstraction was to identify a representation that is amenable to automated verification and yet captures an expressive and computationally sound class of transformations. In particular, we categorize transformations as one of the three types: (*i*) *re-randomization* (used, e.g., to make forwarded proofs unlinkable), (*ii*) *logical transformations* (used, e.g., to produce a proof of the statement $x \wedge y$ from independent proofs of $x$ and $y$), and (*iii*) *existential quantification* (used to selectively hide information from existing proofs). To render automated verification feasible in the presence of these rich transformations, we tailor our abstraction to conjunctive statements. As a case study, we use the protocol verifier ProVerif to verify an improved version of anonymous webs of trust using ProVerif.

Second, we prove the computational soundness of the abstractions with respect to the kind of equivalence properties that can be verified by the ProVerif cryptographic protocol verifier. These results are given in CoSP [BHU09, BMR14a], a modular and generic framework for symbolic protocol analysis and computational soundness proofs. The process of embedding calculi is decoupled from computational soundness proofs of cryptographic primitives: as a result, our work immediately entails a computationally sound symbolic model in the applied-pi calculus, and we show that our result also entails a computationally sound symbolic abstraction in ML (building on results from [BMU10]).

**Outline of the Paper.** We first present and discuss our symbolic abstraction of malleable ZK proofs in Appendix A. Then, we review the CoSP framework in § 3. Finally, we show computational soundness in § 4. § 6 concludes this work and outlines future work.

Proofs and technical details are reported in the full version [ful].

## 2 Symbolic abstraction of malleable ZK proofs

### 2.1 The applied pi-calculus

We concisely review the syntax and the operational semantics of the applied pi-calculus [BAF08]. The set of terms is generated by a countably infinite set of names (denoted as $a$, $t$, $m$, and $n$), variables $x$, and constructors $f$ that can be applied to terms. We denote vectors by underlining, e.g., $\underline{x}$ for $x = (x_1, \ldots, x_k)$ for some $k$. Destructors, ranged over by $D$, are partial functions that processes can apply to terms.

*Plain processes*, ranged over by $P$, are defined as follows. The null process 0 does nothing; *new* $n.P$ generates a fresh name $n$ and then executes $P$; $a(x).P$ receives a message $m$ from channel $a$ and then executes $P\{m/x\}$; $\overline{a}\langle m\rangle.P$ outputs message $m$ on channel $a$ and then executes $P$; $P \mid Q$ runs $P$ and $Q$ in parallel; $!P$ behaves as an unbounded number of copies of $P$ in parallel; *let* $x = D(\underline{t})$ *in* $P$ *else* $Q$ applies the destructor $D$ to the terms $\underline{t}$ and, if the application succeeds and produces the term $t'$ ($D(\underline{t}) = t'$) then the process behaves as $P\{t/x\}$, otherwise (i.e., if $D(\underline{t}) = \bot$) the process behaves as $Q$. The scope of names and variables is bound by restrictions, inputs, and lets. A process is closed if it does not have free variables. A *context* $C[\bullet]$ is a process with a hole $\bullet$ in the place of a subprocess, while an *evaluation context* is a context where the hole is not under a replication, a conditional, an input, or an output.

The operational semantics of the applied pi-calculus is defined in terms of *structural equivalence* ($\equiv$) and *internal reduction* ($\rightarrow$); the former allows for a syntactic rearrangement of processes, the latter rules process synchronizations and let evaluations.

[1] We further consider asymmetric encryptions and digital signatures, handled in a standard way [BHU09].

Intuitively, two processes are said to be observationally equivalent if no context (i.e., adversary) can distinguish them. Observational equivalence is the symbolic counterpart of the indistiguishability property in the cryptographic setting, and it is typically used to formalise various privacy properties. Automated tools for the verification of observational equivalences typically focus on bi-processes: a bi-process is a pair of processes that only differ in the terms they operate on. Formally, they contain expressions of the form $choice[a, b]$, where $a$ is used in the left process and $b$ is used in the right one. The semantics of bi-processes is defined such that they can only reduce if both its processes can reduce in the same way.

**Definition 1** (Uniform bi-process)**.** *A bi-process $Q$ is* uniform *if $left(Q) \to R_{left}$ implies that $Q \to R$ for some bi-process $R$ with $left(R) \equiv R_{left}$, and symmetrically for $right(Q) \to R_{right}$ with $right(R) \equiv R_{right}$.*

Verifying uniformity suffices to enforce observational equivalence.

**Theorem 1.** *Let $Q$ be a closed biprocess. If, for all plain evaluation contexts $C[\bullet]$ and reductions $C[Q] \to P$, the biprocess $P$ is uniform, then $left(Q)$ is observationally equivalent to $right(Q)$.*

For a complete description of the applied pi-calculus, we refer to [BAF08].

## 2.2 Anonymous Webs of Trust

Before describing our symbolic model of malleable zero-knowledge proofs, we introduce the running example utilised in this paper.

In cryptography, one of the most daunting tasks is to ensure that a public key belongs to a certain person. The two most prevailing methods are public-key infrastructures (PKIs) and webs of trust. In both cases, the overall idea is the same: a trusted third party vouches for the connection between a pair of public keys, i.e., encryption key and verification key, and its owner by using a digital signature. The main difference between these two approaches is that the PKI defines several third parties as trust anchors that manage the trust whereas in webs of trust every participant can choose whom to trust.

For instance, if Charlie has been issued a signature sig on his verification key $\mathrm{vk}_B$ by Bob,[2] represented by his verification key $\mathrm{vk}_B$, everybody that trusts Bob can accept that $\mathrm{vk}_C$ belongs to Charlie. In this way, it is possible to build trust chains: if Alice trusts Bob and Bob trusts Charlie, Alice can derive some degree of trust into messages signed by Charlie.

**Anonymous webs of trust.** While the web of trust approach solves the problem of central trusted entities, it lacks important privacy properties: when sending a signed message, Charlie naturally reveals his identity and, in the process authenticates the message. Alice can check that the received message is authentic by checking the signature $\mathsf{verify}(m, \mathsf{sign}(m, \mathrm{sk}_C), \mathrm{vk}_C) = \text{true}$ to ensure that there is a trust chain from her to Charlie.

Technically, $\mathrm{sk}_C = \mathrm{sk}(s_C)$ for some value $s_C$ only to known to Charlie. Analogously, $\mathrm{vk}_C = \mathrm{vk}(s_C)$, i.e., the two keys are connected via $s_C$. The verification key is derived by using the destructor $\mathsf{vkofsk}_{/1}(sk(s)) = vk(s)$. If by the context, it is clear which verification key belongs to which signing key, we neglect to write them as constructors.

The design of webs of trust inherently reveals the creator of a message. Backes, Lorenz, Maffei, and Pecina [BLMP10] solved this problem by introducing the concept of anonymous webs of trust. Intuitively, they use zero-knowledge proofs to show that there exists a trust chain from the recipient of the message to the sender without disclosing the signatures or the verification keys of the principals along the chain, i.e., the sender remains anonymous.[3] The message is authenticated by the proof that shows that there is a trust chain from Alice to the sender of $m$ as depicted in Figure 1. More precisely, the proof shows that $\exists \mathrm{sig}_1, \mathrm{vk}_1, \mathrm{sig}_2, \mathrm{vk}_2, \mathrm{sig}_3.\ \mathsf{verify}(\mathrm{vk}_A, \mathrm{sig}_1, \mathrm{vk}_1) \wedge \mathsf{verify}(\mathrm{vk}_1, \mathrm{sig}_2, \mathrm{vk}_2) \wedge \mathsf{verify}(\mathrm{vk}_2, \mathrm{sig}_3, m)$ (we silently assume that every verification returned true).

[2] For the sake of simplicity, we restrict signatures in webs of trust to verification keys. Actual webs of trust also consider encryption keys and attributes.

[3] The anonymity guarantee is not unconditional since the structure of the web of trust may reveal information about the sender. We refer the interested reader to the paper for a detailed discussion.

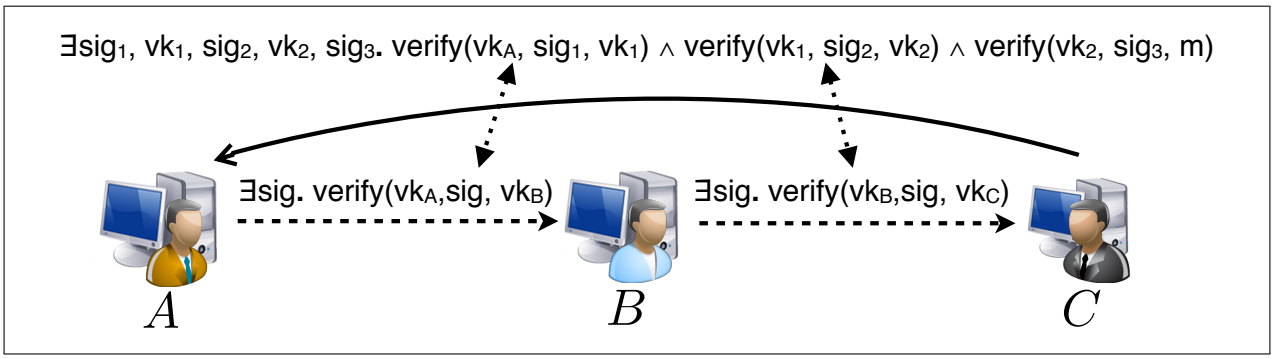


Figure 1: In decentralized anonymous webs of trust, zero-knowledge proofs are exchanged between Alice and Bob, and Bob and Charlie. Since these proofs are reused when proving a trust chain, denoted by the dotted arrows, the proofs have to be re-randomized. Otherwise, Bob, respectively Alice, will be able to identify the proof send to Charlie, respectively Bob, with the corresponding part in the proof send from Charlie to Alice.

$$
\begin{aligned}
T ::= \;& \mathrm{enc}(\mathrm{ek}(N), T, N) \mid \mathrm{ek}(N) \mid \mathrm{dk}(N) \mid \mathrm{pair}(T,T) \mid \\
& \mathrm{sig}(\mathrm{sk}(N), T, N) \mid \mathrm{vk}(N) \mid \mathrm{sk}(N) \mid S \mid N \\
& \mathsf{zkp}(R) \mid \mathsf{com}(T, (M, N)) \\
Q ::= \;& \varepsilon \mid \mathrm{string}_0(Q) \mid \mathrm{string}_1(Q) \\
R ::= \;& (R, R) \mid (M, N) \\
S ::= \;& S \mid O(S) \\
M ::= \;& N \mid \mathsf{cmb}(N, M) \qquad N ::= n \qquad n \in \mathbf{N}
\end{aligned}
$$

Figure 2: The syntax of terms

**Decentralised anonymous webs of trust.** Anonymous webs of trust still require a centralized server that provides access to the signatures on the public keys. In a decentralized anonymous webs of trust, participants do not issue signatures but zero-knowledge proofs.

Zero-knowledge proofs used in a completely distributed setting need to have the following properties: (*i*) Charlie needs to be able to combine her proof issued by Bob with a fresh proof that authenticates the message (the combined proof shows $\exists \mathrm{sig}_1, \mathrm{sig}_2.\ \mathsf{verify}(\mathrm{vk}_B, \mathrm{sig}_1, \mathrm{vk}_C) \wedge \mathsf{verify}(\mathrm{vk}_A, \mathrm{sig}_2, m)$; (*ii*) since the combined proof reveals Alice's identity, she needs to be able to selectively hide her identity from the combined proof; (*iii*) since combining proofs and selectively hiding parts of the statements in general leaves the zero-knowledge proof itself unchanged, Alice needs to be able to re-randomize the proof: otherwise, Bob can check for equality of the proof part that he issued with previously issued proofs to identify Alice as originator of the message, as described in Figure 1. Malleable zero-knowledge proofs satisfy all these requirements.

## 2.3 Symbolic Model of Malleable Zero-Knowledge Proofs

This section details the symbolic model of malleable zero-knowledge proofs. For devising this model, we encountered several subtle pitfalls, which we highlight and show how to circumvent.

A symbolic zero-knowledge proof consist of the proof $\pi$ itself (i.e., the concrete bit string), the commitments $c_i$ that are used in the proven statement, and the opening information $o_i$ to some or all of the commitments. This model closely resembles existing zero-knowledge schemes (e.g., [GS08]) and it is capable of satisfying all our requirements. In the following, we denote zero-knowledge proofs as triples $(\pi, \underline{c}, \underline{o})$. For better readability, we refer to $\pi$ as zero-knowledge proof or proof and to the triple $(\pi, \underline{c}, \underline{o})$ as zero-knowledge triple or triple. Furthermore, we let $\varepsilon$ denote empty opening information that have been removed from the proof; removing the opening information directly translates to selectively hiding the corresponding value.

For instance, the symbolic proof sent from Alice to Bob in Figure 1 consists of three combined proofs and looks as follows:

$$
\begin{array}{lll}
(\pi_{A\to B}, & (c_{\mathrm{vk}_A}, c_{\mathrm{sig}_1}, c_{\mathrm{vk}_B}), & (o_{\mathrm{vk}_A}, \varepsilon, \varepsilon)) \\
(\pi_{B\to C}, & (c_{\mathrm{vk}_B}, c_{\mathrm{sig}_2}, c_{\mathrm{vk}_C}), & (\varepsilon, \varepsilon, \varepsilon)) \\
(\pi_m, & (c_{\mathrm{vk}_C}, c_{\mathrm{sig}_3}, c_m), & (\varepsilon, \varepsilon, o_m))
\end{array}
$$

where $c_x$ denotes a commitment to $x$ and $o_y$ the opening information for $y$. Notice that the commitments to $\mathrm{vk}_B$ (respectively, $\mathrm{vk}_C$) in the first and the second (respectively, the second and the third) verification are equal. This is necessary to connect the malleable proofs [MPR13].

The most notable difference between non-malleable and malleable zero-knowledge proofs is the re-randomization property of the latter. Since this influenced the model most, we start by describing the re-randomization property. In the remainder of this section, we let $n$ denote the number of commitments contained in a proof. Furthermore, we write the superscript $i/n$ to denote that the destructor handles the $i$-th of $n$ entries, i.e., commitments or opening information.

**Symbolic re-randomization.** Computationally, re-randomization typically works by adding (or multiplying, depending on the group structure) a uniformly random value $r$ to some value $v$ and, using a one-time pad argument, the resulting value is uniformly random and independent of $v$. Although appealing, modeling such algebraic properties symbolically is inherently unsound [Unr10].

We circumvent this issue by requiring honest protocol participants to always use freshly-chosen random values for the re-randomization process: since the randomness is always fresh and chosen uniformly at random, algebraic properties do not play a role.

At the same time, the model must consider that the attacker cannot be restricted in any way. A first approach is to let re-randomization replace randomness in a zero-knowledge triple consistently in $\pi$ and in $\underline{c}$. Since the attacker can re-randomize a proof and commitments can be opened once the randomness is known, such an approach cannot provide any privacy properties. Another idea is to symbolically combine the randomness. While functional, this approach effectively leads to non-termination in the verification due to the unbounded number of possible combinations that the attacker can derive.

We solve the two aforementioned problems by introducing for every term that contains randomness two different random values: one that can only be modified by honest protocol participants and one that can be arbitrarily modified by the attacker. If a protocol participant applies a re-randomization operation, we distinguish two cases: if the corresponding opening information is available, we replace the existing honest randomness; otherwise, we combine the existing honest randomness and the new randomness using the constructor $\mathsf{cmb}$. Since a protocol only comprises a finite number of re-randomization operations, the number of nested $\mathsf{cmb}$ constructors is bounded, making automated verification viable. If the attacker applies re-randomization, we replace the corresponding attacker randomness.

This unusual treatment of the honest randomness is necessary to obtain computational soundness while still being able to connect the malleable proofs [MPR13].

At first, this over-approximation seems to let the attacker selectively replace the randomness and, consequently, learn the committed values used in a proof. The attacker, however, cannot touch the honest randomness. Consequently, the attacker can only know the complete randomness (honest and attacker randomness) of a commitment, if the randomness was revealed in the first place.

We describe the re-randomization destructors after we described the symbolic model of all the zero-knowledge proof ingredients.

**Modeling symbolic zero-knowledge proofs.** The next step is to model the proof $\pi$. Consider the following scenario: the attacker has a valid proof $(\pi, (c), (\varepsilon))$. Furthermore, the attacker has some commitment $c' \neq c$. Notice that even though $c$ is different from $c'$, they may still be commitments to the same value. Our model needs to prevent the following attack: the attacker checks whether the triple $(\pi, (c'), (\varepsilon))$ verifies and, in the process, learns that $c$ and $c'$ are commitments to the same value. This purely symbolic attack does not exist for computational zero-knowledge proofs, thus the model would not be unsound. Leaving it in the symbolic model, however, would virtually break all protocols and therefore render the symbolic model essentially useless.

We solve this problem by incorporating into $\pi$ the randomness of the commitments used in the proven statements. The verification checks that the randomness contained inside the commitments matches the randomness inside $\pi$. If a commitment is exchanged, the randomness in that commitment differs from the randomness contained in $\pi$ and the verification fails (naturally, if the randomness is the same but the committed values are different, the verification fails as well). Additionally, $\pi$ contain its own randomness. This enables us to re-randomize the proof only, without re-randomizing any commitments.

We use the constructor $\mathsf{zkp}$ to construct symbolic zero-knowledge proofs $\pi = \mathsf{zkp}(r_0^H, r_0^A, \ldots, r_n^H, r_n^A)$ where $r_i^H$ and $r_i^A$ denote the honest and the attacker randomness of the $i$-th commitment, respectively; $r_0^H$ and $r_0^A$ denote the honest and the attacker randomness, respectively, of the proof $\pi$ itself.

**Modeling symbolic commitments.** Given the insight from the re-randomization, modeling of commitments is straightforward: we model the commitment $c_i$ to the $i$-th committed message $m_i$ as $c_i = \mathsf{com}(m_i, r_i^H, r_i^A)$. Honest participants put $\varepsilon$ as attacker randomness.

**Modeling symbolic opening information.** Analogously to computational opening information, the symbolic opening information consist of the three arguments to the $\mathsf{com}$ constructor. If the opening information has been removed because the corresponding committed value has been selectively hidden, it consists of the dedicated value $\varepsilon$.

**Modeling symbolic verification.** The symbolic zero-knowledge verification destructor $\mathsf{verzk}$ has to enforce two properties: first, it needs to assert that the randomness in $\pi$ matches the randomness in $\underline{c}$; second, it must ensure, that the proven statement holds. As a result, we cannot state a general verification function. The structure, however, is always the same: the destructor takes as input the proof $\pi$ and the tuple of commitments. The verification of the proven statement is encoded by requiring that the input terms have a given shape. The verification ignores the opening information as it is not necessary to verify a given zero-knowledge proof.

To illustrate the verification mechanism, we detail the zero-knowledge verification for anonymous webs of trust. The destructor looks as follows:

$$\mathsf{verzk}^{\mathrm{AWOT}}\left(\begin{array}{l}\mathsf{zkp}(r^H, r^A, r_{\mathrm{vk}}^H, r_{\mathrm{vk}}^A, r_{\mathrm{sig}}^H, r_{\mathrm{sig}}^A, r_m^H, r_m^A),\\ \left(\begin{array}{l}\mathsf{com}(\mathrm{vk}(s), r_{\mathrm{vk}}^H, r_{\mathrm{vk}}^A),\\ \mathsf{com}(\mathsf{sign}(m, \mathrm{sk}(s)), r_{\mathrm{sig}}^H, r_{\mathrm{sig}}^A),\\ \mathsf{com}(m, r_m^H, r_m^A)\end{array}\right)\end{array}\right)$$

The equality on the randomness is enforced as expected. The signature verification $\mathsf{verify}(m, \mathsf{sign}(m, \mathrm{sk}(s)), \mathrm{vk}(s))$ is encoded inside the commitments.

**Symbolic selective hiding destructors.** The destructor $\mathsf{hide}^{i/n}$ for selectively hiding a value is straightforward: the opening information of the respective commitment is replaced with the special value $\varepsilon$. The destructor equations are shown in Table 1.

**Symbolic re-randomization destructors.** The destructors need to treat honest and attacker randomness in different ways. We accommodate the distinction between the two kinds of randomness by introducing two destructors for re-randomization: $\mathsf{rerand}^{i/n}$ that can only be used by honest protocol participants and $\mathsf{att\text{-}rerand}^{i/n}$ that give the attacker access to the re-randomization operations. The proof $\pi$ itself is re-randomized using $i = 0$.

**Modeling symbolic verification.** The symbolic zero-knowledge verification destructor $\mathsf{verzk}$ consists of two parts: first, it needs to enforce that the proof $\pi$ matches the commitments $\underline{c}$; second, it must ensure, that the proven statement holds true. As a result, we cannot state a general verification function. The structure, however, is always the same: the destructor takes as input the proof $\pi$ and the tuple of commitments. The verification of the proven statement is encoded by enforcing that the input terms have a given shape. Notice that the verification ignores the opening information as it is not necessary to verify a given zero-knowledge proof.

To illustrate the verification mechanism, we detail the zero-knowledge verification for anonymous webs of trust. Naturally, the verification must ensure that the randomnesses contained in $\pi$ and in the commitments match. Additionally, the verification must ensure, that the signature verification succeeds. The destructor looks as follows:

$$\mathsf{verzk}^{\mathrm{AWOT}}\left(\begin{array}{l}\mathsf{zkp}(r^H, r^A, r_{\mathrm{vk}}^H, r_{\mathrm{vk}}^A, r_{\mathrm{sig}}^H, r_{\mathrm{sig}}^A, r_m^H, r_m^A),\\ \left(\begin{array}{l}\mathsf{com}(\mathrm{vk}(s), r_{\mathrm{vk}}^H, r_{\mathrm{vk}}^A),\\ \mathsf{com}(\mathsf{sign}(m, \mathrm{sk}(s)), r_{\mathrm{sig}}^H, r_{\mathrm{sig}}^A),\\ \mathsf{com}(m, r_m^H, r_m^A)\end{array}\right)\end{array}\right)$$

The equality on the randomness is enforced as expected. The signature verification $\mathsf{verify}(m, \mathsf{sign}(m, \mathrm{sk}(s)), \mathrm{vk}(s))$ is encoded inside the commitments.

Destructors

Re-randomization, if the opening information is available

$\mathsf{rerand}^i((\mathsf{zkp}(r_j^H, r_j^A)_{j=0}^n, (\mathsf{com}(m_j, r_j^H, r_j^A))_{j=1}^n, (o_j)_{j=1}^n), r^{H\prime}) = (\mathsf{zkp}(r_j^{H\prime}, r_j^{A\prime})_{j=0}^n, (\mathsf{com}(m_j, r_j^{H\prime}, r_j^{A\prime}))_{j=1}^n, (o'_j)_{j=1}^n)$
where $r_j^{H\prime} := r_j^H$, $r_j^{A\prime} := r_j^A$, and $o'_j := o_i$ for $i \neq j$; $r_i^{H\prime} := r^{H\prime}$, $r_i^{A\prime} := r_i^A$, and $o'_i := (m_i, r_i^{H\prime}, r_i^A)$ if $o_i = (m_i, r_i^H, r_i^A)$

Re-randomization, if the opening information has been removed

$\mathsf{rerand}^i((\mathsf{zkp}(r_j^H, r_j^A)_{j=0}^n, (\mathsf{com}(m_j, r_j^H, r_j^A))_{j=1}^n, (o_j)_{j=1}^n), r^{H\prime}) = (\mathsf{zkp}(r_j^{H\prime}, r_j^{A\prime})_{j=0}^n, (\mathsf{com}(m_j, r_j^{H\prime}, r_j^{A\prime}))_{j=1}^n, (o'_j)_{j=1}^n)$
where $r_j^{H\prime} := r_j^H$, $r_j^{A\prime} := r_j^A$, and $o'_j := o_i$ for $i \neq j$; $r_i^{H\prime} := \mathsf{cmb}(r_i^H, r^{H\prime})$, $r_i^{A\prime} := r_i^A$, and $o'_i := \varepsilon$ if $o_i = \varepsilon$

$\mathsf{att\text{-}rerand}^i((\mathsf{zkp}(r_j^H, r_j^A)_{j=0}^n, (\mathsf{com}(m_j, r_j^H, r_j^A))_{j=1}^n, (o_j)_{j=1}^n), r^{A\prime}) = (\mathsf{zkp}(r_j^{H\prime}, r_j^{A\prime})_{j=0}^n, (\mathsf{com}(m_j, r_j^{H\prime}, r_j^{A\prime}))_{j=1}^n, (o'_j)_{j=1}^n)$
where $r_j^{H\prime} := r_j^H$, $r_j^{A\prime} := r_j^A$, and $o'_j := o_i$ for $i \neq j$; $r_i^{H\prime} := r_i^H$, $r_i^{A\prime} := r^{A\prime}$
and $o'_i := (m_i, r_i^H, r_i^{A\prime})$ if $o_i = (m_i, r_i^H, r_i^A)$ and $o'_i := \varepsilon$ if $o_i = \varepsilon$

$\mathsf{hide}^i((\mathsf{zkp}(r_j^H, r_j^A)_{j=0}^n, (\mathsf{com}(m_j, r_j^H, r_j^A))_{j=1}^n, (o_j)_{j=1}^n)) = (\mathsf{zkp}(r_j^H, r_j^A)_{j=0}^n, (\mathsf{com}(m_j, r_j^H, r_j^A))_{j=1}^n, (o'_j)_{j=1}^n)$
where $o'_i := o_j$ for $i \neq j$ and $o_i := \varepsilon$

$\mathrm{open}(\mathrm{com}(m, r^H, r^A), (m, r^H, r^A)) = m$

$\mathrm{iscom}(\mathrm{com}(m, r^H, r^A)) = \mathrm{com}(m, r^H, r^A)$

Table 1: Set of constructor and destructor equations for a zero-knowledge proof that contains $n$ commitments.

**Symbolic selective hiding destructors.** The destructor $\mathsf{hide}^{i/n}$ for selectively hiding a value is straightforward: the opening information of the respective commitment is replaced with the special value $\varepsilon$. The superscript $i/n$ denotes that the destructor removes the $i$-th opening information of a proof that contains $n$ commitments. The destructor equations are shown in Table 1.

**Symbolic re-randomization destructors.** The destructors need to treat honest and attacker randomness in different ways. We accommodate the distinction between the two kinds of randomness by introducing two destructors for re-randomization: $\mathsf{rerand}^{i/n}$ that can only be used by honest protocol participants and $\mathsf{att\text{-}rerand}^{i/n}$ that give the attacker access to the re-randomization operations. The superscript $i/n$ denotes that the destructor is used to re-randomize the $i$-th commitment of a proof that contains $n$ commitments; the proof $\pi$ itself is re-randomized using $i = 0$.

**Symbolic length.** We standard the same notion of symbolic length as in [BMR14a]. Length is modelled in a peano arithmetic manner: by the two constructors S and O(S).

Figure 2 depicts the full syntax of our symbolic model. Table 1 depicts the destructors connected to zero-knowledge proofs. The destructors for all other terms are as in the basic model of the full version [ful] and similar to previous work [BHU09, BMR14a].

**Difference to the soundness result's symbolic model.** The symbolic model presented here differs from the one used for the computational soundness result. The sound model is more complex in the sense that it allows arbitrary logical operations on statements whereas we restrict the model in the body of the paper to a specific class of statements. The advantage of the restriction is that it allows ProVerif to terminate, however, we additionally need to show that we can reduce the model used in the paper's body to the generalized one. This is done in Appendix D. The intuition behind the reduction is that we cannot construct proofs for certain statements, but also our verification of these proofs fails in the simpler model.

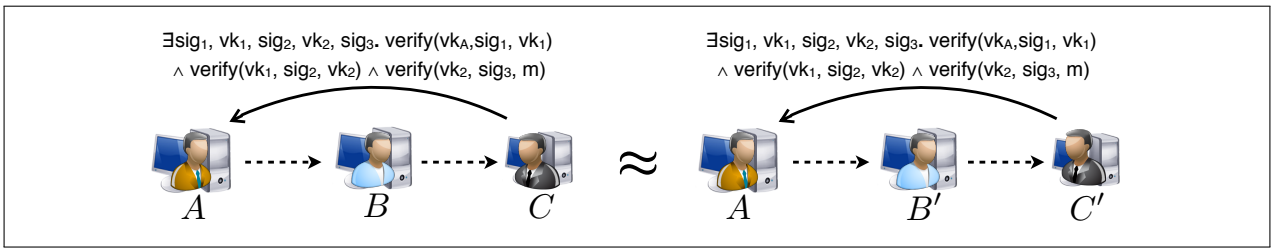


Figure 3: Anonymity game. A dashed arrow from principal $I$ to principal $J$ denotes that $I$ signed the public-key pair of $J$. A solid arrow from principal $I$ to $J$ denotes that $I$ send a zero-knowledge proof for the shown statement to $J$.

## 2.4 Verification

In this section, we focus of the verification of anonymity properties. We model the decentralized anonymous webs of trust as an bi-process in the applied pi-calculus as described in § 2.1: two distinguished, honest principals act in a web of trust set up by the attacker and one of them authenticates a message by proving a trust chain chosen by the attacker. If the attacker cannot guess which of the two principals generated the corresponding proof, then the protocol guarantees anonymity. Our model includes an arbitrary number of honest and compromised parties as well as the two (honest) principals engaging in the anonymity game. Due to limitations of ProVerif, we concentrate on chains of length 3 and on a setting without re-randomization. Concerning the length of the chains, there is a straight-forward interpretation for chains of length 3: "I trust some who certifies the trustworthiness of a verification key." For longer chains, the trust guarantees are hard to interpret. Concerning the lack of re-randomization, the case study does not include protocol re-randomization, but the protocol does include the more verification-intensive re-randomization rules for the attacker.

For a verification of the fully fledged protocol, stronger verification tools are required, e.g., the Tamarin prover can be given invariants that simplify the analysis of larger protocols and an extension to observational equivalence has been announced [MSC+].[4] Moreover, this simplified model only allows a bounded length for verification chains, which should still provide a useful protocol for many practical purposes.

The anonymity game is defined by two distinct processes that are replicated (that is, spawned an unbounded number of times) and in parallel composition (i.e., concurrently executed). In the first process, each of the two distinguished principals signs public-key pairs as dictated by the attacker. Since the attacker controls also the digital signatures released by the other parties in the system, both honest and compromised ones, the attacker controls the topology of the whole web of trust. In the second process, the two distinguished principals receive two (possibly different) certificate chains from the attacker. If both certificate chains are valid and of the same length, we non-deterministically choose one of the two principals $C$ and $C'$, and we let it output the corresponding proof. The observational equivalence relation $\approx$ (see Figure 3) says that the attacker should not be able to determine whether model $\mathcal{M}_1^{\mathsf{Anon}}$ in which $C$ outputs the proof or $\mathcal{M}_2^{\mathsf{Anon}}$ in which $C'$ outputs the proof is being executed. In other words, the attacker cannot tell if $C$ or $C'$ generated the proof, i.e., they are anonymous.

**Theorem 2** (Anonymity)**.** *For the two processes $\mathcal{M}_1^{\mathsf{Anon}}$ and $\mathcal{M}_2^{\mathsf{Anon}}$, the observational equivalence relation $\mathcal{M}_1^{\mathsf{Anon}} \approx \mathcal{M}_2^{\mathsf{Anon}}$ holds true.*

*Proof.* This statement is proven using ProVerif. The scripts can be found online [cas]. □

**Discussion.** Malleable zero-knowledge models are conceptually simple and often yields more compact and cleaner models than non-malleable counterparts. The reason is that malleability requires only the modeling of the atomic proofs occurring in the protocol. Complex proofs can be assembled from the atomic proofs, exploiting the malleable nature. In the non-malleable case, proofs cannot be combined and all used constellations of atomic proofs occurring in the protocol have to be explicitly considered in the model. This conceptual simplicity becomes apparent in the anonymous webs of trust case because the protocol relies solely on signature verifications. Using the malleability property, these verifications are stringed-together to prove

[4] For Tamarin, the uniformity would either have to be proven by hand or by lemmas that are formulated in Tamarin.

trust chains. The malleable nature, however, causes problems in the termination behaviour of automated verification techniques.

Malleable proofs are designed to be separable and arbitrarily combinable. As already hinted in Appendix A, the resulting sheer number of possible combinations causes problems for automated verification techniques underlying such as ProVerif. The biggest performance impact, however, is credited to the re-randomization: the model without re-randomization terminates within a few minutes while the model with re-randomization requires several days.

# 3 Reviewing the CoSP framework for equivalence properties

The abstraction and the computational soundness result put forward in this work are cast in CoSP [BHU09, BMR14a], a framework for symbolic protocol analyses and conceptually modular computational soundness proofs that hold for several languages, such as the applied pi-calculus and RCF [BMU10] (a core calculus of ML). In this section, we review the basic concepts underlying the CoSP framework.

Using the self-monitoring technique, we will modularize our computational soundness proof by first proving computational soundness w.r.t. trace properties and then imply (via self-monitoring) that computational soundness w.r.t. equivalence properties follows.

**Symbolic model.** In CoSP, symbolic abstractions of protocols and, more importantly, of the attacker, are formulated in a symbolic model $M = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$: a set of free functions $\mathbf{C}$, a countably infinite set $\mathbf{N}$ of nonces, a set $\mathbf{T}$ of terms, a set $\mathbf{D}$ of partial mappings from terms to terms (called destructors), and a *deduction relation* $\vdash$ between $2^{\mathbf{T}}$ and $\mathbf{T}$. In order to unify the notation, we write $\mathrm{eval}_F(t) := F(t)$ if $F \in \mathbf{D}$ or $F \in \mathbf{C}$ and $F(t) \in \mathbf{T}$, otherwise $\mathrm{eval}_F(t) := \bot$.

**Protocols.** In CoSP, protocols are represented as (possibly infinite) trees. Each node in this tree represents an action in the protocol: *computation nodes* are used for drawing fresh nonces, applying constructors, and applying destructors; *input* and *output nodes* are used for sending and receiving operations; *nondeterministic nodes* are used for modeling nondeterministic choices in the protocol; *control nodes* are used for allowing the attacker to schedule some parts of the protocol. A computation node is annotated with its arguments and has two outgoing edges: a yes edge, used for the application of constructors, for drawing a nonce, and for the successful application of a destructor, and a no edge, used if an application of a destructor fails.[5]

## 3.1 Computational soundness w.r.t. trace properties

As a first step, we recall the definition of computational soundness w.r.t. trace properties.

**Trace properties.** A trace property is a prefix-closed set of node-traces. We say that a protocol $\Pi$ *symbolically satisfies* a trace property $\mathbf{p}$ if for all traces $t$ resulting from a symbolic execution, $\mathbf{p}$ holds for $t$, i.e., $t \in \mathbf{p}$. Correspondingly, we state that a protocol $\Pi$ *computationally satisfies* a trace property $\mathbf{p}$ if for any ppt attacker $M$ the probability is overwhelming that $\mathbf{p}$ holds for $t$, i.e., $t \in \mathbf{p}$, for any trace $t$ resulting from a computational execution (with $M$).

**Computational soundness.** Computational soundness establishes a relation between the symbolic abstraction of a protocol and the cryptographic implementation thereof. Roughly, such a result states that all attacks that can be ruled out for the symbolic abstraction can be ruled out for the cryptographic implementation as well.

**Definition 2** (Computational soundness)**.** *A symbolic model* $(\boldsymbol{C}, \boldsymbol{N}, \boldsymbol{T}, \boldsymbol{D}, \vdash)$ *is* computationally sound *with respect to an implementation* $A$ *for a class* $C$ *of protocols if the following holds: for each protocol* $\Pi \in C$ *and each trace property* $\mathbf{p}$*, if* $\Pi$ *symbolically satisfies* $\mathbf{p}$ *then* $\Pi$ *computationally satisfies* $\mathbf{p}$*.*

[5] The application $D(M)$ of a destructor $D$ fails if the destructor is undefined on $M$.

**A simulation-based proof technique.** Proving computational soundness in CoSP is done by constructing a simulator that behaves as the computational execution towards a computational attacker and as a valid symbolic attacker towards a modified symbolic execution, called the *hybrid execution* [BHU09]. Upon receiving a term from the symbolic execution, the simulator translates this term $t$ as a bitstring $\beta(t)$ according to a so-called *construction function* $\beta : \mathbf{T} \to \{0,1\}^*$ and forwards this bitstring $\beta(t)$ to the attacker. Upon receiving a bitstring $m$ from the attacker, the simulator parses $m$ according to a so-called *parsing function* $\tau : \{0,1\}^* \to \mathbf{T}$ and forwards $\tau(m)$ to the symbolic execution. Computational soundness follows if the simulator satisfies two properties: *Indistinguishability*, i.e., the interaction of the attacker with the simulator is indistinguishable for from attacker's interaction with the real computational execution of the protocol. And *Dolev-Yaoness*, i.e., for each message $m$ from the attacker, and for the set $S$ of terms received so far by the symbolic execution, $S \vdash \tau(m)$ holds with overwhelming probability.

## 3.2 Symbolic indistinguishability

In this section, we define a symbolic notion of indistinguishability. Our results hold for pairs of protocols. We represent such pairs of protocols as so-called bi-protocols, which are pairs of protocols that are encoded in one extended CoSP protocol tree.

**Bi-protocols.** To compare two variants of a protocol, we consider bi-protocols, which rely on the same idea as bi-processes in the applied pi-calculus [BAF08]. Bi-protocols are pairs of protocols that only differ in the messages they operate on.

**Definition 3** (CoSP bi-protocol)**.** *A* CoSP bi-protocol $\Pi$ *is defined like a protocol but uses bi-references instead of references. A* bi-reference *is a pair* $(\nu_{left}, \nu_{right})$ *of node identifiers of two (not necessarily distinct) nodes in the protocol tree. In the* left protocol *left*($\Pi$) *the bi-references are replaced by their left components; the* right protocol *right*($\Pi$) *is defined analogously.*

As a next step, we model the capabilities of the symbolic attacker. In contrast to an attacker against trace properties, we do not only have to capture which messages the attacker can derive, via the deduction relation, but also which protocol messages the attacker observes, in particular in which order an attacker observe these messages. Moreover, it has to be captured which tests that an attacker can perform in order to judge whether two protocols are distinguishable. These tests, called *symbolic operations*, capture the sequence of operations a symbolic attacker applies, including the protocol messages that are used.

**Definition 4** (Symbolic operation)**.** *Let* $\mathsf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D})$ *be a symbolic model. A* symbolic operation $O$ of arity $n$ on $\mathbf{M}$ *is a finite tree whose nodes are labeled with constructors from* $\mathbf{C}$*, destructors from* $\mathbf{D}$*, nonces from* $\mathbf{N}$*, and formal parameters* $x_i$ *with* $i \in \{1, \ldots, n\}$*. For constructors and destructors, the children of a node represent its arguments (if any). Formal parameters* $x_i$ *and nonces do not have children.*

*We extend the evaluation function to symbolic operations. Given a list of terms* $\underline{t} \in \mathbf{T}^n$*, the evaluation function* $eval_O : \mathbf{T}^n \to \mathbf{T}$ *recursively evaluates the tree* $O$ *starting at the root as follows: The formal parameter* $x_i$ *evaluates to* $t_i$*. A node with* $F \in \mathbf{C} \cup \mathbf{N}_E \cup \mathbf{D}$ *evaluates according to* $eval_F$*. If there is a node that evaluates to* $\bot$*, the whole tree evaluates to* $\bot$*.*

Note that the identity function is included. It is the tree that contains only $x_1$ as node.

A symbolic execution of a protocol is basically a valid path through the protocol tree. It induces a *view*, which contains the communication with the attacker. Together with the symbolic execution, we define an *attacker strategy* as the sequence of symbolic operations that the attacker performs.

**Definition 5** (Symbolic execution)**.** *Let a symbolic model* $\mathsf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D})$ *and a CoSP protocol* $\Pi$ *be given. A* symbolic view *of the protocol* $\Pi$ *is a (finite) list of triples* $(V_i, \nu_i, f_i)$ *with the following conditions:*
*For the first triple, we have* $V_1 = \varepsilon$*,* $\nu_1$ *is the root of* $\Pi$*, and* $f_1$ *is an empty partial function, mapping node identifiers to terms. For every two consecutive tuples* $(V, \nu, f)$ *and* $(V', \nu', f')$ *in the list, let* $\underline{\tilde{\nu}}$ *be the nodes referenced by* $\nu$ *and define* $\underline{\tilde{t}}$ *through* $\tilde{t}_j := f(\tilde{\nu}_j)$*. We conduct a case distinction over* $\nu$ *for defining valid successors* $\nu', f', V'$ *as depicted in Figure 4.*

*Here,* $V_{Out}$ *denotes the list of terms in* $V$ *that have been sent at output nodes, i.e., the terms* $t$ *contained in entries of the form* $(\mathtt{out}, t)$ *in* $V$*. Analogously,* $V_{Out\text{-}Meta}$ *denotes the list of out-metadata in* $V$ *that has been sent at control nodes.*

*The set of all symbolic views of* $\Pi$ *is denoted by* SViews($\Pi$)*.*

**switch** $\nu$ **with**
  **case** "$\nu$ *is a computation node with constructor, destructor or nonce* $F$."
    Let $V' = V$.
    **if** $m \coloneqq \mathrm{eval}_F(\tilde{t}) \neq \bot$ **then**
      $\nu'$ is the `yes`-successor of $\nu$ in $\Pi$, and $f' = f(\nu \coloneqq m)$.
    **else**
      $\nu'$ is the `no`-successor of $\nu$, and $f' = f$.
  **case** "$\nu$ *is an input node.*"
    **if** there exists a term $t \in \mathbf{T}$ and a symbolic operation $O$ on $\mathsf{M}$ with $\mathrm{eval}_O(V_{Out}) = t$ **then**
      Let $\nu'$ be the successor of $\nu$ in $\Pi$, $V' = V :: (\mathtt{in}, (t, O))$, and $f' = f(\nu \coloneqq t)$.
  **case** "$\nu$ *is an output node.*"
    Let $V' = V :: (\mathtt{out}, \tilde{t}_1)$, $\nu'$ is the successor of $\nu$ in $\Pi$, and $f' = f$.
  **case** "$\nu$ *is a control node with out-metadata* $l$."
    **if** there is no edge with label $l'$ **then**
      Let $\nu'$ be the lexicographically smallest edge.
    **else**
      Let $\nu'$ be the successor of $\nu$ with the in-metadata $l'$.
    Let $f' = f$, and $V' = V :: (\mathtt{control}, (l, l'))$.

Figure 4: Symbolic execution

*Furthermore, $V_{In}$ denotes the partial list of $V$ that contains only entries of the form $(\mathtt{in}, (*, O))$ or $(\mathtt{control}, (*, l'))$ for some symbolic operation $O$ and some in-metadata $l'$, where the input term and the out-metadata have been masked with the symbol $*$. The list $V_{In}$ is called* attacker strategy. *We write $[V_{In}]_{\mathrm{SViews}(\Pi)}$ to denote the class of all views $U \in \mathrm{SViews}(\Pi)$ with $U_{In} = V_{In}$.*

The symbolic knowledge of the attacker comprises the results of all the symbolic tests the attacker can perform on the messages output by the protocol. The definition captures that the attacker knows exactly which symbolic operation leads to which result. This is essential to reason about indistinguishability: consider a bi-protocol such that the left protocol sends the pair $(n, t)$, but the right protocol sends the pair $(t, n)$, where $t$ is again a publicly known term and $n$ is a fresh protocol nonce. The two protocols do not differ in the terms that the attacker can deduce after their execution; the deducible terms are all publicly known terms as well as $n$. Still, the protocols are trivially distinguishable by the symbolic operation $\mathsf{equals}(O_t, \mathrm{snd}(x_1))$ because $\mathsf{equals}(O_t, \mathrm{snd}((n, t))) \neq \bot$ but $\mathsf{equals}(O_t, \mathrm{snd}((t, n))) = \bot$, where snd returns the second component of a pair and $O_t$ is a symbolic operation that constructs $t$.

**Definition 6** (Symbolic knowledge)**.** *Let* $\mathbf{M}$ *be a symbolic model. Given a view $V$ with $|V_{Out}| = n$, a* symbolic knowledge function $f_V : \mathsf{setSymbolicOperations}(\mathsf{M})_n \to \{\top, \bot\}$ *is a partial function from symbolic operations (see Definition 4) of arity $n$ to $\{\top, \bot\}$. The* full symbolic knowledge function *is a total symbolic knowledge function is defined by*

$$K_V(O) \coloneqq \begin{cases} \bot & \text{if } eval_O(V_{Out}) = \bot \\ \top & \text{otherwise.} \end{cases}$$

Intuitively, we would like to consider two views *equivalent* if they look the same for a symbolic attacker. Despite the requirement that they have the same order of output, input and control nodes, this is the case if they agree on the out-metadata (the control data sent by the protocol) as well as the symbolic knowledge that can be gained out of the terms sent by the protocol.

**Definition 7** (Equivalent views)**.** *Let two views $V, V'$ of the same length be given. We denote their $i$th entry by $V_i$ and $V'_i$, respectively. $V$ and $V'$ are* equivalent *($V \sim V'$), if the following three conditions hold:*

1. *(Same structure) $V_i$ is of the form $(s, \cdot)$ if and only if $V'_i$ is of the form $(s, \cdot)$ for some $s \in \{\mathtt{out}, \mathtt{in}, \mathtt{control}\}$.*
2. *(Same out-metadata) $V_{Out\text{-}Meta} = V'_{Out\text{-}Meta}$.*
3. *(Same symbolic knowledge) $K_V = K_{V'}$.*

Finally, we define two protocols (e.g., the left and right variant of a bi-protocol) to be symbolically indistinguishable if its two variants lead to equivalent views when faced with the same attacker strategy. Thus, a definition of "symbolic indistinguishability" should compare the symbolic knowledge of two protocol runs only if the attacker behaves identically in both runs.

**Definition 8** (Symbolic indistinguishability)**.** *Let* $\mathsf{M}$ *be a symbolic model and* P *be a class of protocols on* $\mathsf{M}$*. Given an attacker strategy* $V_{In}$ *(in the sense of Definition 5), two protocols* $\Pi_1, \Pi_2 \in \mathrm{P}$ *are* symbolically indistinguishable under $V_{In}$ *if for all views* $V_1 \in [V_{In}]_{\mathrm{SViews}(\Pi_1)}$ *of* $\Pi_1$ *under* $V_{In}$*, there is a view* $V_2 \in [V_{In}]_{\mathrm{SViews}(\Pi_2)}$ *of* $\Pi_2$ *under* $V_{In}$ *such that* $V_1 \sim V_2$*, and vice versa.*

*Two protocols* $\Pi_1, \Pi_2 \in \mathrm{P}$ *are* symbolically indistinguishable *(*$\Pi_1 \approx_s \Pi_2$*), if* $\Pi_1$ *and* $\Pi_2$ *are indistinguishable under all attacker strategies. For a bi-protocol* $\Pi$*, we say that* $\Pi$ *is symbolically indistinguishable if* $\mathit{left}(\Pi) \approx_s \mathit{right}(\Pi)$*.*

## 3.3 Computational Indistinguishability

On the computational side, the constructors and destructors in a symbolic model are realized with cryptographic algorithms, which we call computational implementations. These implementations consists of deterministic polynomial-time algorithms and probabilistic polynomial-time algorithms in the case of nonces. The *computational execution* is defined analogously to the symbolic execution where every symbolic application of constructors, destructors and nonces is replaced by the corresponding algorithm and the interaction occurs with respect to an probabilistic polynomial-time adversary instead of a symbolic attacker. The *computational challenger* is canonically defined by the execution of a probabilistic machine that traverses the tree and interacts with an arbitrary ppt attacker: at a computation node the corresponding algorithm is run and depending on whether the algorithm succeeds or fails[6] either the yes-branch or the no-branch is taken; at an output node, the message is sent to the attacker, and at an input node a message is received by the attacker; at a nondeterministic node a random choice is performed; at a control node the attacker sends a command that specifies which branch to take.

**Efficient CoSP (bi-)protocols.** We will use the CoSP execution in cryptographic reductions; however, CoSP (bi-)protocols are by definition infinite. Hence, as is standard in cryptography, we need to ensure the execution of a CoSP (bi-)protocols is computable in polynomial-time. To this end, we require that it is possible to incrementally compute the node information, i.e., the node identifier, for each path through the CoSP (bi-)protocol in polynomially many steps (in the length of the path from the root node to the current node). We call such (bi-)protocols efficient.

In contrast to the symbolic executions, the computational execution relies on time, i.e., the computational adversary can distinguish using termination time. This problem was analysed in [Unr11] and led to the definition termination-insensitive computational indistinguishability (tic-indistinguishability) where two executions are denoted as tic-indistinguishable by $A \approx_{tic} B$. The basic idea is to bound the probability that different outputs are made by $A$ and $B$ within polynomially many steps. We shifted the detailed definition to Appendix G.

Given the tic-indistinguishability, computational indistinguishability for bi-protocols is naturally defined. A bi-protocol is indistinguishable if its challengers are computationally indistinguishable for every ppt attacker $E$.

**Definition 9** (Computational indistinguishability)**.** *Let* $\Pi$ *be an efficient CoSP bi-protocol and let* $A$ *be a computational implementation of* $\mathbf{M}$*.* $\Pi$ *is* (termination-insensitively) computationally indistinguishable *if for all ppt attackers* $E$ *and for all polynomials* $p$*,*

$$\mathrm{Exec}_{A,\mathsf{M},\mathit{left}(\Pi)} \approx_{tic} \mathrm{Exec}_{A,\mathsf{M},\mathit{right}(\Pi)}.$$

## 3.4 Computational Soundness for equivalence properties

The previous definitions culminate in the definition of CS for equivalence properties. It states that the symbolic indistinguishability of a bi-protocol implies its computational indistinguishability.

**Definition 10** (Computational soundness)**.** *Let a symbolic model* $\mathbf{M}$ *and a class* P *of efficient bi-protocols be given. An implementation* $A$ *of* $\mathbf{M}$ *is* computationally sound *for* $\mathsf{M}$ *if for every* $\Pi \in \mathrm{P}$*, we have that* $\Pi$ *is computationally indistinguishable whenever* $\Pi$ *is symbolically indistinguishable.*

**Definition 11** (Symbolic execution of a parameterized CoSP protocol)**.** *Let* $\Pi$ *be a parameterized CoSP protocol and* $f_{par}$ *be a total function* $\mathit{par}(\mathsf{PiS}) \to \mathbf{N}$*. A* full trace of $\mathsf{PiS}$ with parameters $f_{par}$ *is defined like*

[6] An algorithm that fails is considered to output $\bot$.

*a full trace of* PiS *(according to the standard symbolic CoSP execution) with the modification that parameter identifiers are treated like node identifiers and $f_1$ in the first list entry is $f_{par}$ instead of the empty function.*

**Definition 12** (Computational execution of a parameterized CoSP protocol)**.** *Let* $\Pi$ *be a parameterized CoSP protocol and $f_{par}$ be a function par(*PiP*) $\to \{0,1\}^*$. The computational execution of* PiP *with parameters $f_{par}$ is the standard computational CoSP execution with the modification that parameter identifiers are treated like node identifiers and in the initial state $f$ is set to $f_{par}$ instead of the empty function.*

### 3.5 From trace properties to uniformity of bi-protocols

For the connection to trace properties, we concentrate on *uniform* bi-protocols. A bi-protocol is uniform if for each symbolic attacker strategy, both its variants reach the same nodes in the CoSP tree, i.e., they never branch differently. As shown in previous work [BMR14a], uniformity of bi-protocols in CoSP corresponds to uniformity of bi-processes in the applied pi-calculus, i.e., the equivalence property that ProVerif can verify.

**Self-monitor.** The key observation for the connection to trace properties is that, given a bi-protocol $\Pi$, some computationally sound symbolic models allow to construct a self-monitor protocol $\mathsf{Mon}(\Pi)$ (not a bi-protocol!) that has essentially the same interface to the attacker as the bi-protocol $\Pi$ and checks at run-time whether $\Pi$ would behave uniformly. In other words, non-uniformity of bi-protocols can be formulated as a trace property bad, which can be detected by the protocol $\mathsf{Mon}(\Pi)$.

The self-monitor $\mathsf{Mon}(\Pi)$ is basically constructed as an intern execution of the other bi-protocol by the simulator, that outputs a distinguishing event bad whenever the bi-protocols could be distinguished. This basically leads to a reduction from observational equivalence — in the case of bi-protocols — to a trace property. The detailed construction of $\mathsf{Mon}(\Pi)$ needs many technical details such as ensuring that the protocol does something after outputting bad since CoSP protocols are defined as infinite trees. However, these details do not give particularly more insights and are explained in detail in [BMR14a]. As a consequence, we postponed the construction of $\mathsf{Mon}(\Pi)$ to the Appendix G.2 and define here only the final properties that we require in order to reduce observational equivalence to trace equivalence.

**Definition 13** (Distinguishing self-monitors)**.** *Let* M *be a symbolic model and $A$ a computational implementation of* **M***. Let* $\Pi$ *be a bi-protocol and* $\mathsf{Mon}(\Pi)$ *its self-monitor. Let $e \in$* {bad-knowledge, bad-branch} *and $n_{\text{bad-knowledge}}$ denote the node type output node and $n_{\text{bad-branch}}$ denote the node type control node. Then the function $f_{e,\Pi}(b, tr)$, which takes as input $b \in \{left, right\}$ and the path to the root node, including all node and edge identifiers, is a* distinguishing self-monitor for e *for* $\Pi$ *and* M *if it is computable in deterministic polynomial time, and if the following conditions hold for every $i \in \mathbb{N}$:*

1. symbolic self-monitoring: *If $\Pi_i$ is symbolically indistinguishable,* bad *does symbolically not occur in $\mathsf{Mon}(\Pi_{i-1})$, and the $i$th node in $\Pi_i$ is of type $n_e$, then the event $e$ does not occur symbolically in $\mathsf{Mon}(\Pi_i)$.*
2. computational self-monitoring: *If the event $e$ in $\mathsf{Mon}(\Pi_i)$ occurs computationally with at most negligible probability, $\Pi_{i-1}$ is computationally indistinguishable, and the $i$th node in $\Pi_i$ is of type $n_e$, then $\Pi_i$ is computationally indistinguishable.*

*We say that a* M *and a protocol class* allow for self-monitoring *if for every bi-protocol* $\Pi$ *in the protocol class* P*, there are distinguishing self-monitors for* bad-branch *and* bad-knowledge *such that the resulting self-monitor* $\mathsf{Mon}(\Pi)$ *is also in* P*.*

**Theorem 3.** *Let* M *be a symbolic model and* P *be an efficient uniformity-enforcing*[7] *class of bi-protocols. If* M *and* P *allow for self-monitoring (in the sense of Definition 13), then the following holds: If $A$ is a computationally sound implementation of a symbolic model* M *with respect to trace properties then $A$ is also a computationally sound implementation with respect to equivalence properties.*

## 4 Computational soundness

In this section, we show the computational soundness of our symbolic model using the CoSP framework. We establish our computational soundness in three steps. We first identify necessary restrictions on the class of

[7]The property of uniformity-enforcing is basically a syntactic property and required to construct $\mathsf{Mon}(\Pi)$, cf. Appendix G.2

protocols for which we can show computational soundness , e.g., different encryptions always use different randomness(Section 4.1). We then introduce necessary implementation conditions for computationally sound realizations (Section 4.2). Finally, we conduct the actual proof of computational soundness (Section 4.3). Due to space constraints, we primarily elaborate on the protocol restrictions and the implementation conditions, as these illustrate the major insights how to achieve computational soundness for MZK proofs.

## 4.1 MZK-safe protocols

We first characterize the class of computationally sound protocols, which we call MZK-safe. Mainly, our protocol conditions exclude adaptive corruption and regulate the usage of randomness. In this section, we concentrate on the most insightful conditions. To prevent standard problems of adaptive corruption, we disallow decryption keys to be sent over the network. Similarly, we require that the plaintext message of a commitment is either publicly known or secret for the entire execution by imposing the condition that unveiling information $\mathrm{uv}(m, r)$ of a commitment $\mathrm{com}(\mathrm{crs}(n), m, r)$ is only sent over the network as part of an MZK proof. In other words, we basically restrict the usage of commitments to MZK proofs. Moreover, for MZK proofs generated by honest protocol participants, we require that all commitments of one ZK proof use the same honestly generated CRS. Finally, we limit the way in which randomness can be reused. Recall that we distinguish nonces and randomness terms, which are used as randomness in the computation of cryptographic terms, such as encryptions, signatures, and MZK proofs. We only allow reusing randomness terms as witnesses of MZK proofs. For technical reasons, we exclude protocols that reuse the randomness of signatures as witnesses in MZK proofs. However, this restriction is not severe in practice, since we are not aware of any protocol that uses the signature randomness in an ZK proof.

Disallowing the re-usage of randomness completely, however, would result in excluding statements about ownership of ciphertexts, i.e., statements of the form $\exists r, m.c = \mathrm{enc}(ek, m, r)$. There are IND-CCA secure encryption schemes that allow an attacker to retrieve the plaintext once he knows the randomness of the ciphertext. Since we consider malleable zero-knowledge proofs, we cannot exclude that an attacker uses the randomness from some witness and a ciphertext to compute a proofs with the plaintext as a witness. Instead of granting the symbolic attacker this possibility, we require that every proof that carries a randomness terms $N$ of some (protocol) ciphertext $\mathrm{enc}(ek, m, N)$ also carries the corresponding plaintext $m$.In summary, a randomness term $r$ cannot be used for different terms unless the randomness belongs to a protocol ciphertext $\mathrm{enc}(ek, m, r)$. In this case, $r$ may additionally occur as a witness of a zero-knowledge proof if the same proof uses the corresponding plaintext $m$ as a witness.

Typically, a group operation underlies the re-randomization procedure, e.g., fresh randomness is simply added to the previous randomness. Naturally, the question arises whether the impossibility result of the XOR operation applies to our setting as well. If we include protocols that use attacker randomness or re-use for re-randomization it turns out that we run into similar problems as with XOR: then, the attacker is able to cause the protocol to construct two message that are computationally equal but symbolically distinct. Fortunately, for re-randomization we can assume that the protocol always uses fresh randomness and does not re-use this randomness anywhere else, except in the unveil information. With this protocol restriction, the attacker cannot have any meaningful influence to the randomness that is used in the re-randomization procedure; consequently, it cannot happen with more than negligible probability that the protocol constructs two re-randomizations that are symbolically distinct but computationally equal.

## 4.2 Implementation conditions

For the computational soundness result, we define necessary implementation conditions. The conditions can be partitioned into cryptographic requirements to cryptographic primitives and sanity conditions that ensure that the implementation behaves similar to the symbolic model.

**Sanity conditions.** We require that for each constructor and destructor $f \in \mathbf{C} \cup \mathbf{D}$, there is a polynomial-time computable, deterministic algorithm $A_f$ and the algorithm $A_N$ for drawing nonces is randomized. Moreover, for all algorithms the length of the output solely depends on the length of the input. Moreover, we require that all symbolic cancellation rules hold computationally as well, e.g., $\mathrm{fst}(\mathrm{pair}(x, y)) = x$ for all $x, y \in \{0,1\}^*$.

Finally, we assume that all messages have an efficiently recognizable type. Given a message, we require that it is efficiently possible to recognize whether the message is a pair, a signature, a ciphertext, a commitment, a

zero-knowledge proof, or a key, in particular which type of key.[8] More specifically, we require that $A_{\text{vkof}}(m) \neq \bot$, $A_{\text{ekof}}(m) \neq \bot$, and $A_{\text{crsof}}(m) \neq \bot$ for a message $m$ of type signature, ciphertext, and commitment, respectively. These conditions can be achieved by minor changes in the implementation.

**Encryptions and signatures are secure.** We require that the encryption algorithms $A_{\text{ek}}$, $A_{\text{dk}}$, $A_{\text{enc}}$, $A_{\text{dec}}$ constitute an IND-CCA secure encryption scheme. Moreover, we require that whenever $A_{\text{dec}}(\text{dk}_N, c)$ succeeds, $A_{\text{ekof}}(c) = \text{ek}_N$ outputs the corresponding public key, and that $A_{\text{ek}}(r) = A_{\text{ekofdk}}(A_{\text{dk}}(r))$. For the signature algorithms $A_{\text{sk}}$, $A_{\text{vk}}$, $A_{\text{sig}}$, $A_{\text{ver}_{\text{sig}}}$ we require that they constitute a CMA-existentially unforgeable signature scheme. We require that $A_{\text{sig}}$ produces different signatures for different randomnesses.

**Non-interactive zero-knowledge arguments of knowledge.** We require the following properties from zero-knowledge proofs: (*i*) completeness (honest provers and honest verifiers succeed for true statements); (*ii*) zero-knowledge (given a simulation trapdoor, all valid proofs can be efficiently simulated without using the witness); (*iii*) extractability (given an extraction trapdoor, the witness is efficiently extractable from valid proofs); (*iv*) unpredictability (fresh proofs cannot be guessed even if the witness is known); (*v*) length-regularity (the length of the output only depends on the length of the input); (*vi*) deterministic verification and extraction. The conditions (*i*) to (*iii*) are the minimal requirements on proofs of knowledge. Conditions (*iv*) to (*vi*) are properties that we need for our computational soundness proof and that are easy to fulfill. For the FMZK realization, we require that $A_{\text{ZK}}, A_{\text{ver}_{\text{zk}}}, A_{\text{crs}}, A_{\text{com}}$ constitute a non-interactive argument of knowledge (NIZKAoK), and the same commitment algorithms $A_{\text{crs}}, A_{\text{com}}, A_{\text{open}}$ belong to an extractable non-interactive computationally hiding and binding commitment scheme. We also assume an algorithm $A_{\text{getPub}}$ such that $A_{\text{getPub}}(A_{\text{ZK}}(t, r)) = r$ and we assume that the protocol transformations setPub, $\text{and}_{\text{ZK}}$, splitAnd, $\text{or}_{\text{ZK}}$, commute, $\text{rer}_{\text{zk}}$, $\text{rer}_{\text{com}}$ have an implementation. These conditions are compatible with the widely deployed Groth-Sahai proofs.

**Computational statements.** Each ZK-constructor represents one set of statements. Hence, we simple hardcode this statement computationally.

For the computational soundness proof and for the computational implementation, we need four destructors extrSta, extrWit, crsof, extrNon, which traverse through the statement and extract the statement, the witness, the CRS, and the list of nonces, respectively. The computational ZK relation $R_{\text{hon}}^{\text{comp}}$ for the protocol is then defined as follows:

$$\big\{((t, \text{img}_\eta(x)), \text{img}_\eta(w)) \mid \ t \text{ symb. statement } \wedge x = \text{extrSta}(t), w = \text{extrWit}(t))\big\}$$

**Controlled malleability.** As shown by Fuchsbauer [Fuc10, Lemma 6], it is possible to transform the witness inside a Groth-Sahai proofs. In a recent work, Chase, Kohlweiss, Lysyanskaya, and Meiklejohn, introduced a relaxation of simulation-sound extractability for controlled malleability: controlled-malleable simulation sound extractability [CKLM12], which states that an attacker can at most perform a fixed set of unary transformations on an honestly generated proof. We require that the zero-knowledge proof scheme satisfies this notion of controlled-malleability w.r.t. to the following two transformations: re-randomization of proofs and selectively hiding public information, e.g., used for existential quantification. In that work, additionally a generic construction for controlled-malleability out of non-interactive zero-knowledge arguments of knowledge is presented. There are two recent and practical examples that satisfy this notion of controlled malleability: the Groth-Sahai proof scheme [GS08], after the generic construction from [CKLM12] is applied to it, and the malleable SNARKs of Chase, Kohlweiss, Lysyanskaya, and Meiklejohn [Cha13].

We stress that we do not need to require that conjunctions are possible since simply sending two proofs proves the conjunctions of the two respective statements. For conjunctions it is more important to require that a proof about the logical conjunction $A \wedge B$ of two proofs about $A$ and $B$, respectively, should be indistinguishable from a fresh proof about $A \wedge B$. This additional requirement (derivation privacy) is described below.

[8] This requirement is similar in spirit as the recent, more general, result by Mödersheim and Katsoris [MK14] that characterizes which message formats can be soundly abstracted in Dolev-Yao model.

**Derivation privacy.** We require that a transformed proof is indistinguishable from a freshly produced proofs, in particular for selectively hiding public information or for constructing a proof about the logical conjunction of two given proofs. This property has been formalized as *derivation privacy* by Chase, Kohlweiss, Lysyanskaya, and Meiklejohn [CKLM12].

**Efficient statements.** In contrast to previous work on computationally symbolic ZK proofs [BBU13, BU08], the FMZK and the CMZK model have computationally sound realizations that allow efficient statements for encryption schemes and signature [GS08, CK11, CHK+11]. There are, in particular, encryption and signature schemes that are compatible with the malleable zero-knowledge proofs that satisfy our implementation conditions.

**MZK transformations.** For MZK transformations, we basically require that a transformed proof looks like a freshly generated proof. We even require that a transformed proof is indistinguishable from a simulated proof. This property, called *strong derivation privacy*, was introduced in [CKLM12]. The constructions of malleable zero-knowledge proofs in [CKLM12, Cha13] that satisfy our implementation conditions are also shown to satisfy this strong derivation privacy property.

Moreover, we require that $\mathrm{rer}_{\mathrm{zk}}$ and $\mathrm{rer}_{\mathrm{com}}$ have indeed re-randomizing implementations. Given $z$ the proof $z' \leftarrow A_{\mathrm{rer}_{\mathrm{zk}}}(z, r)$ is unpredictable, for a randomly chosen nonce $r$. Similarly, given $c$ the commitment $c' \leftarrow A_{\mathrm{rer}_{\mathrm{com}}}(c, r)$ is unpredictable, for a randomly chosen nonce $r$.

## 4.3 Computational soundness w.r.t. trace properties

In this section, we discuss the main challenges for proving computational soundness w.r.t. trace properties for the class of MZK-safe protocols.

The core of the simulator are the construction function $\beta$ and the parsing function $\tau$. The construction function $\beta$ maintains a memory. Whenever a bitstring $\beta(t)$ for a term $t$ is computed for the first time, $\beta(t)$ is stored; in all future calls $\beta(t)$ the stored bitstring is used. We recursively define $\beta$ over terms, e.g., $\beta(\mathrm{enc}(t_1, t_2, t_3)) := A_{\mathrm{enc}}(\beta(t_1), \beta(t_2), \beta(t_3))$.

**Reconstructing protocol transformations.** In the presence of zero-knowledge transformations, it can happen that the simulator parses (using $\tau$) a zero-knowledge proof from the attacker, sends it to the hybrid execution (i.e., the symbolic execution) which symbolically transforms this proof term, and then this transformed proof term is sent back to the simulator which has to construct the corresponding bitstring for the attacker. Then, the simulator needs to construct (using $\beta$) a proof, even though some witnesses are only known to the attacker. We let the simulator keep track of the transformations and apply these transformations computationally to the zero-knowledge proof that lead to this term. In order to be able to determine which zero-knowledge transformations have been applied to a term, we modify the hybrid execution with which the simulator interacts as follows: the result of a destructor node is in the yes-branch instead of $f(\underline{t})$ the term $\hat{f}(\underline{t})$, where for every destructor $f$, we introduce a free constructor $\hat{f}$. Moreover, instead of checking whether $f(\underline{t})$ holds, the hybrid execution first needs to get rid of the $\hat{f}$ constructors, by evaluating each $t_i$: $\mathrm{eval}(\hat{f}(\underline{s})) := f(\underline{\mathrm{eval}(s)})$ and $\mathrm{eval}(t) := t$ otherwise.

**Parsing re-randomized proofs.** The re-randomization operation is typically implemented using group operations and therefore prone to similar problems as computationally sound symbolic abstractions of XOR [BP05, Unr10]. At its core, the problem is that the attacker can potentially combine protocol-generated messages in a way that is not captured by the symbolic model. In particular, the attacker can find a particular combination of fresh nonces that equals another fresh nonce in the computational model, and, crucially, the attacker can in general cause a protocol-check that succeeds computationally and fails symbolically. In the symbolic model, however, fresh nonce are always distinct. Our protocol conditions for MZK-safe (see Section 4.1) ensure for protocol parties that the randomness used in a re-randomization operation is freshly chosen and stochastically independent of any other messages. As a consequence, the attacker cannot influence the randomness used in an honest re-randomization operation. We show that then the attacker cannot cause a protocol-check that succeeds computationally.

**Challenges in the construction of $\tau$ and $\beta$.** We only discuss the parts of the construction of $\tau$ and $\beta$ that are different from previous work since the other cases are conducted as in previous work [BHU09, BBU13]: Upon receiving a zero-knowledge proof bitstring $z$ that is different from the ones that have been sent by the protocol, the parsing function $\tau$ first symbolically checks whether the proof only contains information that the attacker could have derived on his own. If the check succeeds, $\tau$ outputs an purely attacker-generated proof term $t_z$ and we store the bitstring $z$ in the index of the attacker-randomness $N^z$ that we use for $t_z$. If the check fails, $\tau$ further checks symbolically whether the public information $p_z$ of $z$ coincides with the public information $p_{z'}$ of a protocol-generated proof $z'$ that has been earlier sent, up to secret information in the received proof that has potentially been hidden by the attacker and was public in the original proof. If such an earlier proof $z'$ exists, $\tau$ first applies all potentially necessary selective hiding transformations and then potentially a re-randomization transformation, where $\tau$ stores in the index of the symbolic attacker-randomness $N^z$ the bitstring $z$. Let $f$ denote this sequence of transformations. Then, $\tau$ applies this sequence of transformations $f$ to the term $t_{z'}$ that corresponds to the original proof $z'$ and outputs the resulting term $\mathrm{eval}(f(z'))$. We stress that $\tau$ does not need to find exactly the randomness bitstring that was used by the computational attacker since the re-randomization transformation contained in $f$ is purely symbolic.

For the converse direction, the construction function $\beta$ we are given a proof term $t$. For all honestly generated proofs $t$, we recursively evaluate $\beta$ on its subterms, e.g., $\beta(\mathsf{com3}(m, r_h, \bot)) = \mathsf{com3}(\beta(m), \beta(r_h), \bot)$. For proofs that are not purely protocol-generated, there is a unique bitstring $z$ received by the attacker that has been transformed. Moreover, recall that the transparent hybrid execution gives us all transformations $\hat{f}$ that have been applied to the proof by the protocol. If the proof term $t$ origins from a purely attacker-generated proof (bitstring) $z$ or proof (bitstring) $z$ that has been randomized by the attacker, the index of the symbolic attacker-randomness $N^z$ carries $z$. In this case, we apply the implementations of the transformations $f$ to $z$, i.e., $\beta = A_f(z)$ where $A_f$ denotes the implementations of the sequence of transformations.

If the proof term $t$ origins from a proof $z$, received from the attacker, that is in turn a transformed protocol-generated proof, with a corresponding term $t'$, to which only the selective hiding transformation $f'$ has been applied, i..e, $t = \mathrm{eval}f(f'(t'))$, we recursively construct $\beta(f(f'(t')))$.

**Dolev-Yaoness of Sim.** The Dolev-Yaoness of Sim is proven by constructing a faking simulator $\mathrm{Sim}_f$ that fakes all honest encryptions, i.e., $\mathrm{Sim}_f$ instead computes encryptions of the constant-zero bitstring, and simulates all proofs and handles all transformations are simulated proofs. In this way no plaintext is used while constructing encryptions, and no witness is used while constructing zero-knowledge proofs. Additionally, we go one step further and do not even use the protocol randomness in the computation of $\beta$: $\mathrm{Sim}_f$ uses an encryption faking oracle for constructing ciphertexts and a simulation oracle for constructing zero-knowledge proofs.

The proofs contains two parts that are significantly different to previous work: parsing re-randomized proofs and proving controlled malleability. As discussed above, we show that we parse re-randomized proofs in a computationally sound way by leveraging the protocol condition that honest re-randomizations always use fresh, and thus stochastically independent, randomness.

We show that that the attacker can only perform those transformations on zero-knowledge proofs that we symbolically model by a reduction to the controlled-malleable simulation sound extractability property (see Section 4.2).

**The indistinguishability of Sim.** We then show that Sim and $\mathrm{Sim}_f$ are indistinguishable. This follows from the cryptographic properties of the encryptions scheme, the commitment scheme, and the ZK proof, and it can be shown using standard techniques. Since $\mathrm{Sim}_f$ satisfies *Dolev-Yaoness* by construction, this entails that Sim does as well.

There are two aspects in the indistinguishability proof that are non-standard. First, our parsing function $\tau$ and our construction function $\beta$ potentially interpret transformed proofs as freshly generated proofs. Using derivation privacy [CKLM12], we show that a simulator Sim that uses $\beta$ and $\tau$ is indistinguishable from the original computational execution. Second, if the unveil information of a faked commitment (i.e., in the simulated proofs) is leaked, the indistinguishability breaks because the adversary can realize that the commitments are faked. At this point, however, we use the protocol condition that for each commitment the unveil information is either publicly known or secret for the entire execution.

**Theorem 4** (MZK-safe-computational soundness w.r.t. trace properties)**.** *Any computational implementation satisfying the aforementioned implementation conditions is computationally sound w.r.t. trace properties for the class of MZK-safe protocols.*

## 4.4 Computational soundness w.r.t. uniformity: self-monitoring

In this section, we discuss the distinguishing self-monitors for the symbolic model $\mathsf{M}$. Let $b \in \{\text{left}, \text{right}\}$ throughout this section and let, for a CoSP bi-protocol $\Pi$, $b(\Pi)$ be for the sake of exposition the protocol that is currently executed and $\overline{b}(\Pi)$ the complementary protocol, which is internally simulated. We construct a family of distinguishing self-monitors $f_{\mathsf{bad\text{-}branch},\Pi}(b, tr)$ for computation nodes, which we call *branching monitors*, and a family of distinguishing self-monitors $f_{\mathsf{bad\text{-}knowledge},\Pi}(b, tr)$ for output nodes, which we call *knowledge monitors*.

### 4.4.1 The branching monitor

We construct a distinguishing self-monitor $f_{\mathsf{bad\text{-}branch},\Pi}(b, tr)$, the branching monitor, for a computation node $\nu$ that investigates each message that has been received at an input node (in the execution trace $tr$ of $\mathsf{Mon}(\Pi)$) by parsing the message using computation nodes. The distinguishing self-monitor then reconstructs an attacker strategy by reconstructing a possible symbolic operation for every input message. In more detail, in the symbolic execution, $f_{\mathsf{bad\text{-}branch},\Pi}(b, tr)$ parses the input message by applying (the implementation of) all symbolic operations in the model $\mathsf{M}$ that the attacker could have performed as well, i.e., by applying all tests from the *shared knowledge*. Parsing the bitstrings of all primitives except for zero-knowledge proofs is standard and done as in previous work. Parsing the bitstrings of zero-knowledge proofs is conducted similar to the parsing function $\tau$ of the computational soundness simulator (see Section 4.3).

This enables $f_{\mathsf{bad\text{-}branch},\Pi}(b, tr)$ to simulate the symbolic execution of $\overline{b}(\Pi)$ on the constructed attacker strategy. In the computational execution of the self-monitor, the distinguishing self-monitor constructs the symbolic operations (i.e., the symbolic inputs) by parsing the input messages with the implementations of all tests in the shared knowledge (i.e., lookups on output messages and implementations of the destructors). With this reconstructed symbolic inputs (i.e., symbolic operations, from messages that were intended for $b(\Pi)$, $f_{\mathsf{bad\text{-}branch},\Pi}(b, tr)$ is able to simulate the symbolic execution of $\overline{b}(\Pi)$ even in the computational execution. The branching monitor $f_{\mathsf{bad\text{-}branch},\Pi}(b, tr)$ then checks whether this simulated symbolic execution of $\overline{b}(\Pi)$ takes in the same branch as $b(\Pi)$ would take, for the computation node $\nu$ in question. If this is not the case, the event $\mathsf{bad\text{-}branch}$ is raised.

**Symbolic self-monitoring.** Symbolic self-monitoring follows by construction because the branching monitor reconstructs a correct attacker strategy and correctly simulates a symbolic execution. We stress that a protocol is also able to change the attacker-randomness during a re-randomization operation. Since we only consider conjunctive statements, the symbolic attacker does not need to be able to construct the disjunction of two proofs, which is for all known ZK schemes not be possible for the protocol. All present attacker-operations are possible for a protocol as well. Hence, $f_{\mathsf{bad\text{-}branch},\Pi}(b, tr)$ can find any distinguishing attacker strategy for $b(\Pi)$ and $\overline{b}(\Pi)$. The proof from [BMR14a] applies verbatim.

**Lemma 1** (Symbolic self-monitoring of the branching monitor)**.** *Let $\Pi$ be a bi-protocol from the protocol class $\mathrm{P}$, and $\mathsf{M}$ be the symbolic model from Section A. The branching monitor satisfies symbolic self-monitoring (see Definition 13).*

**Computational self-monitoring.** We show computational self-monitoring by applying the CS result for trace properties to conclude that the symbolic simulation of $\overline{b}(\Pi)$ suffices to check whether $b(\Pi)$ computationally branches differently from $\overline{b}(\Pi)$. The main idea in the proof is that every branching that is different can be reduced to the violation of a trace property of the following protocol. The protocol implements guards after every input node such that only those messages are received that have also been received in the run in which the branching violation took place. Then, the protocol runs $b(\Pi)$ and thereafter $\overline{b}(\Pi)$, using the same guards. If the branching in the first and the second run differ, an alarm is raised, i.e., the protocol goes into a bad state. The proof goes along the lines of the proof in previous work [BMR14a].

**Lemma 2** (Computational self-monitoring of the branching monitor)**.** *The branching monitor satisfies computational self-monitoring (see Definition 13).*

### 4.4.2 The knowledge monitor

The distinguishing self-monitor $f_{\mathsf{bad\text{-}knowledge},\Pi}(b, tr)$, the knowledge monitor, for an output node $\nu$ starts like $f_{\mathsf{bad\text{-}branch},\Pi}(b, tr)$ by reconstructing a (symbolic) attacker strategy and simulating a symbolic execu-

tion of $\bar{b}(\Pi)$. However, instead of testing the branching behavior of $\bar{b}(\Pi)$, the distinguishing self-monitor $f_{\mathsf{bad\text{-}knowledge},\Pi}(b, tr)$ characterizes the message $m$ that is output in $b(\Pi)$ at the output node $\nu$ in question, and then $f_{\mathsf{bad\text{-}knowledge},\Pi}(b, tr)$ compares $m$ to the message that would be output in $\bar{b}(\Pi)$. This characterization must honor that ciphertexts generated by the protocol are indistinguishable if the corresponding decryption key has not been revealed to the attacker so far. If a difference in the output of $b(\Pi)$ and $\bar{b}(\Pi)$ is detected, the event bad-knowledge is raised.

**Symbolic self-monitoring.** Symbolic self-monitoring for the knowledge monitor $f_{\mathsf{bad\text{-}knowledge},\Pi}(b, tr)$ follows by the same arguments as for $f_{\mathsf{bad\text{-}branch},\Pi}(b, tr)$.

**Lemma 3** (Symbolic self-monitoring of the knowledge monitor)**.** *Let $\Pi$ be a bi-protocol. Let $i \in \mathbb{N}$. Let $\Pi'_i$ be the self-monitor for $\Pi_i$. For all $i \in \mathbb{N}$ the following holds. If there is an attacker strategy such that in $\Pi'_i$ the event* bad-knowledge *occurs but in $\Pi'_{i-1}$ the event* bad *does not occur and $\Pi_{i-1}$ is symbolically indistinguishable, then $\Pi_i$ is symbolically distinguishable because of knowledge.*

**Computational self-monitoring.** The core challenge in proving self-monitoring is the computational self-monitoring of the knowledge monitor, as at this point it has to be shown that all computationally distinguishing tests can be computationally reconstructed as a trace property by the knowledge monitor. The proof for computational self-monitor is along the lines of previous work [BMR14a]. First, we show that the simulator-indistinguishability w.r.t. trace properties (see Section 4.3) implies indistinguishability for the same simulators w.r.t. uniformity of bi-protocols. Then, we leverage the indistinguishability results to the scenario with the faking simulator $\mathrm{Sim}_f$ (see Section 4.3) from the computational soundness proof for trace properties: in the faking setting, all honestly generated ciphertexts generated by the protocol do not carry any information about their plaintexts, all secret commitments are faked, and all zero-knowledge proofs are faked (i.e., simulated). Second, we discuss in a case distinction all kinds of messages that remain unfaked. We show that in the faking setting, $f_{\mathsf{bad\text{-}knowledge},\Pi}(b, tr)$ is able to characterize all information that is information theoretically contained in a message. We conclude by showing that in the execution with the faking simulator, the knowledge monitor raises the event bad-knowledge whenever the bi-protocol $\Pi$ is distinguishable.

**Simulator-indistinguishability.** The first step, follows from Lemma 12 in [BMR14b]. In that lemma it is shown that if each protocol can be transformed to a so-called decision variant, which between any two nodes enables the attacker (in the proof the reduction) to store the distinguisher decision in the trace, then simulator-indistinguishability w.r.t. trace properties implies indistinguishability w.r.t. uniformity of bi-protocols. Since our protocol conditions for the class of MZK-safe protocols do not exclude such decision variants, Lemma 12 holds.

**Characterizing all messages in the faking setting.** Since the bitstrings contain no secret information[9] in the faking setting, we can show that applying the implementations of all relevant symbolic operations, which are basically all sequences destructor applications, yields a full characterization of the information that is contained in the bitstring.

**Lemma 4** (Computational self-monitoring of knowledge monitor)**.** *The parametric CoSP protocol $f_{\mathsf{bad\text{-}knowledge},\Pi}$ satisfies computational self-monitoring (see Definition 13).*

Plugging these results together it follows the symbolic model allows for self-monitoring. Using Theorem 3, we conclude that the symbolic model is computationally sound w.r.t. uniformity of bi-protocols.

**Theorem 5** (CS w.r.t. uniformity of bi-protocols)**.** *Let $A$ be an implementation that satisfies the conditions from Section 4.2. Then, $A$ is computationally sound w.r.t. uniformity of CoSP bi-protocols for the class of MZK-safe CoSP bi-protocols.*

# 5 Soundness of the verification

In section 2.4, we have shown that the symbolic abstraction of the simplified AWoT protocol provides strong anonymity, i.e., the processes $\mathcal{M}_1^{\mathsf{Anon}}$ and $\mathcal{M}_2^{\mathsf{Anon}}$ are observational equivalent in the applied $\Pi$ calculus. More-

[9] Technically, the bitstrings have been constructed without using secret information.

over, we have shown in section 4 that the symbolic model that also contains re-randomization is computationally sound. Combined we can conclude the following result for an implementation:

**Theorem 6** (Anonymity of the case study)**.** *The computational implementations of the two processes* $\mathcal{M}_1^{\mathsf{Anon}}$ *and* $\mathcal{M}_2^{\mathsf{Anon}}$ *are tic-indistinguishable* $\mathcal{M}_1^{\mathsf{Anon}} \approx_{tic} \mathcal{M}_2^{\mathsf{Anon}}$*.*

*Proof.* We have the soundness result for the symbolic model, cf., Theorem 5. Consequently, for the symbolic model used in the case study, section 2.4, it follows that two processes are symbolically indistinguishable if and only if they are computationally indistinguishable.

By Theorem 2 we have that $\mathcal{M}_1^{\mathsf{Anon}}$ and $\mathcal{M}_2^{\mathsf{Anon}}$ are observational equivalent which concludes the theorem. Combining these two facts, we get that the two processes are computationally indistinguishable, i.e., tic-indistinguishable. □

# 6 Conclusion and Future Work

We have presented a computationally sound, symbolic abstraction of malleable ZK proofs by means of an equational theory that is accessible to existing tools for automated verification of security protocols. We have proved the computational soundness of our abstraction with respect to trace properties and using weak cryptographic assumptions that are compatible with the widely deployed Groth-Sahai proof scheme. The abstraction and the computational soundness result are presented in CoSP, a framework for symbolic protocol analyses and conceptually modular computational soundness proofs.

# References


[BAF08] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated Verification of Selected Equivalences for Security Protocols. *Journal of Logic and Algebraic Programming*, 75:3–51, 2008.

[BBFR15] Michael Backes, Manuel Barbosa, Dario Fiore, and Raphael M. Reischuk. ADSNARK: Nearly Practical and Privacy-Preserving Proofs on Authenticated Data. In *Proc. 36th IEEE Symposium on Security & Privacy (S&P)*, page to appear. IEEE Computer Society Press, 2015.

[BBU13] Michael Backes, Fabian Bendun, and Dominique Unruh. Computational soundness of symbolic zero-knowledge proofs: Weaker assumptions and mechanized verification. In *POST'13*, pages 206–225, 2013.

[BCC+09] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In *CRYPTO'09*, pages 108–125. Springer, 2009.

[BHM08] Michael Backes, Cătălin Hriţcu, and Matteo Maffei. Type-checking zero-knowledge. In *Proc. 15th ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 357–370. ACM Press, 2008.

[BHM12] Michael Backes, Cătălin Hriţcu, and Matteo Maffei. Union and intersection types for secure protocol implementations. In *Proc. 2011 International Conference on Theory of Security and Applications*, TOSCA'11, pages 1–28. Springer-Verlag, 2012.

[BHU09] Michael Backes, Dennis Hofheinz, and Dominique Unruh. Cosp: A general framework for computational soundness proofs. In *Proc. 16th ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 66–78, November 2009.

[BL13] Foteini Baldimtsi and Anna Lysyanskaya. Anonymous Credentials Light. In *Proc. 20th ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1087–1098. ACM Press, 2013.

[BLMP10] Michael Backes, Stefan Lorenz, Matteo Maffei, and Kim 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.

[BMM10] Michael Backes, Matteo Maffei, and Esfandiar Mohammadi. Computationally Sound Abstraction and Verification of Secure Multi-Party Computations. In *Proc. 30th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 352–363. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

[BMP12] Michael Backes, Matteo Maffei, and Kim Pecina. Automated Synthesis of Privacy-Preserving Distributed Applications. In *NDSS'12*. Internet Society, 2012.

[BMR14a] Michael Backes, Esfandiar Mohammadi, and Tim Ruffing. Computational Soundness Results for ProVerif. In *Proceedings of the 3rd Conference on Principles of Security and Trust (POST)*, pages 42–62. Springer, 2014.

[BMR14b] Michael Backes, Esfandiar Mohammadi, and Tim Ruffing. Full Version:Computational Soundness Results for ProVerif.

`http://www.infsec.cs.uni-saarland.de/~mohammadi/paper/bridge.pdf`, 2014.

[BMU08] Michael Backes, Matteo Maffei, and Dominique Unruh. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In *S&P'08*, pages 158–169, 2008.

[BMU10] Michael Backes, Matteo Maffei, and Dominique Unruh. Computationally sound verification of source code. In *Proc. 17th ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 387–398. ACM Press, October 2010.

[BP05] Michael Backes and Birgit Pfitzmann. Limits of the Cryptographic Realization of Dolev-Yao-Style XOR. In *Proc. 10th European Symposium on Research in Computer Security (ESORICS)*, volume 3679 of *LNCS*, pages 178–196. Springer Berlin Heidelberg, 2005.

[BU08] Michael Backes and Dominique Unruh. Computational soundness of symbolic zero-knowledge proofs against active attackers. In *Proc. 21th IEEE Computer Security Foundations Symposium (CSF)*, 2008.

[cas] Case Study: Verifying AWoT in ProVerif.

`http://malleable-zk.bplaced.net/awot.zip`.

[CCM08] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a Secure Voting System. In *Proc. 29th IEEE Symposium on Security & Privacy (S&P)*, pages 354–368. IEEE Computer Society Press, 2008.

[Cha13] Chase, Melissa and Kohlweiss, Markulf and Lysyanskaya, Anna and Meiklejohn, Sarah. Succinct malleable nizks and an application to compact shuffles. In *Proc. 10th Theory of Cryptography Conference (TCC)*, volume 7785 of *LNCS*, pages 100–119. Springer-Verlag, 2013.

[CHK+11] Jan Camenisch, Kristiyan Haralambiev, Markulf Kohlweiss, Jorn Lapon, and Vincent Naessens. Structure preserving cca secure encryption and applications. In *Proc. of the 17th international conference on The Theory and Application of Cryptology and Information Security (ASIACRYPT'11)*, ASIACRYPT'11, pages 89–106. Springer-Verlag, 2011.

[CK11] Melissa Chase and Markulf Kohlweiss. A domain transformation for structure-preserving signatures on group elements. Cryptology ePrint Archive, Report 2011/342, 2011.

[CKLM12] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable proof systems and applications. In *EUROCRYPT'12*, volume 7237 of *LNCS*, pages 281–300. Springer, 2012.

[DY83] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IT'83*, 29(2):198–208, 1983.

[EG83] Shimon Even and Oded Goldreich. On the security of multi-party ping-pong protocols. In *FOCS'83*, pages 34–39, 1983.

[FP09] Georg Fuchsbauer and David Pointcheval. Proofs on encrypted values in bilinear groups and an application to anonymity of signatures. In *Pairing'09*, pages 132–149. Springer, 2009.

[Fuc10] Georg Fuchsbauer. *Automorphic Signatures and Applications.* PhD thesis, École normale supérieure, Paris, 2010.

[ful] Full version: Symbolic Malleable Zero-knowledge Proofs.

`http://malleable-zk.bplaced.net/malleable-zk.pdf`.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–207, 1989.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT'08*, pages 415–432. Springer-Verlag, 2008.

[Mer83] Michael Merritt. *Cryptographic Protocols.* PhD thesis, Georgia Institute of Technology, 1983.

[MK14] Sebastian Mödersheim and Georgios Katsoris. A Sound Abstraction of the Parsing Problem. In *Proc. 27th IEEE Computer Security Foundations Symposium (CSF)*, pages 259–273. IEEE Computer Society Press, 2014.

[MP11] Matteo Maffei and Kim Pecina. Position paper: Privacy-aware proof-carrying authorization. In *PLAS 2011*, 2011. To appear.

[MPR13] Matteo Maffei, Kim Pecina, and Manuel Reinert. Security and privacy by declarative design. In *Proc. 26th IEEE Computer Security Foundations Symposium (CSF)*. IEEE Computer Society, 2013.

[MSC+] Simon Meier, Benedikt Schmidt, Cas Cremers, Cedric Staub, Ralf Sasse, and David Basin. Accessed on 19 April 2015 at `http://www.infsec.ethz.ch/research/software/tamarin.html`.

[PHGR13] B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly Practical Verifiable Computation. In *Proc. 34th IEEE Symposium on Security & Privacy (S&P)*, pages 238–252. IEEE Computer Society Press, 2013.

[Ram29] F. Ramsey. On a Problem of Formal Logic. *Proceedings of the London Mathematical Society*, 30:338–384, 1929.

[SSCB14] Benedict Schmidt, Ralf Sasse, Cas Cremers, and David Basin. Automated Verification of Group Key Agreement Protocols. In *Proc. 35th IEEE Symposium on Security & Privacy (S&P)*, pages 179–194. IEEE Computer Society Press, 2014.

[Unr10] Dominique Unruh. The Impossibility of Computationally Sound XOR. `http://eprint.iacr.org/2010/389`, 2010.

[Unr11] Dominique Unruh. Termination-Insensitive Computational Indistinguishability (and Applications to Computational Soundness). In *Proc. 24th IEEE Computer Security Foundations Symposium (CSF)*, pages 251–265. IEEE Computer Society Press, 2011.

# Appendix



## A Symbolic abstraction of Malleable ZK Proofs

This section presents our symbolic abstraction for malleable zero-knowledge (MZK) proofs by means of an equational theory. We introduce two variations of our symbolic model: *fully-malleable zero-knowledge proofs* (the FMZK variation) and *controlled-malleable zero-knowledge proofs* (the CMZK variation). The two variations differ only in their degree of malleability: the FMZK variation also permits functional transformations, whereas the CMZK variation excludes this case and is therefore better suited for automated verification tools.

For the sake of presentation, we first present a standard symbolic model for digital signatures and public-key encryptions, which closely resembles previous work [BBU13, BHU09]. Thereafter, we extend this symbolic abstraction with MZK proofs. Our symbolic abstraction constitutes a symbolic model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$. In what follows, we write $\underline{x}$ for a tuple $(x_i)_{i=1}^n$.

### A.1 The basic symbolic model

**Terms.** In the basic symbolic model, messages are modeled as *basic terms* $\mathbf{T}'$; the grammar for this set is presented below. Nonces are modeled as distinct terms from a countably infinite set $\mathbf{N} \subset \mathbf{T}'$, with two disjoint subsets for protocol nonces $\mathbf{N}_P \subset \mathbf{N}$ and attacker nonces $\mathbf{N}_E \subset \mathbf{N}$. Let $N, N' \in \mathbf{N}$ and $M, M' \in \mathbf{T}'$. Then, pairs are represented as $\mathrm{pair}(M, M')$, an encryption key as $\mathrm{ek}(N)$, a decryption key as $\mathrm{dk}(N)$, the encryption of a message $M$ with the key $\mathrm{ek}(N)$ and randomness $N'$ as $\mathrm{enc}(\mathrm{ek}(N), M, N')$, a signature key as $\mathrm{sk}(N)$, a verification key as $\mathrm{vk}(N)$, the signature of a message $M$ with the key $\mathrm{sk}(N)$ and randomness $N'$ as $\mathrm{sig}(\mathrm{sk}(N), M, N')$. Moreover, we also incorporate into the symbolic model payloads, expressed by the three constructors $\varepsilon$, $\mathrm{string}_0$ and $\mathrm{string}_1$. We define the set $\mathbf{C}' := \{\mathrm{enc}, \mathrm{ek}, \mathrm{dk}, \mathrm{sig}, \mathrm{vk}, \mathrm{sk}, \mathrm{pair}, \varepsilon, \mathrm{string}_0, \mathrm{string}_1\}$ of *basic constructors*. The full grammar of terms, which subsumes the grammar of basic terms, is depicted in Figure 7.

**Explicitly modeling randomness.** Cryptographic terms such as encryptions and signatures include an explicit randomness term in our symbolic model. Beside making the model more accurate, such randomness terms allow modeling MZK proofs about ciphertexts. Moreover, this randomness allows MZK proofs that prove ownership of a ciphertext.

**Destructors.** Operations on messages are modeled as partial functions $f : \mathbf{T}'^n \to \mathbf{T}'$. Whenever such a function $f$, called destructor, is undefined on some $\underline{t}$ we write $f(\underline{t}) = \bot$. As an example, consider equality, which is modeled as the 2-ary destructor $\mathrm{equals} := \{((x, x), x) \mid x \in \mathbf{T}'\}$. This set $\mathbf{D}'$ of *basic destructors* is defined in Figure 5.

$$\begin{array}{rlrl}
\mathrm{dec}(\mathrm{dk}(t_1), \mathrm{enc}(\mathrm{ek}(t_1), m, t_2)) & = m & \mathrm{fst}(\mathrm{pair}(t_1, t_2))) & = t_1 \\
\mathrm{ver}_{\mathrm{sig}}(\mathrm{vk}(t_1), \mathrm{sig}(\mathrm{sk}(t_1), t_2, t_3)) & = t_2 & \mathrm{snd}(\mathrm{pair}(t_1, t_2)) & = t_2 \\
\mathrm{ekof}(\mathrm{enc}(\mathrm{ek}(t_1), t_2, t_3)) & = \mathrm{ek}(t_1) & \mathrm{unstring}_1(\mathrm{string}_1(s)) & = s \\
\mathrm{vkof}(\mathrm{sig}(\mathrm{sk}(t_1), t_2, t_3)) & = \mathrm{vk}(t_1) & \mathrm{unstring}_0(\mathrm{string}_0(s)) & = s \\
\mathrm{ekofdk}(\mathrm{dk}(t)) & = \mathrm{ek}(t) & \mathrm{equals}(x, x) & = x
\end{array}$$

Figure 5: Definition of the set $\mathbf{D}'$ of basic destructors

$$\frac{m \in S}{S \vdash m} \qquad \frac{S \vdash \bar{t} \qquad \bar{t} \in \mathbf{T} \qquad F \in \mathbf{C}' \cup \mathbf{D}' \qquad \mathrm{eval}_F(\bar{t}) \neq \bot}{S \vdash \mathrm{eval}_F(\bar{t})} \qquad \frac{N \in \mathbf{N}_E}{S \vdash N}$$

Figure 6: Knowledge relation for the basic model

**Symbolic attacker in the basic model.** The main advantage of the symbolic model is that the attacker is restricted to a small set of well-defined actions. These actions are formalized by a deduction relation $\vdash$: $2^{\mathbf{T}'} \times \mathbf{T}'$, called the *knowledge relation*, where $S \vdash t$ characterizes the messages $t$ that the attacker can compute given some knowledge $S \subseteq \mathbf{T}'$. This knowledge relation is canonically defined over the set of basic constructors $\mathbf{C}'$ and destructors $\mathbf{D}'$ (see Figure 6). For a knowledge set $S \subset \mathbf{T}'$ the expression $S \vdash m$ denotes that the attacker can compute $m$ out of his knowledge $S$.

**Sort destructors & garbage terms.** Because of space constraints, we omitted two kinds of terms and destructors in the presentation of the basic symbolic model. First, in our complete basic symbolic model (see Figure 9 in the appendix)we include for every constructor $f$ a destructor isF, defined as $\mathrm{isF}(f(x)) = f(x)$. Second, since we require (potentially forgeable) message types for each cryptographic primitive, we have to model ill-typed messages, such as signatures that do not pass verification. In order to achieve computational soundness, we model these wrongly-typed messages in our symbolic model via so-called *garbage constructors*.

## A.2 Symbolic MZK proofs

We first present the syntax of our model. Thereafter, we discuss how we define the validity of symbolic MZK proofs, in particular how we prevent an MZK proof of a disjunction $A \vee B$ from leaking the validity $A$ or $B$ alone. Then, we introduce a finite set of destructors that characterizes all possible transformations that a cryptographic attacker (against trace properties) can perform. Finally, we describe the symbolic attacker.

### A.2.1 Terms and statements

A MZK proof is represented symbolically as a term $\mathrm{ZK}(s, r, N)$, where ZK is a constructor, $s$ is the statement, $r$ is auxiliary verification information, and $N$ is a randomness symbol. A statement is basically a propositional logic formula over equations of terms. In order to give a finite characterization of all possible transformations of MZK proofs in our setting, we explicitly model cryptographic commitments that are typically used in a cryptographic realization of MZK proofs. A symbolic commitment on a term $t$ with randomness $r'$ and the CRS[10] $\mathrm{crs}(r)$ is represented as $\mathrm{com}(\mathrm{crs}(r), t, r')$. For a commitment $\mathrm{com}(\mathrm{crs}(r), t, r')$, the corresponding unveiling information is modeled as $\mathrm{uv}(t, r')$, and a destructor open such that $\mathrm{open}(\mathrm{com}(\mathrm{crs}(r), t, r'), \mathrm{uv}(t, r')) = t$. A witnesses $w$ is represented as a commitment $\mathrm{com}(\mathrm{crs}(r), w, r')$ in the statement. Public information is also represented as a commitment $\mathrm{com}(\mathrm{crs}(r), m, r')$; however, in contrast to witnesses, public information in the statement also carries its unveiling information $\mathrm{uv}(m, r')$, hence it is possible to retrieve $m$.

Analogous to common cryptographic MZK proof schemes (e.g., the Groth-Sahai scheme [GS08]), we represent public information as commitments together with unveiling information.

**Statements.** A symbolic statement is the encoding of a propositional logic formula over atomic statements. Atomic statements are equalities and negated equalities of terms.

[10]The common reference string (CRS) is a central assumption in the construction of many commitment schemes that are used in zero-knowledge proofs.

$$
\begin{aligned}
T &::= \mathrm{enc}(\mathrm{ek}(N), T, N) \mid \mathrm{ek}(N) \mid \mathrm{dk}(N) \mid \mathrm{pair}(T, T) \mid \\
&\quad \mathrm{sig}(\mathrm{sk}(N), T, N) \mid \mathrm{vk}(N) \mid \mathrm{sk}(N) \mid ST \mid N \mid \mathrm{crs}(N) \\
&\quad \mathrm{ZK}(ST, R, RS) \mid \mathrm{com}(\mathrm{crs}(N), T, RS) \mid \mathrm{uv}(T, RS) \\
Q &::= \varepsilon \mid \mathrm{string}_0(Q) \mid \mathrm{string}_1(Q) \\
S &::= S \mid O(S)
\end{aligned}
$$

$$
\begin{aligned}
ST &::= ST \wedge ST \mid ST \vee ST \mid TX = TX \mid TX \neq TX \\
TX &::= \langle \mathrm{com}(\mathrm{crs}(n), TY, RS), U \rangle \mid F\langle \underbrace{TX, \ldots, TX}_{m-times} \rangle \\
R &::= R \wedge R \mid R \vee R \mid RX = RX \mid RX \neq RX \\
RX &::= RS \mid F\langle \underbrace{RX, \ldots, RX}_{m-times} \rangle \qquad U ::= \varepsilon \mid \mathrm{uv}(T, RS) \\
RS &::= (M, N) \mid \mathrm{andRand}(RS, RS) \\
M &::= N \mid \mathrm{rand}(M, N) \qquad N ::= n
\end{aligned}
$$

where $F \in \mathrm{StF}$, $n \in \mathbf{N}$ and $m$ is the arity of $F$.

Figure 7: The syntax of terms

We represent statements as encodings of contexts with commitments instead of holes. The tags AND, OR, COM, EQUALS, NEQUALS are encoded using $\mathrm{string}_1$, $\mathrm{string}_0$, $\varepsilon$. We introduce for every constructor and destructor $F \in \mathrm{StF} := \{\mathrm{ver}_{\mathrm{sig}}, \mathrm{fst}, \mathrm{snd}, \mathrm{enc}, \mathrm{pair}, \mathrm{sig}\}$ a tag $\hat{F}$. We use the following abbreviations, where for $A_1, \ldots, A_n, A, B$ we write $\langle A_1, \ldots, A_n \rangle$ for $\mathrm{pair}(A_1, \ldots \mathrm{pair}(A_{n-1}, A_n) \ldots)$, $A \wedge B$ for $\langle \mathrm{AND}, A, B \rangle$, $A \vee B$ for $\langle \mathrm{OR}, A, B \rangle$, $A = B$ for $\langle \mathrm{EQUALS}, A, B \rangle$, $A \neq B$ for $\langle \mathrm{NEQUALS}, A, B \rangle$, and $F\langle A_1, \ldots, A_n \rangle$ for $\langle \hat{F}, A_1, \ldots, A_n \rangle$, where $F \in \mathrm{StF}$.

**Terms.** In addition to the basic symbolic model from the previous section, the set $\mathbf{T}$ of terms contains commitments and MZK proofs. MZK proofs are modeled as terms $\mathrm{ZK}(t, r, N)$ that consist of three parts: a statement $t$ as described above, a so-called *randomness tree* $r$, which is basically a commitment to the commitments in a statement (see Section A.2.2), and a pair of randomness nonces $(M, N)$, where $M$ is the protocol randomness and $N$ is the attacker-randomness.We need the distinction between protocol randomness and attacker-randomness in order to – on the one hand – avoid known impossibility results for computational soundness and – on the other hand – make the automated verification feasible. Below, we elaborate more on our choice of maintaining two kinds of randomness symbols.

Formally, the set of *terms* $\mathbf{T}$ is generated by the grammar from Figure 7 for the non-terminal $T$. The set of *constructors* is defined as $\mathbf{C} := \mathbf{C}' \cup \{\mathrm{ZK}, \mathrm{com}, \mathrm{crs}, \mathrm{uv}, \mathrm{rand}\}$.

**The distinction between protocol and attacker randomness.** Computationally, re-randomization typically works by adding (or multiplying, depending on the group structure) a uniformly random value $r$ to some value $v$ and, using a one-time pad argument, the resulting value is uniformly random and independent of $v$. Although appealing, modeling such algebraic properties symbolically is inherently unsound [Unr10].

We circumvent this issue by requiring honest protocol participants to always use freshly-chosen random values for the re-randomization process: since the randomness is always fresh and chosen uniformly at random, algebraic properties do not play a role.

At the same time, the model must consider that the attacker cannot be restricted in any way. A first approach is to let re-randomization replace randomness in a zero-knowledge triple consistently. Since the attacker can re-randomize a proof and commitments can be opened once the randomness is known, such an approach cannot provide any privacy properties. Another idea is to symbolically combine the randomness. While functional, this approach effectively leads to non-termination in the verification due to the unbounded number of possible combinations that the attacker can derive.

We solve the two aforementioned problems by introducing for every term that contains randomness two different random values: one that can only be modified by honest protocol participants and one that can

be arbitrarily modified by the attacker. If a protocol participant applies a re-randomization operation, we distinguish two cases: if the corresponding opening information is available, we replace the existing honest randomness; otherwise, we combine the existing honest randomness and the new randomness using the constructor rand. Since a protocol only comprises a finite number of re-randomization operations, the number of nested rand constructors is bounded, making automated verification viable. If the attacker applies re-randomization, we replace the corresponding attacker randomness.

This unusual treatment of the honest randomness is necessary to obtain computational soundness while still being able to connect the malleable proofs [MPR13].

At first, this over-approximation seems to let the attacker selectively replace the randomness and, consequently, learn the committed values used in a proof. The attacker, however, cannot touch the honest randomness. Consequently, the attacker can only know the complete randomness (honest and attacker randomness) of a commitment, if the randomness was revealed in the first place.

*Example 1: Anonymous webs of trust.* We will illustrate our approach on the anonymous webs of trust protocol [BLMP10], which can be seen as a form of anonymous delegatable credentials [BCC$^+$09] (also known as anonymous proxy signatures [FP09]). In webs of trust, a party $A$ shows that it trusts a party $B$ by signing the verification key of $B$ ($\mathrm{sig}(\mathrm{sk}_A, \mathrm{vk}_B, r)$). A chain of trust from $A$ to $B$ via $C$ can be expressed as follows: $\mathrm{sig}(\mathrm{sk}_C, \mathrm{vk}_B, r_1), \mathrm{sig}(\mathrm{sk}_A, \mathrm{vk}_C, r_2)$.

Anonymous webs of trust additionally hide the identity of all parties except from the party to whom the message is sent, i.e., the statement of a proof that there is an anonymous chain of trust to $A$ looks as follows:

$$\exists s_C, s_A, v_B, v_C.\mathrm{ver}_{\mathrm{sig}}(v_C, s_C) = v_B \wedge \mathrm{ver}_{\mathrm{sig}}(\mathrm{vk}_A, s_A) = v_C$$

This statement corresponds to the following term:

$$\begin{aligned}&\hat{\mathrm{ver}}_{\mathrm{sig}}\big\langle\langle c_{v_C}, \varepsilon\rangle, \langle c_{s_C}, \varepsilon\rangle\big\rangle = \langle c_{v_B}, \varepsilon\rangle\\ &\wedge \hat{\mathrm{ver}}_{\mathrm{sig}}\big\langle\langle c_{\mathrm{vk}_A}, \mathrm{uv}(\mathrm{vk}_A, r_{c_{\mathrm{vk}_A}})\rangle, \langle c_{s_A}, \varepsilon\rangle\big\rangle = \langle c_{v_C}, \varepsilon\rangle\end{aligned}$$

where $c_t$ denotes the commitment to the message $t$. Such an anonymous chain of trust can be used by $B$ to anonymously authenticate a message $m$ with the trusted verification key $v_B$. Assuming $B$ uses an anonymous channel (such as Tor), then $B$ anonymously sends the authenticated message $m$ to $A$. The authentication is achieved with an MZK proof for the following statement:

$$\begin{aligned}&\exists s_C, s_B, s_A, v_B, v_C.\ \mathrm{ver}_{\mathrm{sig}}(v_B, s_B) = m\\ &\wedge \mathrm{ver}_{\mathrm{sig}}(v_C, s_C) = v_B \wedge \mathrm{ver}_{\mathrm{sig}}(\mathrm{vk}_A, s_A) = v_C\end{aligned}$$

Let $S_{awot}$ be the corresponding term. ◇

### A.2.2 Destructors for MZK Proofs

We consider three kinds of destructors for MZK proofs. First, our model allows for retrieving the public part of a MZK proof (via getPub), which outputs the statement of a ZK proof. Second, our model allows for checking the validity of an MZK proof (via $\mathrm{ver}_{\mathrm{zk}}$), which checks the validity by calling the function $\mathrm{check}_{\mathrm{zk}}$ (see below) on the CRS, the statement and the randomness tree. Third, our model allows for transforming MZK proofs (via setPub, $\mathrm{and}_{\mathrm{ZK}}$, splitAnd, $\mathrm{or}_{\mathrm{ZK}}$, commute, $\mathrm{rer}_{\mathrm{zk}}$). The set of *destructors* is defined as $\mathbf{D} := \mathbf{D}' \cup \mathbf{D}''$, where $\mathbf{D}'$ are the destructors from the basic symbolic model, and $\mathbf{D}''$ are the destructors presented in Figure 8. Below, we describe these MZK transformations.

**Validity of symbolic proofs: $\mathbf{check_{zk}}$.** A statement contains all information that a zero-knowledge proof carries. However, for dealing with equivalence properties, we need to capture the zero-knowledge property. As an example, consider a proof $\mathrm{ZK}(A \vee B, r, N)$ of a disjunctive statement $A \vee B$. We have to ensure that $\mathrm{ZK}(A \vee B, r, N)$ hides which branch ($A$ or $B$) is valid. Even though the attacker should be able to retrieve the full statement $A \vee B$ from a proof $\mathrm{ZK}(A \vee B, r, N)$, the attacker should not be able to learn whether $A$ or whether $B$ is true. In particular, the verification procedure should fail if the attacker tries to verify $A$ or $B$ outside of the original proof $\mathrm{ZK}(A \vee B, r, N)$.

We achieve the zero-knowledge property of a disjunctive proof $\mathrm{ZK}(A \vee B, r, N)$ by keeping track (in the auxiliary verification information $r$) of the randomness of the commitments, loosely speaking by adding a commitment to the commitments. This makes the auxiliary verification information $r$ unique for $A \vee B$ and

$$
\begin{aligned}
\mathrm{ver}_{\mathrm{zk}}(\mathrm{crs}, \mathrm{ZK}(t, r, N)) &= \mathrm{check}_{\mathrm{zk}}(\mathrm{crs}, t, r) \\
\mathrm{getPub}(\mathrm{ZK}(t_1, r_1, N)) &= t_1 \\
\mathrm{setPub}(\mathrm{ZK}(t_1, r_1, N), t_2, N') &= \mathrm{ZK}(t_2, r_1, \mathrm{rand}(N, N')) \\
\mathrm{setPubA}(\mathrm{ZK}(t_1, r_1, N), t_2) &= \mathrm{ZK}(t_2, r_1, N) \\
\mathrm{and}_{\mathrm{ZK}}(\mathrm{ZK}(t_1, r_1, N), \mathrm{ZK}(t_2, r_2, N')) &= \mathrm{ZK}(t_1 \wedge t_2, r_1 \wedge r_2, \mathrm{andRand}(N, N')) \\
\mathrm{splitAnd}(\mathrm{ZK}(t_1 \wedge t_2, r_1 \wedge r_2, \mathrm{andRand}(N, N'))) &= \mathrm{pair}(\mathrm{ZK}(t_1, r_1, N), \mathrm{ZK}(t_2, r_2, N')) \\
\mathrm{or}_{\mathrm{ZK}}(\mathrm{ZK}(t_1, r_1, N), \mathrm{ZK}(t_2, r_2, N')) &= \mathrm{ZK}(t_1 \vee t_2, r_1 \vee r_2, \mathrm{rand}(N, N')) \\
\mathrm{commute}(\mathrm{ZK}(t_1 \vee t_2, r_1 \vee r_2, N)) &= \mathrm{ZK}(t_2 \vee t_1, r_2 \vee r_1, N) \\
\mathrm{rer}_{\mathrm{zk}}(\mathrm{ZK}(t_1, r_1, (M, N)), r_2, N', M) &= \mathrm{ZK}(\mathrm{rerPr}(t_1, r_2), \mathrm{rerPr}(r_1, r_2), \\
&\qquad (N', N)) \\
\mathrm{rer}_{\mathrm{zk}}(\mathrm{ZK}(t_1, r_1, (M, N)), r_2, N', \varepsilon) &= \mathrm{ZK}(\mathrm{rerPr}(t_1, r_2), \mathrm{rerPr}(r_1, r_2), \\
&\qquad (\mathrm{rand}(M, N'), N)) \\
\mathrm{attRer}_{\mathrm{zk}}(\mathrm{ZK}(t_1, r_1, (M, N)), r_2, N') &= \mathrm{ZK}(\mathrm{rerPr}(t_1, r_2), \mathrm{rerPr}(r_1, r_2), \\
&\qquad (M, N')) \\
\mathrm{open}(\mathrm{crs}(t_1), \mathrm{com}(\mathrm{crs}(t_1), t_2, t_3), \mathrm{uv}(t_2, t_3)) &= t_2 \\
\mathrm{crsof}(\mathrm{com}(crs(t_1), t_2, t_3)) &= \mathrm{crs}(t_1) \\
\mathrm{rer}_{\mathrm{com}}(\mathrm{com}(\mathrm{crs}(t_1), t_2, t_3), t_4) &= \mathrm{com}(\mathrm{crs}(t_1), t_2, \mathrm{rand}(t_3, t_4)) \\
\mathrm{applyF}(\mathrm{com}(\mathrm{crs}(t_1), t_2, t_3), t_4) &= \mathrm{com}(\mathrm{crs}(t_1), f(t_2, t_4), t_3) \text{ for } f \in \mathbf{D} \cup \mathbf{C}
\end{aligned}
$$

Figure 8: The set $\mathbf{D}''$ of destructors for ZK proofs for the FMZK model. The CMZK model does not contain $\mathrm{or}_{\mathrm{ZK}}$, applyF, and setPub can only be used to hide witnesses.

unguessable for anybody who did not create the proof. Technically, the term $r$ has exactly the same shape as the statement except for the equalities and negated equalities over pairs of commitments and possible unveiling information terms.

This auxiliary verification information $r$ differs in the FMZK model and the CMZK model. In the FMZK model, $r$ only contains the randomness of the commitments, i.e., we only commit to the randomness of the commitments. Hence, $\mathrm{check}_{\mathrm{zk}}$ only checks whether the randomness in the auxiliary verification information $r$ coincides with the randomness of the commitments. In the CMZK model, $r$ contains the messages and the randomness of the commitments, i.e., we commit to the messages and to the randomness of the commitments. Hence, $\mathrm{check}_{\mathrm{zk}}$ checks whether the messages and the randomness in the auxiliary verification information $r$ coincides with the messages and randomness of the commitments. Adding the messages to the auxiliary information and matching them with the witness in the proof suffices to prevent functional transformations.

We say that a symbolic proof is *valid* if the following partial function $\mathrm{check}_{\mathrm{zk}}$ outputs true[11]. $\mathrm{check}_{\mathrm{zk}}$ is recursively defined as depicted below. (The verification procedure additionally checks whether all commitments use the same CRS.) The partial function extractW extracts the witness of a commitment.

$$
\begin{aligned}
&\mathrm{check}_{\mathrm{zk}}(\mathrm{crs}, S \wedge S', R \wedge R') \Leftrightarrow \mathrm{check}_{\mathrm{zk}}(\mathrm{crs}, S, R) \\
&\ \text{and } \mathrm{check}_{\mathrm{zk}}(\mathrm{crs}, S', R') \\
&\mathrm{check}_{\mathrm{zk}}(\mathrm{crs}, S \vee S', R \wedge R') \Leftrightarrow \mathrm{check}_{\mathrm{zk}}(\mathrm{crs}, S, R) \\
&\ \text{or } \mathrm{check}_{\mathrm{zk}}(\mathrm{crs}, S', R') \\
&\mathrm{check}_{\mathrm{zk}}(\mathrm{crs}, A = B, R = R') \Leftrightarrow \bot \neq \mathrm{extractW}(\mathrm{crs}, A, R) \\
&\ = \mathrm{extractW}(\mathrm{crs}, B, R) \neq \bot \\
&\mathrm{check}_{\mathrm{zk}}(A \neq B, R \neq R') \Leftrightarrow \bot \neq \mathrm{extractW}(\mathrm{crs}, A, R) \\
&\ \neq \mathrm{extractW}(\mathrm{crs}, B, R') \neq \bot
\end{aligned}
$$

where $m$ is the arity of the constructor or destructor $F$.

Recall that the auxiliary verification information differs in the FMZK and the CMZK model. Hence, we have to adjust the witness extraction function extractW so as to reflect the different structure of the auxiliary

[11] true and false are encoded using the payload constructors $\varepsilon, \mathrm{string}_0, \mathrm{string}_1$.

verification information.

$$\begin{aligned}&\text{extractW}(\text{crs}, F\langle \underline{t}\rangle, F\langle \underline{u}\rangle)\\ &= F(\text{extractW}(\text{crs}, t_1, u_1), \ldots, \text{extractW}(\text{crs}, t_m, u_m))\\ &\text{extractW}(\text{crs}, \langle \text{com}(\text{crs}, t, M), U\rangle, \langle t, M\rangle) = t\end{aligned}$$

In the FMZK model, the partial function extractW is defined in the same way, except that the last equation in the FMZK model merely checks the randomness:

$$\text{extractW}(\text{crs}, \langle \text{com}(\text{crs}, t, M), U\rangle, M) = t$$

**MZK transformations.** Malleable zero-knowledge proofs allow a recipient to transform a proof even if this recipient does not know anything about the witnesses of the proof. This flexibility has been used in previous work for re-randomizing proofs, in order to achieve unlinkability to previous proofs [BCC+09], and hiding statements from received proofs, allowing a user to selectively disclose information [BMP12]. We discuss below the various transformations.

*Conjunctions.* We realize the transformations for constructing conjunctions from two proofs, splitting conjunctions, and reordering conjunctions with the destructors $\text{and}_{\text{ZK}}$ and splitAnd. For the destructor splitAnd, we need the randomness to be composed using rand.

*Commuting disjunctions.* Reordering disjunctions requires modifying the randomness tree $r$ in a proof $\text{ZK}(t, r, N)$, which cannot be accessed. We thus introduce a destructor $\text{commute}(\text{ZK}(t_1 \vee t_2, r_1 \vee r_2, N))$, which simply swaps the two literals $A$ and $B$ and the corresponding randomness subtrees $r_1$ and $r_2$.

*Re-randomization.* Re-randomization needs to modify the auxiliary verification information $r$ as well. Hence, we introduce a destructor $\text{rerPr}(\text{ZK}(t, r, N), r', N')$ that takes as input a tree $r'$ and a randomness term $N'$. The former specifies the commitment that is to be re-randomized, while the latter specifies the randomness to be used.

*Hiding public values.* Public values occurring in the statement can be hidden (i.e., made private) by means of the destructor $\text{setPub}(\text{ZK}(t_1, r_1, N), t_2, N')$, which replaces the statement $t_1$ with the statement $t_2$ and re-randomizes the proof with $N'$. For technical reasons, we require that every honest party re-randomizes the proof after hiding public values. However, the attacker can hide values of the witness without re-randomizing the proof; hence we introduce a destructor $\text{setPubA}(\text{ZK}(t_1, r_1, N), t_2)$.

In the CMZK model, setPub is restricted to hiding witnesses. Formally, $\text{setPub}(\text{ZK}(t_1, r_1, N), t_2, N')$ outputs $\perp$ if $t_2$ differs from $t_1$ in more than merely omitting unveil information.

*Functional transformations.* These transformations are modeled by the destructors applyF (for any constructor $f \in \mathbf{D} \cup \mathbf{C}$), which are applicable to commitments such that $\text{applyF}(\text{com}(\text{crs}, t, R), x) = \text{com}(\text{crs}, f(t, x), R)$. applyF can only be applied by the symbolic attacker. Together with $\text{getPub}(z)$ and $\text{setPub}(z')$ the destructors applyF allow the symbolic attacker to modify a proofs witness. In the FMZK model, a proof that is transformed using applyF and setPub might still pass verification. In the CMZK model, a modified proof will not pass verification(even though commitments can be malleable), since the randomness tree additionally carries information about the original message.

*Example 2: Usage of transformations.* Recall the statement $S_{awot}$ for Anonymous Webs of Trust from Example 1: an anonymously authenticated message $m$ for the party $A$. Assume that $m$ requests $A$ to extend the chain of trust to a party $D$ and to send $D$ a message $m'$: $m = (\texttt{extend}, D, m')$. Moreover, assume that $D$ trusts $A$. Yet, $A$ wants to hide from anybody intercepting the ZK proof that it was $A$ who extended the chain of trust to $D$. We can model this scenario by applying the following transformations. First, since $D$ trusts $A$, $A$ has a signature on its verification key from D: $s_D := \text{sig}(\text{sk}_D, \text{vk}_A, r_A)$. Second, $D$ creates an atomic proof for the validity of $s_A$, i.e., a MZK proof $\text{ZK}(S_A, r_A, N_A)$ of the statement $\exists s_D.\text{ver}_{\text{sig}}(\text{vk}_D, s_D) = \text{vk}_A$, denoted as $S_A$:

$$\begin{aligned}&\hat{\text{ver}}_{\text{sig}}\big\langle\langle c_{\text{vk}_D}, \text{uv}(\text{vk}_D, r'_{c_{\text{vk}_D}})\rangle, \langle c_{s_D}, \varepsilon\rangle\big\rangle\\ &= \langle c_{\text{vk}_A}, \text{uv}(\text{vk}_A, r_{c_{\text{vk}_A}})\rangle\end{aligned}$$

Third, $A$ computes the conjunction of $S_{awot}$ and $S_A$, i.e., $A$ applies the transformation $\text{and}_{\text{ZK}}(\text{ZK}(S_{awot}, r, N), \text{ZK}(S_A, r_D, N_D)) =: z$. Fourth, $A$ hides his own verification key $\text{vk}_A$ in the statement $S := S_{awot} \wedge S_D$ by

removing the unveiling information $\mathrm{uv}(\mathrm{vk}_A, r'_{c_{\mathrm{vk}_A}})$ from $S_A$. Similarly, $A$ removes the unveiling information $\mathrm{uv}(\mathrm{vk}_A, r_{c_{\mathrm{vk}_A}})$ from $S_{awot}$ (see above). Let $S'$ be the modified statement with the removed unveiling information. Finally $A$ applies $\mathrm{setPub}(z, S') =: z'$, to obtain an MZK proof of the following statement:

$$\exists s_B, s_C, s_D, s_A, v_A, v_B, v_C.\ \mathrm{ver}_{\mathrm{sig}}(v_B, s_B) = m \wedge \\ \mathrm{ver}_{\mathrm{sig}}(v_C, s_C) = v_B \ \wedge \mathrm{ver}_{\mathrm{sig}}(v_A, s_A) = v_C \wedge \ \mathrm{ver}_{\mathrm{sig}}(\mathrm{vk}_D, s_D) = v_A \qquad \diamond$$

### A.2.3 ZK preservation

Before introducing the symbolic attacker for the FMZK and the CMZK variation, we characterize the functional transformations available to the attacker in the FMZK model. We introduce the notion of *ZK preserving functional transformations*. Consider an MZK proof $z$ of the statement $\exists s, b.\mathrm{ver}_{\mathrm{sig}}(\mathrm{vk}, s) = m \vee A$, where $b$ occurs in $A$. By applying getPub, applyPair, and setPubA to $z$, an attacker can (in the symbolic model) produce an MZK proof $z'$ of $\exists s, b.\ \mathrm{ver}_{\mathrm{sig}}(\mathrm{vk}, s) = \mathrm{pair}(m, m) \vee A$. A successful verification of $z'$ leaks the validity of $\exists s, b.A$, which contradicts the zero-knowledge property. Hence, we only allow ZK preserving functional transformations.

We write $L(t)$ for denoting the logical formula of a statement $t$. Technically, $L(t) := \exists \underline{x}.L'(t)$, where $\underline{x}$ is the list of variables $V$ in $L'(t)$ with $L'(t)$ being recursively defined as $L'(S_1 \vee S_2) := L'(S_1) \vee L'(S_2)$, analogous for $\vee, =, \neq$, and $\hat{F}$ (for $F \in \mathrm{StF}$). The commitments with unveiling information are mapped to the message inside the commitment, and those without unveiling information are mapped to variables that are indexed by the message inside the commitment. A destructor application $f(z)$ is *ZK preserving* if the following logical formula holds for all ZK terms $z = \mathrm{ZK}(t, r, N)$ and $L(t) = \exists \underline{x}.L'(t)$:

$$\forall \underline{x} \in \mathbf{T}^n.\big(L'(\mathrm{getPub}(z)) \Rightarrow L'(\mathrm{getPub}(f(z)))\big)$$

Intuitively, the formula above says that the statement of the original proof implies the statement of the one obtained by the transformation. This, in particular, rules out the aforementioned problem related to proofs of logical disjunctions.

A functional transformation applyF is *ZK preserving* if $f$ is ZK preserving.

**ZK preservation is decidable.** Ramsey showed in 1930 that EPR formulas are decidable, i.e., all first-order logic formulas of the following form $\exists^* \underline{y} \forall^* \underline{x}.\phi(x, y)$ are decidable, where $\phi(x, y)$ is a propositional logic formula about relations [Ram29]. If we consider each atomic statement, i.e, each equation $C(\underline{x}) = C'(\underline{x})$ or $C(\underline{x}) \neq C'(\underline{x})$ as a relation $\phi'(\underline{x})$, checking ZK preservation amounts to checking an EPR formula. Hence, checking ZK preservation is decidable. A verification tool needs to check that every applied functional transformation applyF is ZK preserving.

### A.2.4 Symbolic attacker

The symbolic attacker is defined as for the basic symbolic model (see Figure 6), except that $\mathbf{C}'$ is replaced with $\mathbf{C}$ and $\mathbf{D}'$ with $\mathbf{D}$, with the additional condition (for the FMZK variation) that all destructor applications of the symbolic attacker have to be ZK preserving.

# B Full characterization of MZK-safe protocols

In this section we formally state the protocol conditions. 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 $\mathrm{garb}, \mathrm{garbEnc}$, garbSig, garbZK, or $N \in \mathbf{N}_E$ constructor in the protocol.
3. No commitment is sent to the attacker that is not inside ZK term, i.e., not the result of an term constructed by mkZK.
4. No com term that has been received over the network without being part of a ZK term and no garbCom term is used in a constructor or destructor application.

$$
\begin{aligned}
\mathrm{dec}(\mathrm{dk}(t_1), \mathrm{enc}(\mathrm{ek}(t_1), m, t_2)) &= m \\
\mathrm{ver}_{\mathrm{sig}}(\mathrm{vk}(t_1), \mathrm{sig}(\mathrm{sk}(t_1), t_2, t_3)) &= t_2 \\
\mathrm{ekof}(\mathrm{enc}(\mathrm{ek}(t_1), t_2, t_3)) &= \mathrm{ek}(t_1) \\
\mathrm{ekof}(\mathrm{garbEnc}(t_1, t_2)) &= t_1 \\
\mathrm{vkof}(\mathrm{sig}(\mathrm{sk}(t_1), t_2, t_3)) &= \mathrm{vk}(t_1) \\
\mathrm{vkof}(\mathrm{garbSig}(t_1, t_2)) &= t_1 \\
\mathrm{ekofdk}(\mathrm{dk}(t)) &= \mathrm{ek}(t) \\
\mathrm{fst}(\mathrm{pair}(t_1, t_2))) &= t_1 \\
\mathrm{snd}(\mathrm{pair}(t_1, t_2)) &= t_2 \\
\mathrm{unstring}_1(\mathrm{string}_1(s)) &= s \\
\mathrm{unstring}_0(\mathrm{string}_0(s)) &= s \\
\mathrm{equals}(x, x) &= x
\end{aligned}
\qquad
\begin{aligned}
\mathrm{isenc}(\mathrm{enc}(\mathrm{ek}(t_1), t_2, t_3)) &= \mathrm{enc}(\mathrm{ek}(t_1), t_2, t_3) \\
\mathrm{isenc}(\mathrm{garbEnc}(t_1, t_2)) &= \mathrm{garbEnc}(t_1, t_2) \\
\mathrm{issig}(\mathrm{sig}(\mathrm{sk}(t_1), t_2, t_3)) &= \mathrm{sig}(\mathrm{sk}(t_1), t_2, t_3) \\
\mathrm{issig}(\mathrm{garbSig}(t_1, t_2)) &= \mathrm{garbSig}(t_1, t_2) \\
\mathrm{iscom}(\mathrm{com}(t_1, t_2, t_3)) &= \mathrm{com}(t_1, t_2, t_3) \\
\mathrm{iscom}(\mathrm{garbCom}(t_1, t_2)) &= \mathrm{garbCom}(t_1, t_2) \\
\mathrm{iszk}(\mathrm{ZK}(t_1, t_2, t_3)) &= \mathrm{ZK}(t_1, t_2, t_3) \\
\mathrm{iszk}(\mathrm{garbZK}(t_1, t_2)) &= \mathrm{garbZK}(t_1, t_2) \\
\mathrm{isek}(\mathrm{ek}(t)) &= \mathrm{ek}(t) \\
\mathrm{isvk}(\mathrm{vk}(t)) &= \mathrm{vk}(t) \\
\mathrm{iscrs}(\mathrm{crs}(t_1)) &= \mathrm{crs}(t_1)
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{mkZK}(A = B, r, N) &= \mathrm{ZK}(A = B, r, N) \\
\mathrm{mkZK}(A \neq B, r, N) &= \mathrm{ZK}(A \neq B, r, N) \\
\mathrm{ver}_{\mathrm{zk}}(\mathrm{crs}, \mathrm{ZK}(t, r, N)) &= \mathrm{check}_{\mathrm{zk}}(\mathrm{crs}, t) \\
\mathrm{getPub}(\mathrm{ZK}(t_1, r_1, N)) &= t_1 \\
\mathrm{setPub}(\mathrm{ZK}(t_1, r_1, N), t_2) &= \mathrm{ZK}(t_2, r_1, N) \\
\mathrm{and}_{\mathrm{ZK}}(\mathrm{ZK}(t_1, r_1, N), \mathrm{ZK}(t_2, r_2, N')) &= \mathrm{ZK}(t_1 \wedge t_2, r_1 \wedge r_2, \mathrm{andRand}(N, N')) \\
\mathrm{splitAnd}(\mathrm{ZK}(t_1 \wedge t_2, r_1 \wedge r_2, \mathrm{andRand}(N, N'))) &= \mathrm{pair}(\mathrm{ZK}(t_1, r_1, N), \mathrm{ZK}(t_2, r_2, N')) \\
\mathrm{or}_{\mathrm{ZK}}(\mathrm{ZK}(t_1, r_1, N), \mathrm{ZK}(t_2, r_2, N')) &= \mathrm{ZK}(t_1 \vee t_2, r_1 \vee r_2, \mathrm{orRand}(N, N')) \\
\mathrm{commute}(\mathrm{ZK}(t_1 \vee t_2, r_1 \vee r_2, N)) &= \mathrm{ZK}(t_2 \vee t_1, r_2 \vee r_1, N) \\
\mathrm{rer}_{\mathrm{zk}}(\mathrm{ZK}(t_1, r_1, (M, N)), r_2, N', M) &= \mathrm{ZK}(\mathrm{rerPr}(t_1, r_2), \mathrm{rerPr}(r_1, r_2), (N', N)) \\
\mathrm{rer}_{\mathrm{zk}}(\mathrm{ZK}(t_1, r_1, (M, N)), r_2, N', \varepsilon) &= \mathrm{ZK}(\mathrm{rerPr}(t_1, r_2), \mathrm{rerPr}(r_1, r_2), (\mathrm{rand}(M, N'), N)) \\
\mathrm{attRer}_{\mathrm{zk}}(\mathrm{ZK}(t_1, r_1, (M, N)), r_2, N') &= \mathrm{ZK}(\mathrm{attRerPr}(t_1, r_2), \mathrm{attRerPr}(r_1, r_2), (M, N')) \\
\mathrm{open}(\mathrm{crs}(t_1), \mathrm{com}(\mathrm{crs}(t_1), t_2, t_3), \mathrm{uv}(t_2, t_3)) &= t_2 \\
\mathrm{crsof}(\mathrm{com}(crs(t_1), t_2, t_3)) &= \mathrm{crs}(t_1) \\
\mathrm{crsof}(\mathrm{garbCom}(t_1, t_2)) &= t_1 \\
\mathrm{rer}_{\mathrm{com}}(\mathrm{com}(\mathrm{crs}(t_1), t_2, t_3), t_4) &= \mathrm{com}(\mathrm{crs}(t_1), t_2, \mathrm{rand}(t_3, t_4)) \\
\mathrm{rer}_{\mathrm{com}}(\mathrm{garbCom}(t_1, t_2), t_3) &= \mathrm{garbCom}(t_1, \mathrm{rand}(t_2, t_3)) \\
\mathrm{applyF}(\mathrm{com}(\mathrm{crs}(t_1), t_2, t_3), t_4) &= \mathrm{com}(\mathrm{crs}(t_1), f(t_2, x), t_3) \text{ for } f \in \mathbf{D} \cup \mathbf{C}
\end{aligned}
$$

Figure 9: Definition of the set $\mathbf{D}$ of destructors.

5. For every commitment $\mathrm{com}(c, m, N)$ that is published in a ZK proof it holds that if $\mathrm{com}(c, m, N)$ is a witness commitment, $\mathrm{uv}(m, N)$ is never revealed.

6. The constructor ZK and the destructors setPubA and applyF (for $F \in \mathbf{C} \cup \mathbf{D}$) is not used in the protocol.

7. The last argument of a com, enc, sig constructor and of a mkZK, rerPr, and $\mathrm{rer}_{\mathrm{com}}$ destructor are fresh nonces. These nonces are not used anywhere else except in case of enc as part of a subterm of the second argument in a com-node if the following holds: Construct a symbolic knowledge $S$ out of the path from the com-node to the root, including the com-node. Let $N$ be the nonce of an enc constructor and $m$ be the corresponding (symbolic) plaintext. Then, we require that if $S \vdash_w N$ holds true then also $S \vdash_w m$ holds true.

8. A dk-node is only used as first argument for dec-node or as subterm of the third argument in a ZK-node.

9. A sk-node is only used as first argument for sig-node or as subterm of the third argument in a ZK-node.

If no rules applies, rerPr outputs $\bot$. In the following $m$ is the arity of the constructor or destructor $F$. attRerPr is defined just like rerPr except that in the base case only the rule for attRerPr is valid.

$$\begin{aligned}
\mathrm{rerPr}(S \wedge S', Q \wedge Q') &= \mathrm{rerPr}(S, Q) \wedge \mathrm{rerPr}(S', Q') \\
\mathrm{rerPr}(S \vee S', Q \vee Q') &= \mathrm{rerPr}(S, Q) \vee \mathrm{rerPr}(S', Q') \\
\mathrm{rerPr}(A = A', Q = Q') &= \mathrm{rerPr}(A, Q) = \mathrm{rerPr}(A', Q') \\
\mathrm{rerPr}(A \neq A', Q \neq Q') &= \mathrm{rerPr}(A, Q) \neq \mathrm{rerPr}(A', Q')
\end{aligned}$$

$$\begin{aligned}
&\mathrm{rerPr}(F\langle t_1, \ldots, t_n\rangle, F\langle u_1, \ldots, u_n\rangle) \\
&\qquad = F\langle \mathrm{rerPr}(t_1, u_1), \ldots, \mathrm{rerPr}(t_n, u_n)\rangle \\
&\mathrm{rerPr}(\langle \mathrm{com}(\mathrm{crs}(n), t_1, t_2), \mathrm{uv}(t_3)\rangle, t_4) \\
&\qquad = \langle \mathrm{com}(\mathrm{crs}(n), t, \mathrm{rerPr}(t_2, t_4), \mathrm{uv}(\mathrm{rerPr}(t_3, t_4))\rangle
\end{aligned}$$

$$\begin{aligned}
\mathrm{rerPr}(t, \varepsilon) &= t \\
\mathrm{rerPr}((M, N), (N', M)) &= (N', N) \\
\mathrm{rerPr}((M, N), (N', \varepsilon)) &= (\mathrm{rand}(M, N'), N) \\
\mathrm{attRerPr}((M, N), N') &= (M, N')
\end{aligned}$$

Figure 10: The definition of the re-randomization rerPr

10. The first argument of a dec-computation node is a dk-node.
11. The first argument of a sig-computation node is a sk-node.
12. A mkZK-computation node is consistent: for every com-computation node in the statement of mkZK the destructor crsof would not output $\bot$.
13. The first argument of a com-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.
14. The first argument of a $\mathrm{ver}_{\mathrm{zk}}$-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.
15. For the relation $R_{\mathrm{adv}}^{\mathrm{sym}}$ it holds: There is an efficient algorithm **SymbExtr**, that given a term $M$ together with a set $S$ of terms (which was generated according by any protocol satisfying the protocol conditions above), outputs a term $N$, such that there are $t, t' \in \mathbf{T}$ such that $S \vdash \mathrm{ZK}(t, r, t')$, $\mathrm{extrWit}(t) = N, \mathrm{extrSta}(t) = M$ and $(N, M) \in R_{\mathrm{adv}}^{\mathrm{sym}}$ or outputs $\bot$ if there is no such term $N$. We call a relation satisfying this property symbolically extractable.
16. The relation $R_{\mathrm{adv}}^{\mathrm{sym}}$ is efficiently decidable.
17. All ZK-transformations are only applied to honestly generated proofs or to terms $t$ for which $\mathrm{ver}_{\mathrm{zk}}(\mathrm{crs}(t'), t) \neq \mathrm{false}$ for some $t' \in \mathbf{N}_P$.
18. The protocol only uses polynomially many different nonces.
19. The last node of a ZK-transformations is a fresh nonce. This nonce is not used anywhere else.

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

$$
\begin{aligned}
L'(S_1 \vee S_2) &:= L'(S_1) \vee L'(S_2) \\
L'(S_1 \wedge S_2) &:= L'(S_1) \wedge L'(S_2) \\
L'(A = B) &:= L'(A) = L'(B) \\
L'(A \neq B) &:= L'(A) \neq L'(B) \\
L'(F\langle X_1, \ldots, X_n\rangle) &:= F(L'(X_1), \ldots, L'(X_n)) \\
n &= \text{arity}(F) \\
L'(\langle \text{com}(\text{crs}, m, r), \varepsilon\rangle) &:= x_m \text{ , for } x_m \in V \\
L'(\langle \text{com}(\text{crs}, m, r), \text{uv}(m, r)\rangle) &:= m
\end{aligned}
$$

Figure 11: The mapping $L'$ from statement terms to logical formulas.

# C 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 $\text{unstring}_i$ is the inverse of $\text{string}_i$.

1. The implementation is an implementation according to CoSP [BHU09].
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 $N \in \mathbf{N}$), $A_{\text{enc}}$ have type ciphertext, $A_{\text{ek}}$ have type encryption key, $A_{\text{dk}}$ have type decryption key, $A_{\text{sig}}$ have type signature, $A_{\text{vk}}$ have type verification key, $A_{\text{sk}}$ have type signing key, $A_{\text{crs}}$ have type common reference string, $A_{\text{ZK}}$ have type zero-knowledge proof, $A_{\text{pair}}$ have type pair, and $A_{\text{string}_0}, A_{\text{string}_1}, A_\varepsilon$ have type payload string.
3. The implementation $A_N$ for nonces $N \in \mathbf{N}_P$ implement uniform distributions on $\{0,1\}^k$ where $k$ is the security parameter.
4. If $A_{\text{dec}}(\text{dk}_N, m) \neq \perp$ then $A_{\text{ekof}}(m) = \text{ek}_N$, i.e. the decryption only succeeds if the corresponding encryption key can be extracted out of the ciphertext.
5. $A_{\text{vkof}}(A_{sig}(A_{\text{sk}}(x), y, z)) = A_{\text{vk}}(x)$ for all $y \in \{0,1\}^*$ and $x, z$ nonces. If $e$ is of type signature then $A_{\text{vkof}}(e) \neq \perp$, otherwise $A_{\text{vkof}}(e) = \perp$.
6. For all $m, k \in \{0,1\}^*$, $k$ having type encryption key, and $r \neq r' \in \{0,1\}^*$ with $|r| = |r'|$ holds that $A_{\text{enc}}(k, m, r)$ and $A_{\text{enc}}(k, m, r')$ 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 $A_{\text{sig}}(k, m, r)$ and $A_{\text{sig}}(k, m, r')$ are equal with negligible probability.
8. The implementations $A_{\text{ek}}$, $A_{\text{dk}}$, $A_{\text{enc}}$, and $A_{\text{dec}}$ belong to an encryption scheme $(\text{KeyGen}_{\text{enc}}, \text{ENC}, \text{DEC})$ which is IND-CCA secure.
9. The implementations $A_{\text{vk}}$,$A_{\text{sk}}$,$A_{\text{sig}}$, and $A_{\text{ver}_{\text{sig}}}$ belong to a signature scheme $(\text{KeyGen}_{\text{sig}}, \text{SIG}, \text{VER}_{\text{sig}})$ 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 $A_{\text{fst}}(A_{\text{pair}}(m_1, m_2)) = m_1$ and $A_{\text{snd}}(A_{\text{pair}}(m_1, m_2)) = m_2$

$$
\begin{aligned}
\mathrm{crsof}(S_1 \vee S_2) &:= \mathrm{crsof}(S_1) \quad \text{if } \mathrm{crsof}(A) = \mathrm{crsof}(B)\\
\mathrm{crsof}(S_1 \wedge S_2) &:= \mathrm{crsof}(S_1) \quad \text{if } \mathrm{crsof}(A) = \mathrm{crsof}(B)\\
\mathrm{crsof}(A = B) &:= \mathrm{crsof}(A) \quad \text{if } \mathrm{crsof}(A) = \mathrm{crsof}(B)\\
\mathrm{crsof}(A \neq B) &:= \mathrm{crsof}(A) \quad \text{if } \mathrm{crsof}(A) = \mathrm{crsof}(B)\\
\mathrm{crsof}(F\langle X_1, \ldots, X_n\rangle) &:= \mathrm{crsof}(X_1) \quad \text{if } \forall i, j.\mathrm{crsof}(X_i) = \mathrm{crsof}(X_j)\\
n &= \mathrm{arity}(F)\\
\mathrm{crsof}(\langle \mathrm{com}(\mathrm{crs}, m, r), \varepsilon\rangle) &:= \mathrm{crs}\\
\mathrm{crsof}(\langle \mathrm{com}(\mathrm{crs}, m, r), \mathrm{uv}(r)\rangle) &:= \mathrm{crs}
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{extrSta}(S_1 \vee S_2) &:= (\mathrm{extrSta}(S_1), \mathrm{extrSta}(S_2))\\
\mathrm{extrSta}(S_1 \wedge S_2) &:= (\mathrm{extrSta}(S_1), \mathrm{extrSta}(S_2))\\
\mathrm{extrSta}(A = B) &:= (\mathrm{extrSta}(A), \mathrm{extrSta}(B))\\
\mathrm{extrSta}(A \neq B) &:= (\mathrm{extrSta}(A), \mathrm{extrSta}(B))\\
\mathrm{extrSta}(F\langle X_1, \ldots, X_n\rangle) &:= F\langle \mathrm{extrSta}(X_1), \ldots, \mathrm{extrSta}(X_n)\rangle\\
n &= \mathrm{arity}(F)\\
\mathrm{extrSta}(\langle \mathrm{com}(\mathrm{crs}, m, r), \varepsilon\rangle) &:= \varepsilon\\
\mathrm{extrSta}(\langle \mathrm{com}(\mathrm{crs}, m, r), \mathrm{uv}(m, r)\rangle) &:= m
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{extrNon}(S_1 \vee S_2) &:= (\mathrm{extrNon}(S_1), \mathrm{extrNon}(S_2))\\
\mathrm{extrNon}(S_1 \wedge S_2) &:= (\mathrm{extrNon}(S_1), \mathrm{extrNon}(S_2))\\
\mathrm{extrNon}(A = B) &:= (\mathrm{extrNon}(A), \mathrm{extrNon}(B))\\
\mathrm{extrNon}(A \neq B) &:= (\mathrm{extrNon}(A), \mathrm{extrNon}(B))\\
\mathrm{extrNon}(F\langle X_1, \ldots, X_n\rangle) &:= (\mathrm{extrNon}(X_1), \ldots, \mathrm{extrNon}(X_n))\\
n &= \mathrm{arity}(F)\\
\mathrm{extrNon}(\langle \mathrm{com}(\mathrm{crs}, m, r), \varepsilon\rangle) &:= r\\
\mathrm{extrNon}(\langle \mathrm{com}(\mathrm{crs}, m, r), \mathrm{uv}(m, r)\rangle) &:= r
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{extrWit}(\langle \mathrm{com}(\mathrm{crs}, m, r), \varepsilon\rangle) &:= m\\
\mathrm{extrWit}(\langle \mathrm{com}(\mathrm{crs}, m, r), \mathrm{uv}(m, r)\rangle) &:= \varepsilon
\end{aligned}
$$

extrWit is defined just like extrNon except for the base cases.
The algorithms $A_{\mathrm{extrSta}}$ and $A_{\mathrm{crsof}}$ are defined just like the symbolic counterpart except that the case distinction is performed using the decoding functions $d_\wedge, d_\vee, d_=, d_\neq, d_F$.

Figure 12: The symbolic extraction destructors.

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 $\mathrm{KeyGen}_{\mathrm{sig}}$, then holds for all $m, n \in \{0,1\}^*$: $A_{\mathrm{vkof}}(A_{\mathrm{sig}}(\mathrm{sk}, m, n)) = \mathrm{vk}$.

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

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

17. For $m \in \{0,1\}^*$ holds $A_{\text{unstring}_i}(A_{\text{string}_i}(m)) = m$ for $i \in \{0,1\}$ and $A_{\text{string}_0}(m) \neq A_{\text{string}_1}(m)$.

18. For all $m \in \{0,1\}^*$ of type zero-knowledge proof holds that $\text{iszk}(m) = m$ and if $m$ has not type zero-knowledge proof, then $\text{iszk}(m) = \perp$. 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 $A_{\text{enc}}(k, m, n) = \perp$. The same has to hold for the type signing key and the implementation of signatures.

20. The implementations $A_{\text{crs}}, A_{\text{ZK}}, A_{\text{ver}_{\text{zk}}}$ belongs to a malleable non-interactive zero-knowledge argument of knowledge $(\mathbf{K}, \mathbf{P}, \mathbf{V})$, as defined in Definition 14. In the CMZK model, the implementations $A_{\text{crs}}, A_{\text{ZK}}, A_{\text{ver}_{\text{zk}}}$ additionally have to satisfy controlled-malleable simulation-sound extractability in the sense of Definition 15.

21. For all $z \in \{0,1\}^*$ holds $A_{\text{ver}_{\text{zk}}}(\text{crsof}(z), z) \in \{0,1\}$, where $A_{\text{ver}_{\text{zk}}}(\text{crsof}(z), z) = \text{true}$ 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 $\text{ver}_{\text{zk}}(\text{crsof}(z), z) = \text{false}$.

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

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

25. If $z := A_{\text{ZK}}(\bar{m}) \neq \perp$ then $A_{\text{ver}_{\text{zk}}}(A_{\text{crsof}}(z), z) = 1$.

26. If $(x, w) \notin R_{\text{hon}}^{\text{comp}}$ then for all $c, r \in \{0,1\}^*$, it holds $A_{\text{ZK}}(c, x, w, r) = \perp$.

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

28. 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 $A_{\text{dec}}(d, A_{\text{enc}}(p(d), m, n)) = m$, i.e. $p$ computes the encryption key corresponding to $d$. The analogous statement has to holds for signing keys and verification keys.

29. Every transformation is strongly derivation private in the sense of Definition 16.

30. $A_{\text{crs}}$, $A_{\text{com}}$, $A_{\text{open}}$ belong to an extractable non-interactive computationally binding and hiding commitment scheme.

**NIZK arguments of knowledge.** We present the full definition of non-interactive zero-knowledge arguments of knowledge. This definition is required for the FMZK model. For the CMZK model, which is much better suited for verification, we additionally require controlled-malleable simulation-sound extractability, as defined in Definition 15.

**Definition 14** (NIZK arguments of knowledge [CKLM12])**.** *A* malleable non-interactive zero-knowledge argument of knowledge *for relations $R_{\text{hon}}^{\text{comp}}, R_{\text{adv}}^{\text{comp}}$ is a tuple of polynomial-time algorithms $(\boldsymbol{K}, \boldsymbol{P}, \boldsymbol{V})$ such that there exist polynomial-time algorithms $(\boldsymbol{E}, \boldsymbol{S})$ and the following properties hold:*

- Malleable: *Let $T$ be an allowable set of transformations for $R_{\text{hon}}^{\text{comp}}$. Then this proof system is* malleable *with respect to $T$ if there exists an efficient algorithm ZKEval that on input $(crs, T, x_i, proof_i)$, where $t \in T$ is an n-ary transformation and $V(crs, x_i, proof_i) = 1$ for all $i, 1 \leq i \leq n$, outputs a valid proof proof for the statement $x = T_x(x_i)$ (i.e., a proof proof such that $V(crs, x, proof) = 1$).*

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

- Zero-Knowledge: *Fix a nonuniform polynomial-time oracle adversary $\mathcal{A}$. For given crs, simtd, let $\mathcal{O}_{\boldsymbol{P}}(x,w) := \boldsymbol{P}(x,w,crs)$ if $(x,w) \in R_{\text{hon}}^{\text{comp}}$ and $\mathcal{O}_{\boldsymbol{P}}(x,w) := \perp$ otherwise, and let $\mathcal{O}_{\boldsymbol{S}}(x,w) := \boldsymbol{S}(x,crs,simtd)$ if $(x,w) \in R_{\text{hon}}^{\text{comp}}$ and $\mathcal{O}_{\boldsymbol{S}}(x,w) := \perp$ otherwise. Then $|\Pr[\mathcal{A}^{\mathcal{O}_P}(1^\eta, crs) = 1 : (crs, simtd, extd) \leftarrow \boldsymbol{K}(1^\eta)] - \Pr[\mathcal{A}^{\mathcal{O}_S}(1^\eta, crs) = 1 : (crs, simtd, extd) \leftarrow \boldsymbol{K}(1^\eta)]|$ is negligible in $\eta$.*

- Extractability: *Let a nonuniform polynomial-time oracle adversary $\mathcal{A}$ be given. Let $(crs, simtd, extd) \leftarrow \boldsymbol{K}(1^\eta)$. Let $(x, proof) \leftarrow \mathcal{A}(1^\eta, crs)$. Let $w \leftarrow \boldsymbol{E}(x, proof, extd)$. Then with overwhelming probability, if $\boldsymbol{V}(x, proof, crs) = 1$ then $R_{\text{adv}}^{\text{comp}}(x,w) = 1$.*

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

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

- Deterministic verification and extraction: *The algorithms $V$, $E$ are deterministic.*[12]

**Controlled malleability.** As shown by Fuchsbauer [Fuc10, Lemma 6], it is possible to transform the witness inside a Groth-Sahai proofs. In a recent work, Chase, Kohlweiss, Lysyanskaya, and Meiklejohn, introduced a relaxation of simulation-sound extractability for controlled malleability: controlled-malleable simulation sound extractability [CKLM12], which states that an attacker can at most perform a fixed set of unary transformations on an honestly generated proof. We require that for the CMZK model the zero-knowledge proof scheme satisfies this notion of controlled-malleability w.r.t. to the following two transformations: re-randomization of proofs and selectively hiding public information, e.g., used for existential quantification. In that work, additionally a generic construction for controlled-malleability out of non-interactive zero-knowledge arguments of knowledge is presented. There are two recent and practical examples that satisfy this notion of controlled malleability: the Groth-Sahai proof scheme [GS08], after the generic construction from [CKLM12] is applied to it, and the malleable SNARKs of Chase, Kohlweiss, Lysyanskaya, and Meiklejohn [Cha13].

We stress that we do not need to require that conjunctions are possible since simply sending two proofs proves the conjunctions of the two respective statements. For conjunctions it is more important to require that a proof about the logical conjunction $A \wedge B$ of two proofs about $A$ and $B$, respectively, should be indistinguishable from a fresh proof about $A \wedge B$. This additional requirement (derivation privacy) is described below.

**Definition 15** (Controlled malleability [CKLM12])**.** *Let $(\boldsymbol{K}, \boldsymbol{P}, \boldsymbol{V})$ be a non-interactive zero-knowledge argument of knowledge for relations $R_{\text{hon}}^{\text{comp}}, R_{\text{adv}}^{\text{comp}}$ such that there exist polynomial-time algorithms $(\boldsymbol{E}_1, \boldsymbol{E}_2, \boldsymbol{S}_1, \boldsymbol{S}_2)$.*

*Let $T$ be an allowable set of unary transformations for the relation $R$ such that membership in $T$ is efficiently testable. Let $SE_1$ be an algorithm that, on input $1^k$ outputs $(crs, simtd, extd)$ such that $(crs, simtd)$ is distributed identically to the output of $S_1$. Let $A$ be given, and consider the following game:*

- *Step 1. $(crs, simtd, extd) \leftarrow SE_1(1^k)$.*
- *Step 2. $(x, proof) \leftarrow \mathcal{A}^{\boldsymbol{S}_2(crs, simtd, \cdot)}(crs, extd)$*
- *Step 3. $(w, x', T) \leftarrow \boldsymbol{E}_2(crs, extd, x, proof)$.*

*Then, $(\boldsymbol{K}, \boldsymbol{P}, \boldsymbol{V})$ satisfies* controlled-malleable simulation-sound extractability *(CM- SSE, for short).*

*We say that the NIZKPoK satisfies controlled-malleable simulation-sound extractability (CM- SSE, for short) if for all PPT algorithms $\mathcal{A}$ there exists a negligible function $\mu(\cdot)$ such that the probability (over the choices of $SE_1$, $\mathcal{A}$, and $\boldsymbol{S}_2$) that $V(crs, x, proof) = 1$ and $(x, proof) \notin Q$ (where $Q$ is the set of queried statements and their responses) but either (1) $w \neq \perp$ and $(x,w) \notin R$; (2) $(x', T) \neq (\perp, \perp)$ and either $x' \notin Q_x$ (the set of queried instances), $x \neq T_x(x')$, or $T \notin T$; or (3) $(w, x', T) = (\perp, \perp, \perp)$ is at most $\mu(k)$.*

[12] This is not a necessary requirement but we require them to be deterministic for the sake of simplifying the computational soundness proof.

**Strong derivation privacy.** We require that a transformed proof is indistinguishable from a freshly produced proofs, in particular for selectively hiding public information or for constructing a proof about the logical conjunction of two given proofs. This property has been formalized as *strong derivation privacy* by Chase, Kohlweiss, Lysyanskaya, and Meiklejohn [CKLM12].

**Definition 16** (Strong derivation privacy [CKLM12])**.** *Let* $(\boldsymbol{K}, \boldsymbol{P}, \boldsymbol{V})$ *be a malleable non-interactive zero-knowledge argument of knowledge for relations* $R_{\text{hon}}^{\text{comp}}, R_{\text{adv}}^{\text{comp}}$ *such that there exist polynomial-time algorithms* $(\boldsymbol{S}_1, \boldsymbol{S}_2)$*.*

*For a given adversary* $\mathcal{A}$*, and a bit* $b$*, let* $p_b^A(k)$ *be the probability of the event that* $b' = 0$ *in the following game:*

- *Step 1.* $(crs, simtd) \leftarrow S_1(1^k)$*.*
- *Step 2.* $(state, x_1, proof_1, \cdots, x_q, proof_q, T) \leftarrow A(crs, simtd)$*.*
- *Step 3. If* $V(crs, x_i, proof_i) = 0$ *for some* $i$*,* $(x_1, ..., x_q)$ *is not in the domain of* $T_x$*, or* $T \notin T$ *, abort and output* $\perp$*. Otherwise, form*

$$proof \leftarrow \begin{cases} S_2(crs, simtd, T_x(x_1, ..., x_q)) & \text{if } b = 0 \\ ZKEval(crs, T, x_i, proof_i) & \text{if } b = 1. \end{cases}$$

- *Step 4.* $b \leftarrow A(state, proof)$*.*

*We say that the proof system is strongly derivation private if for all PPT algorithms A there exists a negligible function* $\mu(\cdot)$ *such that* $|p_0^A(k) - p_1^A(k)| < \mu(k)$*.*

**Computational ZK Relation.** For the computational soundness proof and for the computational implementation, we need four destructors $\text{extrSta}, \text{extrWit}, \text{crsof}, \text{extrNon}$, which traverse through the statement and extract the statement, the witness, the CRS, and the list of nonces, respectively (see Appendix 12). The computational ZK relation $R_{\text{hon}}^{\text{comp}}$ is then defined as follows:

$$\big\{(\text{img}_\eta(x), \text{img}_\eta(w)) \mid \begin{array}{l} t \text{ is a valid symbolic statement} \wedge \\ x = \text{extrSta}(t), w = \text{extrWit}(t) \\ \text{for a consistent } \eta \end{array} \big\}$$

We assume efficient encoding algorithms Let $e_\wedge, e_\vee, e_=, e_f$ ($f \in \text{StF}$, see Section A.2.1). The function $\text{img}_\eta$ depends on an environment $\eta$, 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).

We define $\text{img}_\eta$ as follows:

$$\begin{array}{l} \text{img}_\eta(S_1 \wedge S_2) := e_\wedge(\text{img}_\eta(S_1), \text{img}_\eta(S_2)) \\ \text{img}_\eta(S_1 \vee S_2) := e_\vee(\text{img}_\eta(S_1), \text{img}_\eta(S_2)) \\ \text{img}_\eta(A = B) := e_=(\text{img}_\eta(A), \text{img}_\eta(B)) \\ \text{img}_\eta(f(x_1, \ldots, x_n)) := e_f(\text{img}_\eta(x_1), \ldots, \text{img}_\eta(x_n)) \\ \qquad \text{for } f \in \text{StF} \\ \text{img}_\eta(f(x_1, \ldots, x_n)) := A_f(\eta(x_1), \ldots, \eta(x_n)) \\ \qquad \text{for } f \in \mathbf{C} \cup \mathbf{D} \setminus \text{StF} \\ \text{img}_\eta(\text{pair}(\text{com}(\text{crs}(N), m, r), \text{uv}(m, r))) \\ := A_{\text{pair}}(A_{\text{com}}(\text{crs}(N), \text{img}_\eta(m), \eta(r)), A_{\text{uv}}(\text{img}_\eta(m), \eta(r))) \\ \text{img}_\eta(\text{pair}(\text{com}(\text{crs}(N), m, r), \varepsilon)) \\ := A_{\text{pair}}(A_{\text{com}}(\text{crs}(N), \text{img}_\eta(m), \eta(r)), A_\varepsilon()) \\ \text{img}_\eta(N) := \eta(N) \text{ for } N \in \mathbf{N} \end{array}$$

where $X_m$ denotes the computational encoding of a variable with index $m \in \mathbf{T}$.

**Definition 17.** *Let $\mathcal{E}$ be the set of all partial functions $\eta : \boldsymbol{T} \to \{0,1\}^*$. We will call such an $\eta$ an environment.*

*Let an implementation $A$ for the symbolic model by given. Define the partial function $img_\eta : \boldsymbol{T} \to \{0,1\}^*$ for $\eta \in \mathcal{E}$ by taking the first matching rule:*

- *For a nonce $N$ define $img_\eta(N) := \eta(N)$*
- *For a term $t = crs(N)$ define $img_\eta(crs(N)) := \eta(t)$*
- *For a term $t = ZK(crs(N), x, w, M)$ define $img_\eta(t) := \eta(t)$*
- *Let $C$ be a constructor from $\{ek, dk, vk, sk, enc, sig, crs, garbZK, garbSig, garbEnc, garb\}$. For $t = C(t_1, \ldots, t_{n-1}, N)$ with $N \in \boldsymbol{N}_E$ define $img_\eta(t) := \eta(t)$.*
- *For a term $C(t_1, \ldots, t_n)$ define $img_\eta(C(t_1, \ldots, t_n)) := A_C(img_\eta(t_1), \ldots, img_\eta(t_n))$ , if for all $i$ we have $img_\eta(t_i) \neq \perp$, and $\perp$ otherwise.*

*An environment $\eta$ is consistent if the following conditions are satisfied:* [13]

- *$\eta$ is injective.*
- *For each constructor $C$ we require that the bitstring $img_\eta(C(t_1, \ldots, t_n))$ has the type as follows: The constructors $enc, garbEnc$ are mapped to type ciphertext, $sig, garbSig$ to signatures, $ZK, garbZK$ to ZK proofs, $ek, dk, vk, sk$ to encryption, decryption, verification, signing key, respectively. $crs$ to common reference string, pair to pair, $string_0, string_1, \varepsilon$ to payload-string, $\boldsymbol{N}$ to nonce, garb has none of these types.*
- *$A_{ekof}(img_\eta(enc(ek(N), t, M))) = img_\eta(ek(N))$ for all $N, M \in \boldsymbol{N}_P$, $t \in \boldsymbol{T}$.*
- *For all $t = sig(sk(N), u, M)$ with $N, M \in \boldsymbol{N}, u \in \boldsymbol{T}$ it holds: $ver_{sig}(vkof(t), t) \neq \perp$ implies that $A_{ver_{sig}}(img_\eta(vkof(t)), img_\eta(t)) = img_\eta(u)$.*
- *For $t = ZK(crs(N), x, , w, M)$ with $M \in \boldsymbol{N}$ holds:*
  1. *$A_{ver_{zk}}(img_\eta(crs(N)), \eta(t)) = 1$*
  2. *$A_{getPub}(\eta(t)) = img_\eta(x)$*
  3. *$A_{crsof}(\eta(t)) = img_\eta(crs(N))$*
- *For all $t_1, t_2 \in \boldsymbol{T}$ it holds that $A_{ver_{sig}}(img_\eta(garbSig(t_1, t_2))) = \perp$*
- *For all $N, M \in \boldsymbol{N}$, $t \in \boldsymbol{T}$ it holds that $A_{dec}(img_\eta(dk(N)), img_\eta(enc(ek(N), t, M))) = img_\eta(t)$ and $img_\eta(t) \neq \perp$.*
- *For all $enc(ek(N), t, M) \in \boldsymbol{T}$ it holds: If $img_\eta(enc(ek(N), t, M)) =: c \neq \perp$, then it follows $A_{ekof}(c) = img_\eta(ek(N))$.*
- *For all $enc(ek(N), t, M) \in \boldsymbol{T}$ it holds: If $img_\eta(enc(ek(N), t, M)) \neq \perp$ and $d \in \{0,1\}^*$ such that $img_\eta(ek(N)) = p(d)$* [14]*, then it follows that $A_{dec}(d, img_\eta(enc(ek(N), t, M))) = img_\eta(t)$.*

Given these notions, we formalize the conditions on $R^{\text{comp}}_{\text{hon}}, R^{\text{comp}}_{\text{adv}}$ in the following definition.

**Definition 18** (Implementation of relations)**.** *A pair of relations $R^{\text{comp}}_{\text{hon}}, R^{\text{comp}}_{\text{adv}}$ on $\{0,1\}^*$ implement a relation $R^{\text{sym}}_{\text{adv}}$ on $\boldsymbol{T}$ with usage restriction $R^{\text{sym}}_{\text{hon}}$ if the following conditions hold for any consistent $\eta \in \mathcal{E}$:*

*(i)* $(x, w) \in R^{\text{sym}}_{\text{hon}}$ *and* $img_\eta(x) \neq \perp \neq img_\eta(w) \implies (img_\eta(x), img_\eta(w)) \in R^{\text{comp}}_{\text{hon}}$

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

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

**Strong derivation privacy.** We use the definition of strong derivation privacy, as in introduced by [CKLM12].

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

[14] Where $p$ is the function defined in implementation condition 28.

### C.1 Construction for the CMZK realization

Soundly realizing the more restricted form of malleability mandated by the CMZK model requires additional strengthening of the ZK proofs. We use the construction of Chase, Kohlweiss, Lysyanskaya, and Meiklejohn that realizes an efficient NIZKAoK from a non-interactive witness indistinguishable argument of knowledge (NIWIAoK)[15] and a EUF signature scheme. The construction, in particular. applies to the Groth-Sahai proof scheme [GS08].

We define the set of transformations for the construction to contain commute, and we construct a proof of a conjunction $A \wedge B$ such that there is one signature for $A$ and one for $B$.

## D Connecting the symbolic model to the general CMZK symbolic model

In the main body of the paper, we present a symbolic model that is far simpler than the CMZK model, in particular for automated verification tools. For a restricted class of protocols, however, the simpler model is as expressive as the CMZK model. In the simpler model of the main body, the destructors for encryption, signatures, pairs, and payload strings are defined as in Figure 5 and the rules for all other destructors are illustrated in the main body in Table 1.

In this section, we will show that it for a restricted set of protocols there is a translation *trans* from bi-protocols in the simpler symbolic model to bi-protocols in the CMZK model such that the following holds: for all bi-protocols $\Pi$ in the simpler symbolic model, if $\Pi$ is symbolically indistinguishable then $trans(\Pi)$ is symbolically indistinguishable (in the CMZK model) as well.

### D.1 The restricted protocol class in the CMZK model

We restrict the class of protocol to protocols that only use one zero-knowledge statement and conjunctions of this statement.[16] Let $MZK - safe_{\text{AWOT}}$ be the restricted class.

### D.2 The translation

Initially, one CRS is constructed $\text{crs}(\tilde{r})$ and sent to the attacker. The translation assumes a fixed statement. *trans* is recursively defined over a bi-protocol $\Pi$. For computation nodes, *trans* replaces a node $\nu$ with a subtree $t_{yes}$ at the yes-branch and a subtree $t_{no}$ at the no-branch, with a subtree $t_c$ of computation nodes, as defined in the following recursive definition. The no-branches of all computation nodes in $t_c$ have the subtree $t_{no}$ and the last yes-branch has the subtree $t_{yes}$. *tree* is the mapping from a pair of list of commitments and randomness symbols to a statement tree (in the sense of the CMZK model) that fits the structure of the AWOT statement. Analogously, *randtree* is a mapping from a $\mathsf{zkp}((r_j)_{j=0}^n)$ computation node to a randomness tree (in the sense of the CMZK model) that uses $(r_j)_{j=1}^n$.

1. $trans(\mathsf{com}(m, R))) := \text{com}(\text{crs}(\tilde{r}), m, R)$
2. $trans(\mathsf{zkp}((r_j)_{j=0}^n) := \text{ZK}(\varepsilon, randtree(\mathsf{zkp}((r_j)_{j=0}^n)), (r)_0)$
3. $trans(\mathsf{verzk}^{\text{AWOT}}(z, (c_j)_{j=1}^n))$
   $:= \text{ver}_{\text{zk}}(\text{crs}(\tilde{r}), \text{setPub}(trans(z), tree((trans(c_j))_{j=1}^n, \varepsilon))$
4. $trans(\mathsf{hide}^i(z, (c_j)_{j=1}^n, (o_j)_{j=1}^n)) := \text{setPub}(trans(z), tree(trans((c_j)_{j=1}^n), trans((o'_j)_{j=1}^n))$,
   where $o'_i := o_j$ for $i \neq j$ and $o_i := \varepsilon$
5. $trans(\mathsf{rerand}^i(z, (c_j)_{j=1}^n, (o_j)_{j=1}^n, r')) := \text{rer}_{\text{zk}}(zk, encode^i_{rer}(r'))$
   where $zk = \text{setPub}(trans(z), tree(trans(c_j)_{j=1}^n, trans((o_j)_{j=1}^n)))$
6. $trans(\mathsf{att\text{-}rerand}^i(z, (c_j)_{j=1}^n, (o_j)_{j=1}^n, r')) := \text{attRer}_{\text{zk}}(zk, encode^{i,att}_{rer}(r'))$
   where $zk = \text{setPub}(trans(z), tree(trans(c_j)_{j=1}^n)), trans((o_j)_{j=1}^n))$

[15] In particular, a FMZK realization constitutes a NIWIAoK.
[16] Our approach can easily be extended to constantly many disjoint zero-knowledge statements.

7. $trans(\text{pair}(t_1, t_2)) := \text{pair}(trans(t_1), trans(t_2))$
8. $trans(\text{enc}(t_1, t_2, t_3)) := \text{enc}(t_1, trans(t_2), t_3)$
9. $trans(\text{sig}(t_1, t_2, t_3)) := \text{sig}(t_1, trans(t_2), t_3)$
10. $trans(p) := p$, if no other case applies

$encode^i_{rer}(r)$ denotes a function that encodes the randomness $r$ such that it replaces the $i$-th randomness in the ZK-proof.

For all other kinds of nodes, trans does not change the protocol: *trans* maps control nodes to control nodes (without changing the labels), input nodes to input nodes, and output nodes to output nodes, where the references are appropriately adjusted.

**Lemma 5** (The translation preserves symbolic indistinguishability)**.** *For a bi-protocol* $\Pi$ *in the class* $MZK - safe_{AWOT}$*,* $trans(\Pi)$ *is symbolically indistinguishable (in the CMZK model) if* $\Pi$ *is symbolically indistinguishable in the simpler model.*

*Proof.* Given a bi-protocol $\Pi$, it suffices to show (in contraposition) that $\Pi$ is symbolically distinguishable whenever $trans(\Pi)$ is distinguishable. Assume $trans(\Pi)$ is symbolically distinguishable and let $V_{In}$ be an arbitrary but fixed attacker strategy for $trans(\Pi)$.

- $parse((\texttt{in}, (O, t)) :: vr) = (\texttt{in}, ((star, O), trans^{-1}(t))) :: parse(vr)$
- $parse((\texttt{out}, t) :: vr) = (\texttt{out}, trans^{-1}(t)) :: parse(vr)$
- $parse((\texttt{control}, (l, l')) :: vr) = (\texttt{control}, (l, l')) :: parse(vr)$

**Claim 5.1.** *For view* $V_b$ *that matches the attacker strategy* $V_{In}$*,* $parse(V_b)$ *and* $V_b$ *have the same structure and the same out-metadata in the sense of Condition 1 and Condition 2.*

*Proof.* Recall that $trans(\Pi)$ does change the control flow of the bi-protocol; it merely maps each computation node $\nu$ in $\Pi$ to a subtree $t_c = trans(\nu)$ of computation nodes. It remains to show that for all computation nodes $\nu$ in $\Pi$ and for the attacker strategy $V_{In}$ all nodes in $t_c$ take the yes-branch if and only if $\nu$ takes the yes-branch against $parse(V_{In})$. Towards contradiction, let $\nu$ be the first computation node such that $\nu$ branches differently than $t_c$; w.l.o.g. assume $\nu$ goes into the yes-branch and $t_c$ go into a no-branch.

- For Case 1 (commitment) in the construction of *trans*, the only difference is that the CRS that was initially created is included, hence this case cannot occur.
- For Case 7 (pair) to case 9 (signature), *trans* solely descends recursively, and for case 10 (rest) nothing is changed, hence this case cannot occur.
- For Case 2 (ZK), an empty proof tree is constructed with the randomness tree that is reconstructed (via *tree*). Both constructors, $\mathsf{zkp}$ and ZK, fail for the same inputs, hence this case cannot occur.
- For Case 3 (verification), the tree $t_c$ goes into a no-branch if either setPub fails or $\text{ver}_{\text{zk}}$ fails. The computation node setPub fails if $trans(z)$ is not a ZK proof or for some $j \in \{1, \cdots, n\}$ $trans(c_j) = \bot$. In the latter case, $\mathsf{verzk}^{\text{AWOT}}$ and $t_c$ take both the no-branch, since for $\mathsf{verzk}^{\text{AWOT}}$ the corresponding $c_j = \bot$ as well ($\nu$ is by assumption the first computation node that branches differently). In the first case, $trans(z)$ is not of the form $\text{ZK}(t_1, t_2, t_3)$. By the definition of *trans* $z$ is not of the form $\mathsf{zkp}((r_j)_{j=0}^n)$. Thus, $\mathsf{verzk}^{\text{AWOT}}$ fails as well. Now, assume that $\text{ver}_{\text{zk}}$ fails, thus $\text{check}_{\text{zk}}(\text{crs}(\tilde{r}), \text{setPub}(trans(z), tree((trans(c_j))_{j=1}^n, \varepsilon), randtree(z))$ fails. Since setPub does not fail and $\text{crs}(\tilde{r})$ is well-formed, $\text{check}_{\text{zk}}$ has to fail while checking the validity of the commitments and whether they match the randomness tree (see Section A.2.2). In these cases, however, we know that $\mathsf{verzk}^{\text{AWOT}}$ fails as well.
- For Case 4 (hiding), the tree $t_c$ goes into a no-branch if either setPub. As in Case 3, since $\nu$ is the first node where the branching does not coincide, setPub only fails if $(\mathsf{hide}^i(z, (c_j)_{j=1}^n, (o_j)_{j=1}^n))$ fails.

- For Case 5 (protocol re-randomization) and Case 6 (attacker re-randomization), the tree $t_c$ goes into a no-branch if either setPub or $\text{rer}_{\text{zk}}$ fails. By the same arguments as in Case 3, setPub does not fail (i.e., go into a no-branch). Assume $\text{rer}_{\text{zk}}$ fails, thus one of the rerPr applications fails. The rerPr only fail if the statement tree or the corresponding randomness trees do not coincide. Since the statement (and thus also the randomness trees) are the same for all proofs and $tree(trans(c_j)_{j=1}^n)$ construct this statement, the invocations of rerPr do not fail.

□

**Claim 5.2.** *For control, output, and input nodes there is a one-to-one correspondence between* $\Pi$ *and* $trans(\Pi)$.

*Proof.* The mapping *trans* maps computation nodes to subtrees that consists solely of computation nodes. The rest remains unchanged. Moreover, Claim 5.1 implies that the control flow of $\Pi_b$ and $trans(\Pi)_b$ matches for $b \in \{\text{left}, \text{right}\}$. Hence, every control, output, and input node in $\Pi$ is mapped to exactly one node in $trans(\Pi)$. □

It remains to show that the knowledge of both views is the same. For this proof, we generalize the translation *trans* to attacker strategies.

**Claim 5.3.** $V_{left} \approx_s V_{right} \implies trans(V_{left}) \approx_s trans(V_{right})$.

*Proof.* Towards contradiction, assume that $V_{\text{left}} \approx_s V_{\text{right}}$ but $trans(V_{\text{left}}) \not\approx_s trans(V_{\text{right}})$. Then, there is an attacker strategy $V_{In}$ such that ..

Observe that for every output node in $\Pi$ there is a unique output node in $trans(\Pi)$ since *trans* solely replaces some computation nodes with subtrees that purely consist of computation nodes, in particular no output nodes. By Claim 5.2, for each output node $\nu$ there is a unique output node $trans(\nu)$ in $trans(\Pi)$. Let $t$ be the term sent by $\nu$ against the attacker strategy $V_{In}$ in the symbolic execution of $\Pi_b$. By the construction of *trans*, we know that for the term $t'$ sent by $trans(\nu)$ against $trans(V_{In})$ in the symbolic execution of $trans(\Pi)_b$ the following hods: $trans(t) = t'$. Since $trans(\Pi)_{\text{left}} \not\approx trans(\Pi)_{\text{right}}$, there is a so-called *distinguishable* output node $\nu'$ and two terms $t'_{\text{left}}$ and $t'_{\text{right}}$ that are sent on $\nu'$ against the attacker strategy $V_{In}$ such that there is a so-called *distinguishing* symbolic operation $O'$ with

$$\text{eval}_{O'}(\hat{V}_{\text{left}}) \neq \text{eval}_{O'}(\hat{V}_{\text{right}}),$$

where $\hat{V}_b$ is the prefix of the symbolic view that corresponds to the respective symbolic execution (of $trans(\Pi)_{\text{left}}$ or $trans(\Pi)_{\text{right}}$). Let $\nu'$ be the first such distinguishable output node. Then, there is a corresponding node $\nu := trans^{-1}(\nu')$ in $\Pi$. However, the node $\nu$ is not distinguishing (against $V_{In}$) since $V_{\text{left}} \approx_s V_{\text{right}}$.

As a next step, we will show that for each distinguishing symbolic operation $O'$ for $trans(\Pi)$ there is a distinguishing symbolic operation $O$ for $\Pi$, which contradicts the assumption that $V_{\text{left}} \approx_s V_{\text{right}}$ and thereby proves the claim. We define a function *parse* that maps a distinguishing symbolic operation $O'$ in the CMZK model to a distinguishing symbolic operation $O$ in the simpler model. We use three functions: $listof$, $witnesslistof$ and $unveillistof$. The first function, $listof$, converts a statement tree into a list of commitments. Whenever, it encounters a projection, i.e., a reference to a protocol message, it keeps that reference in the list. The second function, $witnesslistof$, works as $listof$ with the exception that it converts the randomness tree into a list of randomness symbols. The third function, $unveillistof$, works as $listof$ with the exception that it transforms the statement tree into a list of unveil information, i.e., message-randomness pairs or $\varepsilon$ symbols. (The first matching rule is taken.)

1. $parse(\text{com}(O_1, O_2, O_3)) = parse(\text{COM}(parse(O_2), parse(O_3)))$
2. $parse(\text{uv}(O_1, O_2)) = \text{pair}(parse(O_1), parse(O_2))$
3. $parse(\text{open}(O_1, O_2, \text{uv}(O_3))) = \text{open}(parse(O_2), parse(O_3))$
4. $parse(\text{crsof}(O_1) = \text{iscom}(parse(O_1))$
5. $parse(\text{ZK}(O_1, O_2, O_3)) = (\mathsf{zkp}(parse(witnesslistof(O_1)) :: parse(O_3)), parse(listof(O_2)), parse(unveillistof(O_2)))$
6. $parse(\text{ver}_{\text{zk}}(O_1, O_2)) = \mathsf{verzk}^{\text{AWOT}}(parse(O_2), parse(listof(O_2)))$

7. $parse(\mathrm{rer}_{\mathrm{zk}}(O_1, O_2, O_3, O_4)) = O$ where $O$ is constructed as follows: $O$ is the sequence of symbolic operations of the form $\mathsf{rerand}^i(parse(O_2), parse(listof(O_2)), parse(unveillistof(O_2)), parse(O_3))$ for every $i$ for which the randomness of the $i$th commitment of the AWOT statement is re-randomized in $O_2$.

8. $parse(\mathrm{attRer}_{\mathrm{zk}}(O_1, O_2, O_3)) = O$ where $O$ is defined as above except that $\mathsf{att\text{-}rerand}^i$ is used instead of $\mathsf{rerand}^i$

9. $parse(\mathrm{setPub}(O_1, O_2, O_3)) = \mathsf{rerand}^0(\hat{O}, O_3)$ where $\hat{O}$ is constructed as follows: first, compute $\hat{O}' := parse(O_1)$, then replace the second argument of $\hat{O}'$ (the commitments) with $parse(listof(O_2))$.

10. $parse(\mathrm{setPub}(O_1, O_2)) = O$ where $O$ is constructed as follows: first, compute $\hat{O} := parse(O_1)$, then replace the second argument of $\hat{O}$ (the commitments) with $parse(listof(O_2))$.

11. $parse(\mathrm{getPub}(O_1) = O$ where $O$ is the second argument of $parse(O_1)$

12. $parse(C(O_1, \ldots, O_m)) = C(parse(O_1), \ldots, parse(O_m))$ for a constructor or a destructor $C \notin \{\mathrm{com}, \mathrm{uv}, \mathrm{open}, \mathrm{crs}, \mathrm{crsof}, \mathrm{ZK}, \mathrm{ver}_{\mathrm{zk}}, \mathrm{rer}_{\mathrm{zk}}, \mathrm{attRer}_{\mathrm{zk}}, \mathrm{setPub}, \mathrm{getPub}\}$

13. $parse(c) = c$ if $c \in \{\varepsilon, \mathrm{O}\}$ or an attacker nonce.

14. $parse(\mathrm{crs}(O_1) = \varepsilon$

It remains to show that $O := parse(O')$ is distinguishing whenever $O'$ is distinguishing. This can be shown by a case distinction similar to the proof of Claim 5.1. We will briefly discuss the cases. In Case 12 and Case 13 the parsing function does not change anything. In Case 1 to Case 5, merely the symbolic encoding is changed and the CRS is removed, since the simpler model only assumes one CRS. In Case 6, $parse(O')$ transforms the representation of the statement and then checks whether all arguments coincide, just as $O'$. In Case 7 and Case 8, the re-randomization fails in both cases if the terms are not well-formed and otherwise succeeds. Both destructors replace randomness symbols. They merely do the replacement for different encodings. In Case 9 and Case 10, changing the public part is translated to changing the second part of the triple that constitutes a ZK proof (and an additional rerandomization in Case 9). In Case 11, we first parse the ZK proof and then retrieve the second argument, i.e., the commitments. All tests that can be performed on the statement can (in this restricted scenario with only one statement) be performed on the commitments. Finally, there is one special case. In Case 14, *parse* parses a CRS term to the empty string $\varepsilon$. Even though this is inaccurate, this case only occurs if the overall symbolic operation $O'$ equals $\mathrm{crs}(O_1)$. But then, $O'$ cannot be not distinguishing; hence this case does not occur. □

Claim 5.1 and Claim 5.3 entail the statement. □

**Theorem 5.** *[CS w.r.t. uniformity of bi-protocols] Let A be an implementation that satisfies the conditions from Section 4.2. Then, A is computationally sound w.r.t. uniformity of CoSP bi-protocols for the class of MZK-safe CoSP bi-protocols.*

*Proof.* The theorem directly follows from Lemma 5 and from Theorem 8. □

# E Complete proof of computational soundness

The computational soundness proof of our symbolic model first presents a simulator Sim (Section E.2) and then shows that this simulator satisfies Dolev-Yaoness (Section E.3) and indistinguishability (Section E.3), as defined in Section 3.

**Symbolic and computation ZK relation.** Symbolically, we have the normal relation $R_{\mathrm{hon}}^{\mathrm{sym}}$ for protocols

$$\big\{(x, w) \mid \; L'(x)\{w_i/x_{w_i}\} \text{ is true, where } x = \mathrm{extrSta}(t), w = \mathrm{extrWit}(t))\big\}$$

$$
\begin{aligned}
L''(S_1 \vee S_2) &:= L''(S_1) \vee L''(S_2) \\
L''(S_1 \wedge S_2) &:= L''(S_1) \wedge L''(S_2) \\
L''(A = B) &:= L''(A) = L''(B) \\
L''(A \neq B) &:= L''(A) \neq L''(B) \\
L''(\mathrm{enc}\langle X_1, X_2, X_n\rangle) &:= \exists r.F(L''(X_1), X_2, r) \\
L''(F\langle X_1, \ldots, X_n\rangle) &:= F(L''(X_1), \ldots, L''(X_n)) \\
n &= \mathrm{arity}(F) \\
L''(\langle \mathrm{com}(\mathrm{crs}, m, r), \varepsilon\rangle) &:= x_m \text{ , for } x_m \in V \\
L''(\langle \mathrm{com}(\mathrm{crs}, m, r), \mathrm{uv}(m, r)\rangle) &:= m
\end{aligned}
$$

Since parsing the correct randomness from attacker generated encryptions is in general not possible, we have to allow the attacker to use any randomness. Hence, we relax the symbolic ZK relation for attacker messages in the ZK relation $R_{\mathrm{adv}}^{\mathrm{sym}}$. We define the following relaxed logical formulas:

$$
\begin{aligned}
L''(S_1 \vee S_2) &:= L''(S_1) \vee L''(S_2) \\
L''(S_1 \wedge S_2) &:= L''(S_1) \wedge L''(S_2) \\
L''(A = B) &:= L''(A) = L''(B) \\
L''(A \neq B) &:= L''(A) \neq L''(B) \\
L''(\mathrm{enc}\langle X_1, X_2, X_n\rangle) &:= \exists r.F(L''(X_1), X_2, r) \\
L''(F\langle X_1, \ldots, X_n\rangle) &:= F(L''(X_1), \ldots, L''(X_n)) \\
n &= \mathrm{arity}(F) \\
L''(\langle \mathrm{com}(\mathrm{crs}, m, r), \varepsilon\rangle) &:= x_m \text{ , for } x_m \in V \\
L''(\langle \mathrm{com}(\mathrm{crs}, m, r), \mathrm{uv}(m, r)\rangle) &:= m
\end{aligned}
$$

Then, $R_{\mathrm{adv}}^{\mathrm{sym}}$ is defined as follows:

$$
\big\{(x, w) \mid \; L''(x)\{w_i/x_{w_i}\} \text{ is true, where } x = \mathrm{extrSta}(t), w = \mathrm{extrWit}(t))\big\}
$$

Computationally, recall $R_{\mathrm{hon}}^{\mathrm{comp}}$ is defined as follows:

$$
\big\{(\mathrm{img}_\eta(x), \mathrm{img}_\eta(w)) \mid \; (x, w) \in R_{\mathrm{hon}}^{\mathrm{sym}} \text{ for a consistent } \eta\big\}
$$

We stress that our definition is equivalent to the previous definition.

Analogously, $R_{\mathrm{adv}}^{\mathrm{comp}}$ is defined using $R_{\mathrm{hon}}^{\mathrm{sym}}$:

$$
\big\{(\mathrm{img}_\eta(x), \mathrm{img}_\eta(w)) \mid \; (x, w) \in R_{\mathrm{adv}}^{\mathrm{sym}} \text{ for a consistent } \eta\big\}
$$

By construction Definition 6 obviously holds.

## E.1 Transparent hybrid executions

In the presence of zero-knowledge transformations it might happen that the simulator parses a zero-knowledge proof from the attacker, further transforms this proof, and sends this transformed proof back to the attacker. Then, the simulator in interaction with the hybrid execution would need to construct a proof even though some witnesses are only known to the attacker. We remedy this problem by directly applying the implementations of the zero-knowledge transformations that lead to this term.

In order to be able to determine which zero-knowledge transformations have been applied to a term, we modify the hybrid execution to a so-called *transparent hybrid execution* that is defined as follows: the result of a destructor node is in the yes-branch instead of $f(\underline{t})$ the term $\hat{f}(\underline{t})$, where for every destructor $f$, we introduce a free constructor $\hat{f}$. We define the set of *extended terms* $\mathbf{T}'$ as the set of terms $\mathbf{T}$ that may also contain the constructors $\hat{f}$ for $f \in \mathbf{D}$. Moreover, instead of checking at a destructor node whether $f(\underline{t})$ outputs $\bot$ or another term, the symbolic execution first needs to get rid of the $\hat{f}$ constructors, by evaluating each $t_i$: $\mathrm{eval}'\hat{f}(\underline{s}) := f(\underline{\mathrm{eval}'s})$ and $\mathrm{eval}'t := t$ otherwise. With this transparent hybrid execution the simulator is able to track the symbolically applied transformations and thereby to compute $\beta(\hat{f}(\underline{t}))$ as $A_{\hat{f}}(\underline{\beta(t)})$.

We generalize this idea and let for a set of destructors $\mathbf{D}_t$ the transparent hybrid execution only send terms for which every destructor application $f(\underline{t})$ with $f \in \mathbf{D}_t$ has been replaced by a distinguished constructor application $\hat{f}(\underline{t})$. For the definition of a transparent hybrid execution, we use the evaluation function $\mathrm{eval}''$, where $\mathbf{T}''$ denotes the extended set of terms in which for each destructor $d$ a constructor $\hat{d}$ has been introduced. We define $\mathrm{eval}_d$ as in the CoSP paper: for $d \in \mathbf{C} \cup \{\hat{g} \mid g \in \mathbf{D}_t\}$ we have $\mathrm{eval}_d(\underline{t}) := d(\underline{t})$ if $d(\underline{t}) \in \mathbf{T}$, otherwise $d(\underline{t}) := \bot$; for $d \in \mathbf{D} \setminus \mathbf{D}_t$ we have $\mathrm{eval}_d(\underline{t}) := d(\underline{t})$ if $d(\underline{t}) \neq \bot$, otherwise $d(\underline{t}) := \bot$.

Recall that our symbolic abstraction explicitly models the commitments on the witnesses. Compared to previous work, we therefore need to be able to determine whether a commitment has been reused in an MZK proof. However, unless we assume extraction zero-knowledge we cannot use the extractor on simulated MZK proofs. As a consequence, we allow the simulator Sim to not only send terms but also variables and refine the assignment $f$ from nodes to terms or variables upon every destructor application of the transparent hybrid execution. Let $X$ be the set of variables. We define the set of terms as $\mathbf{T}'$ as $\mathbf{T}'' \cup X \cup \{\bot\}$. Moreover, we allow the simulator to send an assignment $\alpha : X \to \mathbf{T}'$ in order to refine the interpretation of previously sent terms. In detail, upon each destructor application the transparent hybrid execution queries the simulator Sim with (`update`) for a refinement $\alpha$ of the terms. Our destructors are defined by a set of conditional rewriting rules; hence we can naturally extend $\mathrm{eval}''d(\underline{t})$ with $\underline{t} \in \mathbf{T}''^{|\underline{t}|}$ to $\mathrm{eval}'d(\underline{t})$ with $\underline{t} \in \mathbf{T}'^{|\underline{t}|}$ where all variables $x$ in $t$ are treated as attacker nonces $n_x \in \mathbf{N}_E$.

We define the following function:

$$\begin{aligned}\mathrm{saturate}(f,\alpha) &:= \lim_{i\to\infty} g_i \text{ where}\\ g_0 := f \text{ and } g_i &:= \big\{(n,t) \mid \exists C, \underline{y}.C[\underline{y}] = g_i(n)\\ &\qquad \wedge t = C[\alpha(y_1),\ldots,\alpha(y_n)]\big\}\end{aligned}$$

We stress that saturate is poly-time computable if $\alpha$ is the identity function up to polynomially many variables (which is how we use it in our proof).

**Definition 19** (Transparent hybrid execution)**.** *Let $\Pi_{\mathsf{p}}$ be a probabilistic CoSP protocol, and let Sim be a simulator. We define a probability distribution TH-Trace$_{\boldsymbol{M},\boldsymbol{D}_t,\Pi_{\mathsf{p}},Sim}(k)$ on (finite) lists of tuples $(S_i, \nu_i, f_i)$ called the* full $\mathbf{D}_t$-transparent hybrid trace *according to the following probabilistic algorithm $\Pi^T$, run on input $k$, that interacts with Sim. ($\Pi^T$ is called the transparent hybrid protocol machine associated with $\Pi_{\mathsf{p}}$ and internally runs a symbolic simulation of $\Pi_{\mathsf{p}}$ as follows:)*

- Start*: $S_1 := S := \varnothing$, $\nu_1 := \nu$ is the root of $\Pi^T$, and $f_1 := f$ is a totally undefined partial function mapping node identifiers to $\boldsymbol{T}$.*
- Transition*: For $i = 2, 3, \ldots$ do the following:*
  - *Let $\underline{\tilde{\nu}}$ be the node identifiers in the label of $\nu$. Define $\underline{\tilde{t}}$ through $\tilde{t}_j := f(\tilde{\nu}_j)$.*
  - *Proceed depending on the type of $\nu$:*
    - *If $\nu$ is a computation node with destructor $d$, then send (`update`) to Sim. Wait for a variable assignment $\alpha : X \to \boldsymbol{T}'$ as a response. Let $id_S$ be the identity function of a set $S$. Update $f := saturate(f, \alpha)$ with $\alpha$. Let $m := eval'_d(\underline{\tilde{t}})$. If $eval'd(\underline{\tilde{t}}) \neq \bot$, let $\nu'$ be the yes-successor of $\nu$ and let $f' := f(\nu := m)$. If $eval'f(\underline{\tilde{t}}) = \bot$, let $\nu'$ be the no-successor of $\nu$ and let $f' := f$. Set $f := f'$ and $\nu := \nu'$.*
    - *If $\nu$ is a computation node with constructor, destructor, or nonce $d$, then let $m := eval_d(\underline{\tilde{t}})$. If $eval'd(\underline{\tilde{t}}) \neq \bot$, let $\nu'$ be the yes-successor of $\nu$ and let $f' := f(\nu := m)$. If $eval'f(\underline{\tilde{t}}) = \bot$, let $\nu'$ be the no-successor of $\nu$ and let $f' := f$. Set $f := f'$ and $\nu := \nu'$.*
    - *If $\nu$ is an output node, send $\tilde{t}_1$ to Sim (but without handing over control to Sim). Let $\nu'$ be the unique successor of $\nu$ and let $S' := S \cup \{\tilde{t}_1\}$. Set $\nu := \nu'$ and $S := S'$.*
    - *If $\nu$ is an input node, hand control to Sim, and wait to receive $m \in \boldsymbol{T}$ from Sim. Let $f' := f(\nu := m)$, and let $\nu'$ be the unique successor of $\nu$. Set $f := f'$ and $\nu := \nu'$.*

    - *If $\nu$ is a control node labeled with out-metadata $l$, send $l$ to Sim, hand control to Sim, and wait to receive a bitstring $l'$ from Sim. Let $\nu'$ be the successor of $\nu$ along the edge labeled $l'$ (or the lexicographically smallest edge if there is no edge with label $l'$). Set $\nu := \nu'$.*
    - *If $\nu$ is a nondeterministic node, sample $\nu'$ according to the probability distribution specified in $\nu$. Set $\nu := \nu'$.*
  - *Send $(info, \nu, t)$ to Sim. When receiving an answer (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 $\text{TH-Nodes}'_{\boldsymbol{M},\boldsymbol{D}_t,\Pi_\mathsf{p},Sim}(k)$. By replacing every variable by distinct fresh nonce from $\boldsymbol{N}_E$, and by replacing $com(c, \bot, N)$ by $garbCom(c, N)$, we get (the distribution of traces) $\text{TH-Nodes}_{\boldsymbol{M},\boldsymbol{D}_t,\Pi_\mathsf{p},Sim}(k)$. We call this distribution the* $\mathbf{D}_t$-transparent hybrid node trace. *In abuse of notation, we often omit the adjective transparent since we only consider transparent hybrid executions and traces.*

We stress that the definition of a good simulator can be extended to the transparent hybrid execution.

Analogous to the proof of the CoSP paper which shows that the existence of a good simulator implies computational soundness, it suffices to show the lemma about the hybrid node traces.

**Lemma 6.** *Consider a transparent hybrid execution of Sim+$\Pi^T$ in which Sim is DY, i.e., we have $\{\boldsymbol{T}_1, \ldots, \boldsymbol{T}_\ell\} \vdash m_\ell$ for all $\boldsymbol{T}_i$ and $m_\ell$ as in the definition of the symbolic execution in CoSP [BHU09] and all $\ell$.*

*Let $tr$ be the full hybrid trace of that execution. Then $tr$ is a full symbolic trace of $\Pi_\mathsf{s}$.*

*Proof.* We show that $tr$ fulfills the conditions on full traces of the definition of the symbolic execution from the CoSP paper. [BHU09] This is clear for constructor, and control nodes, since the processing of these nodes in the hybrid setting of Definition 19 matches the one in the symbolic setting of the definition of symbolic execution in CoSP. For destructor nodes we have for all $\underline{t} \in \mathbf{T}^n$ $\text{eval}\hat{f}(\underline{t}) = \bot \Leftrightarrow \text{eval}_f(\underline{t}) = \bot$. Consequently, the same branches are taken in both settings.

Input/output nodes in $tr$ consist of an extended term $t \in \mathbf{T}'$ sent from $\Pi^T$ to Sim, or an extended term $t'$ sent from Sim to $\Pi^T$. By the DY property of Sim, we know that $S \vdash \text{eval}'t'$, where $S$ denotes all terms (including $t$) sent from $\Pi^T$ to Sim so far. Hence, the node satisfies the requirement for input/output nodes from the definition of symbolic execution in CoSP. This completes the proof of the lemma. □

**Lemma 7** (Good simulator implies soundness)**.** *Let $\mathbf{M} = (\boldsymbol{C}, \boldsymbol{N}, \boldsymbol{T}, \boldsymbol{D}, \vdash)$ be a symbolic model, let $\boldsymbol{D}_t \subseteq \boldsymbol{D}$ be a set of destructors,let $P$ be a class of CoSP protocols, and let $A$ be a computational implementation of $\boldsymbol{M}$. Assume that for every efficient probabilistic CoSP protocol $\Pi_\mathsf{p}$ (whose corresponding CoSP protocol is in $P$), every probabilistic polynomial-time adversary $\mathcal{A}$, and every polynomial $p$, there exists a good simulator against the $\boldsymbol{D}_t$-transparent hybrid execution for $\boldsymbol{M}$, $\Pi_\mathsf{p}$, $A$, $\mathcal{A}$, and $p$. Then $A$ is computationally sound for all protocols in $P$.*

## E.2 The simulator Sim

In the definition of the simulator we needed two functions $\beta$ and $\tau$ to translate between the symbolic and computational worlds. Note, we denote $A_{\text{mkZK}}$ by $A_{\text{ZK}}$. Here are their full definitions:

**The partial function $\beta : \mathbf{T} \to \{0,1\}^*$.**

- $\beta(N) \coloneqq r_N$ if $N \in \mathbf{N}_P$
- $\beta(N^m) \coloneqq m$
- $\beta(\text{enc}(\text{ek}(N), t, M)) \coloneqq A_{\text{enc}}(\beta(\text{ek}(N)), \beta(t), r_M)$ if $M \in \mathbf{N}_P$
- $\beta(\text{enc}(\text{ek}(M), t, N^m)) \coloneqq m$ if $M \in \mathbf{N}_P$
- $\beta(\text{ek}(N)) \coloneqq A_{\text{ek}}(r_N)$ if $N \in \mathbf{N}_P$
- $\beta(\text{ek}(N^m)) \coloneqq m$
- $\beta(\text{dk}(N)) \coloneqq A_{\text{dk}}(r_N)$ if $N \in \mathbf{N}_P$
- $\beta(\text{sig}(\text{sk}(N), t, M)) \coloneqq A_{\text{sig}}(A_{\text{sk}}(r_N), \beta(t), r_M)$ if $N, M \in \mathbf{N}_P$
- $\beta(\text{sig}(\text{sk}(M), t, N^s)) \coloneqq s$
- $\beta(\text{vk}(N)) \coloneqq A_{\text{vk}}(r_N)$ if $N \in \mathbf{N}_P$
- $\beta(\text{vk}(N^m)) \coloneqq m$
- $\beta(\text{sk}(N)) \coloneqq A_{\text{sk}}(r_N)$ if $N \in \mathbf{N}_P$

- $\beta(\mathrm{crs}(N)) \coloneqq A_{\mathrm{crs}}(r_N)$ if $N \in \mathbf{N}_P$
- $\beta(\mathrm{crs}(N^c)) \coloneqq c$
- $\beta(\mathrm{ZK}(t_1, t_2, N_2)) \coloneqq$
  $A_{\mathrm{ZK}}(\beta(\mathrm{crsof}(t_1)), \beta(\mathrm{extrSta}(t_1)), \beta(\mathrm{extrWit}(t_1)), r_{N_2})$ if $N_2 \in \mathbf{N}_P$
- $\beta(\mathrm{ZK}(t_1 \wedge t_1', t_2 \wedge t_2', \mathrm{rand}(N, N'))) \coloneqq$
  $A_{\mathrm{and_{ZK}}}(\beta(\mathrm{ZK}(t_1, t_2, N)), \beta(\mathrm{ZK}(t_1', t_2', N')))$ if $N \in \mathbf{N}_E$ or $N' \in \mathbf{N}_E$
- $\beta(\mathrm{ZK}(t_1, t_2, N^s)) \coloneqq s$
- $\beta(\hat{f}(t_1, \ldots, t_n)) \coloneqq A_f(\beta(t_1), \ldots, \beta(t_n))$ if $f \in \hat{F}$
- $\beta(\mathrm{com}(\mathrm{crs}(N), m, M)) \coloneqq$
  $A_{\mathrm{com}}(\beta(\mathrm{crs}(N)), \beta(m), r_M)$
- $\beta(\mathrm{pair}(t_1, t_2)) \coloneqq A_{\mathrm{pair}}(\beta(t_1), \beta(t_2))$
- $\beta(\mathrm{string}_0(t)) \coloneqq A_{\mathrm{string}_0}(\beta(t))$
- $\beta(\mathrm{string}_1(t)) \coloneqq A_{\mathrm{string}_1}(\beta(t))$
- $\beta(\varepsilon) \coloneqq A_\varepsilon()$
- $\beta(\mathrm{garb}(N^c)) \coloneqq c$
- $\beta(\mathrm{garbCom}(t, N^c)) \coloneqq c$
- $\beta(\mathrm{garbEnc}(t, N^c)) \coloneqq c$
- $\beta(\mathrm{garbSig}(t, N^s)) \coloneqq s$
- $\beta(t) \coloneqq \bot$ if no case matches

**The total function $\tau : \{0,1\}^* \to \mathbf{T}$.** (by taking the first matching rule)

- $\tau(r) \coloneqq N$ if $r = r_N$ for some $N \in \mathbf{N}_P$ and $N$ occurred in a term sent from $\Pi^C$
- $\tau(r) \coloneqq N^r$ if $r$ is of type nonce
- $\tau(c) \coloneqq \mathrm{enc}(\mathrm{ek}(M), t, N)$ if $c$ has earlier been output by $\beta(\mathrm{enc}(\mathrm{ek}(M), t, N))$ for some $M \in \mathbf{N}, N \in \mathbf{N}_P$
- $\tau(c) \coloneqq \mathrm{enc}(\mathrm{ek}(M), \tau(m), N^c)$ if $c$ is of type ciphertext and $\tau(A_{\mathrm{ekof}}(c)) = \mathrm{ek}(M)$ for some $M \in \mathbf{N}_P$ and $m \coloneqq A_{\mathrm{dec}}(A_{\mathrm{dk}}(r_N), c) \neq \bot$
- $\tau(c) \coloneqq \mathrm{garbEnc}(\tau(A_{\mathrm{ekof}}(c)), N^c)$ if $c$ is of type encryption
- $\tau(c) \coloneqq \mathrm{ek}(N)$ if $c$ has earlier been output by $\beta(\mathrm{ek}(N))$ for some $N \in \mathbf{N}_P$
- $\tau(c) \coloneqq \mathrm{ek}(N^c)$ if $c$ is of type encryption key
- $\tau(s) \coloneqq \mathrm{sig}(\mathrm{sk}(M), t, N)$ if $s$ has earlier been output by $\beta(\mathrm{sig}(\mathrm{sk}(M), t, N))$ for some $M, N \in \mathbf{N}_P$
- $\tau(s) \coloneqq \mathrm{sig}(\mathrm{sk}(M), \tau(m), N^s)$ if $s$ is of type signature and $\tau(A_{\mathrm{vkof}}(s)) = \mathrm{vk}(M)$ for some $M \in \mathbf{N}$ and $m \coloneqq A_{\mathrm{ver_{sig}}}(A_{\mathrm{vkof}}(s), s) \neq \bot$
- $\tau(s) \coloneqq \mathrm{garbSig}(\tau(A_{\mathrm{vkof}}(s)), N^s)$ if $s$ is of type signature
- $\tau(s) \coloneqq \mathrm{vk}(N)$ if $s$ has earlier been output by $\beta(\mathrm{vk}(N))$ for some $N \in \mathbf{N}_P$
- $\tau(s) \coloneqq \mathrm{vk}(N^s)$ if $s$ is of type verification key.
- $\tau(z) \coloneqq \mathrm{crs}(N)$ if $z$ has earlier been output by $\beta(\mathrm{crs}(N))$ for some $N \in \mathbf{N}_P$.
- $\tau(z) \coloneqq \mathrm{crs}(N^z)$ if $z$ is of type crs.
- $\tau(z) \coloneqq \mathrm{com}(c, m, N)$ if $z$ is of type commitment and $z$ has been output earlier by $\beta(\mathrm{com}(c, m, N))$ for some $N \in \mathbf{N}_P$.
- $\tau(z) \coloneqq \mathrm{com}(\tau(c), x_z, N^z)$ if $z$ is of type commitment and where $c \coloneqq A_{\mathrm{crsof}}(z)$ and $x_z \in X$ is a variable.
- $\tau(z) \coloneqq \hat{f}(\overline{s})$ if $z$ is of type zero-knowledge and $z$ has been output earlier by $\beta(\hat{f}(\overline{s}))$ for some $\overline{s}$
- $\tau(z) \coloneqq \mathrm{ZK}(t_1, t_2, N_2)$ if $z$ is of type zero-knowledge proof and $z$ has been output earlier by $\beta(\mathrm{ZK}(t_1, t_2, N_2))$ for some $N_2 \in \mathbf{N}_P$.
- $\tau(z) \coloneqq \mathrm{ZK}(t_1, t_2, N^z)$ if $z$ is of type zero-knowledge proof, $A_{\mathrm{ver_{zk}}}(A_{\mathrm{crsof}}(z), z) = 1$, $m_{t_1} \coloneqq A_{\mathrm{getPub}}(z) \neq \bot$, $t_1 \coloneqq C[x_1, \ldots, x_n] \coloneqq \tau(m_{t_1}) \neq \bot$, $\mathrm{crs}(N) = \tau(A_{\mathrm{crsof}}(m_{t_1}))$ for some $N \in \mathbf{N}_P$, and $\alpha' \coloneqq$ $\mathbf{SymbExtr}(S, C[\alpha(x_1), \ldots, \alpha(x_n)])$ where $S$ is the set of terms sent to the adversary before and $t_2 \coloneqq$ randomnessTree$(t_1)$, where randomnessTree$(t_1)$ is defined by replacing $\langle \mathrm{com}(\mathrm{crs}, t, r), U \rangle$ by $r$ (in the FMZK variant), or $\langle t, r \rangle$ (in the CMZK) variant.
  If $\bot \in \alpha'(\mathrm{extrWit}(t_1))$, we say an *extraction-failure* on $(N, z, m_{t_1})$ occurred, see below for the behavior of Sim in this case. Otherwise, set $\alpha \coloneqq \alpha' \circ \alpha$.[17]
- $\tau(z) \coloneqq \mathrm{and_{ZK}}(\tau(z'), \tau(z''))$ if $z$ is of type zero-knowledge proof and $A_{\mathrm{splitAnd}}(z) = (z', z'') \neq \bot$, $A_{\mathrm{ver_{zk}}}(A_{\mathrm{crsof}}(z), z) = 0$.
- $\tau(z) \coloneqq \mathrm{ZK}(t_1, t_2, N^z)$ if $z$ is of type zero-knowledge proof, $c \coloneqq \tau(A_{\mathrm{crsof}}(z))$ and $x \coloneqq \tau(A_{\mathrm{getPub}}(z))$, where $t_2 \coloneqq$ randomnessTree$(t_1)$. Moreover, set $\alpha'(x) \coloneqq \bot$ for each $x \in \mathit{extractWitness}(t_1)$ and $\alpha \coloneqq \alpha' \circ \alpha$.

[17] Formally, this is defined as $\{(x, C[C_1'[\underline{z}_1], \ldots, C_n'[\underline{z}_n]]) \mid C[\underline{y}] = \alpha(x) \wedge C'[\underline{z}] = \alpha'(y)\}$.

- $\tau(m) \coloneqq \text{pair}(\tau(A_{\text{fst}}(m)), \tau(A_{\text{snd}}(m)))$ if $m$ is of type pair
- $\tau(m) \coloneqq \text{string}_0(\tau(m'))$ if $m$ is of type payload-string and $m' \coloneqq A_{\text{unstring}_0}(m) \neq \bot$
- $\tau(m) \coloneqq \text{string}_1(\tau(m'))$ if $m$ is of type payload-string and $m' \coloneqq A_{\text{unstring}_1}(m) \neq \bot$
- $\tau(m) \coloneqq \varepsilon$ if $m$ is of type payload-string and $m = A_\varepsilon()$
- $\tau(m) \coloneqq \text{garb}(N^m)$ otherwise

When an extraction-failure on $(N, z, m_{t_1})$ occurs (i.e., when in the computation of $\tau$, $\alpha' = \mathbf{SymbExtr}(z, t_1, \alpha)$, 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 $m_w := \mathbf{E}(m_{t_1}, z, \text{extd})$. If $(m_{t_1}, m_w) \notin \mathcal{R}$, we say a *ZK-break* occurred. If no ZK-break occurred but $(\tau(m_{t_1}), \tau(m_w)) \notin \mathcal{R}$, we call this a *ZK-flop*. Then (no matter whether a ZK-break or ZK-flop occurred), the simulator aborts.

Upon a message (`update`) by the transparent hybrid execution, Sim responds with its current witness variable assignment $\alpha : X \to \mathbf{T}'$, and sets $\alpha := \emptyset$.

We define $\tau^*$ by the same case distinction as $\tau$ 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 by $\tau^*$ parsed as a proof that contains garbage commitments. Thus, by definition, there is no extraction failure during a computation of $\tau^*$.

**The faking simulator $\text{Sim}_f$.** In the transformations from Sim over $\text{Sim}_i$ ($i = 1$ to 5) to $\text{Sim}_f$ all invocations of cryptographic algorithms (such as $A_{\text{ZK}}$ and $A_{\text{enc}}$) are replaced by oracles and thereafter by faking oracles.

- We define $\text{Sim}_1$ like Sim but we change $\beta$ to use zero-knowledge oracles instead of computing $A_{\text{crs}}$ and $A_{\text{ZK}}$. 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 (`crs`)-query, it returns crs, and upon a (`prove`, $x, w$)-query, it returns $\mathbf{P}(x, w, \text{crs})$ if $(x, w) \in R_{\text{hon}}^{\text{comp}}$ and $\bot$ otherwise. Upon a (`extd`)-query, it returns extd. For each $N \in \mathbf{N}_P$, $\text{Sim}_1$ maintains an instance $\mathcal{O}_{\text{ZK}}^N$ of $\mathcal{O}_{\text{ZK}}$. Then $\text{Sim}_1$ computes $\beta(\text{crs}(N))$ with $N \in \mathbf{N}_P$ as $\beta(\text{crs}(N)) := \mathcal{O}_{\text{ZK}}^N(\texttt{crs})$, and $\text{Sim}_1$ computes $\beta(\text{ZK}(t_1, t_2, N_2))$ with $N_2 \in \mathbf{N}_P$ as $\beta(\text{ZK}(t_1, t_2, N_2)) := \mathcal{O}_{\text{ZK}}^N(\texttt{prove}, \beta(t_1), \beta(t_2))$. In case of an extraction-failure, $\text{Sim}_1$ performs a (`extd`)-query to get extd. (Here and in the descriptions of $\text{Sim}_2, \ldots, \text{Sim}_5, \text{Sim}_f$, we implicitly require that $\beta(t)$ caches the results of the oracle queries and does not repeat the oracle query when $\beta$ 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 $z = A_{\text{crs}}(r_N)$, $\text{Sim}_1$ checks whether $z$ is equal to the (`crs`)-query outcomes for all oracles $\mathcal{O}_{\text{ZK}}^N$ which have been used so far.
- We define $\text{Sim}_2$ like $\text{Sim}_1$, 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 (`prove`, $x, w$)-query, it returns $\mathbf{S}(x, \text{crs}, \text{simtd})$ if $(x, w) \in R_{\text{hon}}^{\text{comp}}$ and $\bot$ otherwise.
- We define $\text{Sim}_3$ like $\text{Sim}_2$, 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 (`prove`, $x, w$)-query, it returns $\mathbf{S}(x, \text{crs}, \text{simtd})$ (even if $(x, w) \notin R_{\text{hon}}^{\text{comp}}$). Therefore the simulator only queries $(\mathit{prove}, x)$ and does not compute $w$ any more.
- We define $\text{Sim}_4$ like $\text{Sim}_3$, but we change $\beta$ to call the oracle $\mathcal{O}_0$ instead of computing $A_f$. For every $N \in \mathbf{N}_P$ the simulator $\text{Sim}_4$ maintains a distinct oracle $\mathcal{O}_0^N(\texttt{f}, \cdot, \cdot, \text{simtd})$ for $\text{crs}(N)$. Moreover, $\mathcal{O}_0^N(\texttt{f}, \cdot, \cdot, \text{simtd})$ gets a special command (simtd) for $\mathcal{O}_{\text{ZK}}^N$ upon which the simulation trapdoor simtd is output. Upon the first call $\mathcal{O}_0^N$ queries $\mathcal{O}_{\text{ZK}}^N$ with (simtd).
- We define $\text{Sim}_5$ like $\text{Sim}_6$, but we change $\beta$ and $\tau$ to call the simulating oracle $\mathcal{O}_1(\texttt{f}, \cdot, \cdot, \text{simtd})$ instead of querying $\mathcal{O}_0(\texttt{f}, \cdot, \cdot, \text{simtd})$.
- We define $\text{Sim}_6$ like $\text{Sim}_5$, but we change $\beta$ and $\tau$ to use encryption oracles instead of computing $A_{\text{enc}}$, $A_{\text{dec}}$, $A_{\text{ek}}$, $A_{\text{dk}}$. 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 (`ek`)-query, it returns ek. Upon a (`enc`, $m$)-query, it returns $\text{ENC}(\text{ek}, m)$. Upon a (`dec`, $c$)-query, it returns $\text{DEC}(\text{dk}, c)$. $\text{Sim}_6$ maintains an instance $\mathcal{O}_{\text{enc}}^N$ for each $N \in \mathbf{N}_P$. Then $\text{Sim}_6$ computes $\beta(\text{ek}(N))$ with $N \in \mathbf{N}_P$ as $\beta(\text{ek}(N)) := \mathcal{O}_{\text{enc}}^N(\texttt{ek})$. And it computes $\beta(\text{enc}(\text{ek}(N), t, M))$ with $N, M \in \mathbf{N}_P$ as $\beta(\text{enc}(\text{ek}(N), t, M)) := \mathcal{O}_{\text{enc}}^N(\texttt{enc}, \beta(t))$. And it computes $\beta(\text{dk}(N)) := \bot$. 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 $\mathcal{O}_{\text{enc}}^N(\texttt{dec}, c)$.
  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 $c = A_{\text{dk}}(r_N)$, $\text{Sim}_6$ checks whether $c$ is equal to the corresponding query outcomes for all oracles $\mathcal{O}_{\text{enc}}^N$ which have been used so far.
- We define $\text{Sim}_7$ like $\text{Sim}_6$, 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 (`enc`, $x$)-query, it returns $\text{ENC}(\text{ek}, 0^{|x|})$.

- We define $\mathrm{Sim}_f$ like $\mathrm{Sim}_7$, but we change $\beta$ to use signing oracles instead of computing $A_{\mathrm{vk}}, A_{\mathrm{sk}}, A_{\mathrm{sig}}$. More precisely, we assume an oracle $\mathcal{O}_{\mathrm{sig}}$ that internally picks $(\mathrm{vk}, \mathrm{sk}) \leftarrow \mathrm{KeyGen}_{\mathrm{sig}}(1^\eta)$ and that responds to two kinds of queries: Upon a $(\mathtt{vk})$-request, it returns vk, and upon a $(\mathtt{sig}, m)$-request, it returns $\mathrm{SIG}(\mathrm{sk}, m)$. $\mathrm{Sim}_f$ maintains an instance $\mathcal{O}_{\mathrm{sig}}^N$ for each $N \in \mathbf{N}_P$. Then $\mathrm{Sim}_f$ computes $\beta(\mathrm{vk}(N))$ with $N \in \mathbf{N}_P$ as $\beta(\mathrm{vk}(N)) := \mathcal{O}_{\mathrm{sig}}^N(\mathtt{vk})$. And $\beta(\mathrm{sk}(N))$ with $N \in \mathbf{N}_P$ as $\beta(\mathrm{sk}(N)) := \bot$. And $\beta(\mathrm{sig}(\mathrm{sk}(N), t, M))$ with $N, M \in \mathbf{N}_P$ as $\beta(\mathrm{sig}(\mathrm{sk}(N), t, M)) := \mathcal{O}_{\mathrm{sig}}^N(\mathtt{sig}, \beta(t))$.
  In the definition of $\tau(c) = \mathrm{vk}(N)$ and $\tau(c) = \mathrm{sk}(N)$ for $N \in \mathbf{N}_P$, instead of checking $c = A_{\mathrm{vk}}(r_N)$ and $c = A_{\mathrm{sk}}(r_N)$, $\mathrm{Sim}_f$ checks whether $c$ is equal to the corresponding query outcomes for all oracles $\mathcal{O}_{\mathrm{sig}}^N$ which have been used so far.

Large parts of the computational soundness proof follow the computational soundness proof for non-malleable ZK proofs from Backes, Bendun and Unruh [BBU13].

The following two lemmas are shown in the work of Backes, Bendun, and Unruh [BBU13].

## E.3 Sim is Dolev-Yao

**Lemma 8** (Underivable subterms)**.** *In a given step of the hybrid execution of $Sim_f$, let $S$ be the set of messages sent from $\Pi^C$ to $Sim_f$ before.*

*Let $u' \in \boldsymbol{T}$ be the message sent from $Sim_f$ to the protocol in that step. Let $C$ be a context and $u \in \boldsymbol{T}$ such that $u' = C[u]$ and $S \not\vdash u$.*

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

$$\begin{aligned} D ::= \Box \mid & \mathit{pair}(t, D) \mid \mathit{pair}(D, t) \mid \mathit{enc}(\mathit{ek}(N), D, M) \\ \mid & \mathit{enc}(D, t, M) \mid \mathit{sig}(\mathit{sk}(M), D, M) \\ \mid & \mathit{com}(D, r, N) \mid \mathit{com}(t, D, M) \\ \mid & \mathit{com}(t, D, N) \ (\textit{for public commitments}) \\ \mid & \mathit{ZK}(D, r, M) \mid \mathit{garbEnc}(D, M) \\ \mid & \mathit{garbSig}(D, M) \mid \mathit{garbCom}(D, M) \\ & \textit{with } N \in \boldsymbol{N}_P, M \in \boldsymbol{N}_E, t, t' \in \boldsymbol{T} \end{aligned}$$

*with $u = D[t_{bad}]$ such that $S \not\vdash t_{bad}$ and such that one of the following holds:*

1. $t_{bad} \in \boldsymbol{N}_P$
2. $t_{bad} = \mathit{enc}(p, m, N)$ *with* $N \in \boldsymbol{N}_P$
3. $t_{bad} = \mathit{sig}(k, m, N)$ *with* $N \in \boldsymbol{N}_P$
4. $t_{bad} = \mathit{ZK}(x, r, N)$ *with* $N \in \boldsymbol{N}_P$
5. $t_{bad} = \mathit{sig}(\mathit{sk}(N), m, M)$ *with* $N \in \boldsymbol{N}_P$, $M \in \boldsymbol{N}_E$
6. $t_{bad} = \mathit{com}(\mathit{crs}(N), m, M)$ *with* $N, M \in \boldsymbol{N}_P$ *and $t_{bad}$ is witness commitment*
7. $t_{bad} = \mathit{ZK}(t, r, N)$ *with* $N \in \boldsymbol{N}_P$
8. $t_{bad} = \mathit{crs}(N)$ *with* $N \in \boldsymbol{N}_P$
9. $t_{bad} = \mathit{ek}(N)$ *with* $N \in \boldsymbol{N}_P$
10. $t_{bad} = \mathit{vk}(N)$ *with* $N \in \boldsymbol{N}_P$
11. $t_{bad} = \mathit{sk}(N)$ *with* $N \in \boldsymbol{N}_P$
12. $t_{bad} = \mathit{dk}(N)$ *with* $N \in \boldsymbol{N}_P$

*Proof.* We prove the lemma by structural induction on $u$. We formulate the proof for an invocation of $\tau$, for an invocation of $\tau^*$ the proof is identical. There are the following cases:

**Case 1:** " $u \in \{\mathrm{ek}(N), \mathrm{vk}(N), \mathrm{crs}(N), \mathrm{dk}(N), \mathrm{sk}(N)\}$ with $N \notin \mathbf{N}_P$"

Let $u = C(N)$ for $C \in \{\mathrm{ek}, \mathrm{vk}, \mathrm{crs}, \mathrm{dk}, \mathrm{sk}\}$. By protocol conditions 1 and 13 each $C$-node has as annotation a nonce from $\mathbf{N}_P$. Therefore $u$ cannot be honestly generated, that means there is some $e \in \{0,1\}^*$ such that $\tau(e) = u$ and $u$ has the form $C(N^e)$. But then $S \vdash u$ contradicting the premise of the lemma.

**Case 2:** " $u \in \{\mathrm{ek}(N), \mathrm{vk}(N), \mathrm{crs}(N), \mathrm{dk}(N), \mathrm{sk}(N)\}$ with $N \in \mathbf{N}_P$"

Then the claim is fulfilled with $D := \Box$ and $t_{bad} = u$.

**Case 3:** "$u = \text{garb}(u_1)$"

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

**Case 4:** "$u = \text{garbEnc}(u_1, u_2)$ or $u = \text{garbSig}(u_1, u_2)$"

By protocol condition 2 no garbage term is generated by the protocol. So there exists a $c \in \{0,1\}^*$ with $\tau(c) = \text{garbEnc}(u_1, N^c)$ or $\tau(c) = \text{garbSig}(u_1, N^c)$. Since $S \vdash N^c$ it follows that $S \not\vdash u_1$, because $S \not\vdash u$. 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 $\text{garbEnc}(D', N^c)$, respectively $\text{garbSig}(D', N^c)$, shows the claim.

**Case 5:** "$u = \text{garbZK}(u_1, u_2, u_3)$"

As in the previous case follows $u_3 = N^c$ with $c \in \{0,1\}^*$, so $S \not\vdash u_1$ or $S \not\vdash u_2$. For the first case we can apply the induction hypothesis to $u_1$ leading to $t_{bad}$ and context $D'$. Then we use context $\text{garbZK}(D', u_2, u_3)$ to satisfy the lemma. In the other case we apply the induction hypothesis to $u_2$ leading to context $\text{garbZK}(u_1, D', u_3)$ and $t_{bad}$.

**Case 6:** "$u = \text{pair}(u_1, u_2)$"

Since $S \not\vdash u$ there is an $i \in \{1,2\}$ such that $S \not\vdash u_i$. Let $D$ be the context and $t_{bad}$ the term given by applying the induction hypothesis to $u_i$. Then $D_1 \coloneqq \text{pair}(D, M)$ or $D_2 \coloneqq \text{pair}(M, D)$ is the context for the term $u$ depending on $i$ with the same term $t_{bad}$.

**Case 7:** "$u = \varepsilon$"

This case cannot happen because $S \vdash \varepsilon$, 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 $u_1 = \varepsilon$ 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 $\tau$.

**Case 10:** "$u = N$ with $N \in \mathcal{N}$"

The context $D \coloneqq \square$ and term $t_{bad} := u$ satisfy the lemma in this case.

**Case 11:** "$u = N$ with $N \in \mathbf{N}_E$"

In this case $S \vdash u$ by definition and therefore the lemma's premise does not hold.

**Case 12:** "$u = \text{enc}(u_1, u_2, N)$ 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 $S \not\vdash u_1$"

Since $u_3 \notin \mathbf{N}_P$ it follows that $u$ cannot be honestly generated because of protocol condition 7. Therefore there is a $c \in \{0,1\}^*$ with $\tau(c) = \text{enc}(\text{ek}(M), u_2, N^c) = u$ for some $M \in \mathbf{N}_P$. Apply the induction hypothesis to $u_1$ getting $t_{bad}$ and context $D$ we can define $D' \coloneqq \text{enc}(D, u_2, N^c)$ fulfilling the claim of the lemma with $t_{bad}$.

**Case 14:** "$u = \text{enc}(u_1, u_2, u_3)$ 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 7. Therefore there is an $c \in \{0,1\}^*$ with $\tau(c) = \text{enc}(\text{ek}(M), u_2, N^c) = u$ for some $M \in \mathbf{N}_P$. Since $S \vdash u_1$, $S \vdash N^c$, and $S \not\vdash u$, it follows that $S \not\vdash u_2$. Let $D$ be the context and $t_{bad}$ be the term resulting by the induction hypothesis applied to $u_2$. Then $D' \coloneqq \text{enc}(\text{ek}(M), D, N^c)$ together with $t_{bad}$ satisfies the lemma.

**Case 15:** "$u = \text{com}(u_1, u_2, N)$ with $N \in \mathbf{N}_P$ and $u$ is a witness commitment"

By protocol condition 13, we know that there is a $c \in \{0,1\}^*$ such that $\tau(c) = \text{com}(\text{crs}(N), t, N^c)$. Hence, the lemma is satisfied by $t_{bad} = \text{com}(\text{crs}(N), t, N^c)$ and $D = \square$.

**Case 16:** "$u = \mathrm{com}(u_1, u_2, M)$ with $M \in \mathbf{N}_P$ and $u$ is a public commitment"

As above, by protocol condition 13 $u_1 = \mathrm{crs}(N)$ for some $N \in \mathbf{N}_P$. Since $u$ is a public commitment, $S \vdash \mathrm{uv}(u_2, M)$. Hence, by applying $\mathrm{open}(\mathrm{com}(u_1, u_2, M), \mathrm{uv}(u_2, M)) = u_2$, we know that by induction hypothesis there is a $D$ such that $S \vdash D[t_{bad}]$. Consequently, the lemma is satisfied by $D' = \mathrm{com}(u_1, D, M)$.

**Case 17:** "$u = \mathrm{com}(u_1, u_2, M)$ with $M \in \mathbf{N}_E$"

Since $u_3 \notin \mathbf{N}_P$ it follows that $u$ cannot be honestly generated because of protocol condition 7. Hence, there is a $c \in \{0,1\}^*$ such that $\tau(c) = \mathrm{com}(u', x, N^c)$. Since $S \vdash u'$, $S \vdash N^c$, and $S \nvdash u$, it follows that $S \nvdash x$. Let $D$ be the context and $t_{bad}$ be the term resulting by the induction hypothesis applied to $u_2$. Then $D' \coloneqq \mathrm{com}(u', D, N^c)$ together with $t_{bad}$ satisfies the lemma.

**Case 18:** "$u = \mathrm{sig}(u_1, u_2, N)$ with $N \in \mathbf{N}_P$"

Use context $D \coloneqq \Box$ and $t_{bad} = u$.

**Case 19:** "$u = \mathrm{sig}(\mathrm{sk}(N), u_1, u_3)$ 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 $D \coloneqq \Box$ and $t_{bad} = u$ proves the claim.

**Case 20:** "$u = \mathrm{sig}(u_1, u_2, u_3)$ 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 7 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 $D' \coloneqq \mathrm{sig}(\mathrm{sk}(M), D, N^s)$ completes the claim.

**Case 21:** "$u = \mathrm{ZK}(u_1, u_2, N)$ with $N \in \mathbf{N}_P$"

Defining $t_{bad} = u$ and $D \coloneqq \Box$ suffices.

**Case 22:** "$u = \mathrm{ZK}(u_1, u_2, N)$ with $N \in \mathbf{N}_E$"

Since $\tau$ uses randomnessTree for $u_2$ and $N \in \mathbf{N}_E$, we know that $S \nvdash u_1$.

In this case we use the induction hypothesis on $\mathrm{getPub}(u) = u_1$ to get the term $t_{bad}$ and a context $D$. Then using $t_{bad}$ and $D' \coloneqq \mathrm{ZK}(D, u_2, N)$ satisfies the lemma.

**Case 23:** "$u = \mathrm{ZK}(u_1, u_2, N)$ with $N \in \mathbf{N}_E$"

This case cannot occur because $u$ is not in the range of $\tau$.

**Case 24:** "$u = \hat{f}(t_1, \ldots, t_n, x, w, N)$"

By the definition of $\tau$, we know that $\tau(m) = u$ only if $m$ was already sent by the protocol; hence already $S \vdash u$ holds true and this case cannot occur. □

□

**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 $\Pi^C$ to $Sim_f$ up to that point.*

*Proof.* We prove it by induction on the recursion depth of the $\beta$-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 $\beta$ 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. $t' = \mathrm{ZK}(t, t_2, N)$ with $N \in \mathbf{N}_P$

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 $\mathrm{Sim}_f$ does not recursively invoke $\beta$ on $t$ but uses an oracle and produces $\mathrm{ENC}(\mathrm{ek}_N, 0^{\ell(t)})$. Analogously, in $\mathrm{Sim}_f$ $\mathrm{com}(\mathrm{crs}(N), t, M)$ does not recursively invoke $\beta$ but uses an oracles that commits to $0^{\ell(t)}$. By protocol condition 13, we know that the protocol does not send the term $\mathrm{com}(\mathrm{crs}(N^e), t, M)$ for $N^e \in \mathbf{N}_E$. The case $t' = \mathrm{ZK}(t_1, t, N)$ is not possible, either, because the simulator $\mathrm{Sim}_f$ calls the simulation oracle to construct the proof and therefore $\beta(\cdot)$ is not called on the witness $t$. Also the case $t' = \hat{f}(t_1, \ldots, t_n, t_{n+1}, t_{n+2}), N$ with $t \in \{t_1, \ldots, t_{n+2}\}$ and $N \in \mathbf{N}_P$ does not occur since all honest ZK-transformations are simulated.

**Case 1:** $S \vdash \mathrm{sig}(\mathrm{sk}(N), t, M) = t'$. Using $\mathrm{ver}_{\mathrm{sig}}(\mathrm{vkof}(t'), t') = t$ 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:** $S \vdash \mathrm{ZK}(t, t_2, N) = t'$. The lemma follows by applying the destructor getPub. □

□

**Lemma 10.** *For any invocation $\tau(m)$ of $\tau$ 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 $\tau(m)$. Then $S \vdash \tau(m)$.*

*In particular, $Sim_f$ is DY for $\boldsymbol{M}$ and* Π.

*Proof.* Assume $\mathrm{Sim}_f$ is not DY for $\mathbf{M}$ and Π. Then there is a $l \in \mathbb{N}$ such that $S_l \not\vdash t_l$ according to the definition of Dolev-Yaoness in the CoSP paper [BHU09]. Let $t$ be the first such message and $S$ the corresponding set having $S \not\vdash t$. 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 1-12 given by Lemma 8. 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.

**Case 1:** "$t_{bad} = N \in \mathbf{N}_P$".
By construction of $\beta$ and $\mathrm{Sim}_f$ follows that $\mathrm{Sim}_f$ has only access to $r_N$ if $\beta(N)$ is computed directly or in $\tau$. Because $S \not\vdash N$ we get by Lemma 9 that $\beta$ was never invoked on $N$, i.e. $\mathrm{Sim}_f$ has only access to $r_N$ via $\tau$. Considering the definition of $\tau$ 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 $N \in \mathbf{N}_P$ 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 $\tau$ is negligible.

**Case 2:** "$t_{bad} = \mathrm{enc}(p, m, N)$ with $N \in \mathbf{N}_P$".
By definition $\tau$ 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:** "$t_{bad} = \mathrm{com}(\mathrm{crs}(N), m, M)$ with $N, M \in \mathbf{N}_P$ and $t_{bad}$ is witness commitment".
By definition $\tau$ 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 5:** "$t_{bad} = \mathrm{ZK}(x, r, N)$ with $N, M \in \mathbf{N}_P$".
By definition of $\tau$, $t_{bad}$ 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 6:** "$t_{bad} = \mathrm{crs}(N)$ with $N \in \mathbf{N}_P$".
Since Lemma 9 $\beta$ was not called on $\mathrm{crs}(N)$, therefore $\tau(m) \neq \mathrm{crs}(N)$ for all $m \in \{0,1\}^*$.

**Case 7:** "$t_{bad} = \text{sig}(\text{sk}(N), m', M)$ with $N \in \mathbf{N}_P$, $M \in \mathbf{N}_E$".
Because $S \not\vdash t_{bad}$ follows that $\beta$ 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 $\text{vk}_N$. Because of the strongly existential unforgeability this can only be the case with negligible probability.

**Case 8:** "$t_{bad} = \text{ek}(N)$ with $N \in \mathbf{N}_P$".
By Lemma 9 follows that the function $\beta$ was not called on $t_{bad}$. So $\text{ek}_N$ was not requested from the encryption oracle. Protocol condition 8 implies that the only term sent by $\Pi^C$ containing $\text{dk}(N)$ is of the form $\text{ZK}(\cdot, \square, N)$. But since the zero-knowledge proofs are simulated in the latter case $\text{dk}(N)$ is not part of any computation, i.e. $\beta(\text{dk}(N))$ is not computed and $\text{dk}_N$ is never requested from the encryption oracle. By Lemma 9 $S \not\vdash \text{dk}(N)$. Hence for all terms of the form $t = \text{enc}(\text{ek}(N), \cdot, \cdot)$ holds $S \not\vdash t$ and therefore no encryption having encryption key $\text{ek}_N$ is requested from the oracle. Thus the only remaining possibly use of $\text{ek}_N$ is requesting the decryption oracle. But by implementation condition 4 these requests will always fail unless $A_{\text{ekof}}(m_{bad}) = \text{ek}_N$, i.e. $\text{ek}_N$ has already been guessed. This can only happen with negligible probability.

**Case 9:** "$t_{bad} = \text{vk}(N)$ with $N \in \mathbf{N}_P$".
By Lemma 9, $\beta(\text{vk}(N))$ is never computed and $\text{vk}_N$ is never requested from the signing oracle. Assume $S \vdash \text{sk}(N)$. Then $S \vdash \text{sig}(\text{sk}(N), \varepsilon, N^e)$ for some $N^e \in \mathbf{N}_E$. But using the destructor vkof follows that $S \vdash \text{vk}(N)$, a contradiction. So we also have $S \not\vdash \text{sk}(N)$ and $S \not\vdash \text{sig}(\text{sk}(N), \cdot, \cdot) = t$. Thus $\beta(\text{sk}(N))$ and $\beta(t)$ are never computed and therefore neither $\text{sk}_N$ nor a signature with respect to $\text{sk}_N$ is requested from the signing oracle. So the probability that $\text{vk}_N = m_{bad}$ occurs as input of $\tau$ is negligible.

**Case 10:** "$t_{bad} = \text{sk}(N)$ with $N \in \mathbf{N}_P$".
Because of protocol condition 9 $t_{bad}$ can only occur in terms of the form $\text{sig}(\square, \cdot, \cdot, N)$ and $\text{ZK}(\cdot, \square, N)$. In the second case the construction of the proof is replaced by the simulation oracle and could be done by the adversary itself. So the adversary is be able to compute $\text{sk}_N$ only having signatures. By the strongly existential unforgeability of the signature scheme, this can only happen with negligible probability.

**Case 11:** "$t_{bad} = \text{dk}(N)$ with $N \in \mathbf{N}_P$".
Procotol condition 8 ensures that dk only occurs in terms of the form $\text{dec}(\square, \cdot)$ and $\text{ZK}(\cdot, \square, N)$. The latter case is replaced by the simulation oracle and can be computed by the adversary itself. Therefore the adversary is able to decrypt ciphertexts only knowing ciphertexts. By the IND-CCA property, this can only occur with negligible probability.

In total we get that if $\text{Sim}_f$ is not DY, then with non-negligible probability $\text{Sim}_f$ performs a computation of $\tau(m_{bad})$ but $m_{bad}$ can only occur with negligible probability as an argument of $\tau$, a contradiction. Therefore the assumption that $\text{Sim}_f$ is not DY has to be false, i.e. $\text{Sim}_f$ is DY. □

## E.4 Sim is indistinguishable

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

1. *In the hybrid execution of Sim and $\text{Sim}_3$ it holds with overwhelming probability: If $(x, w) \in R^{\text{sym}}_{\text{hon}}$ and $x, w$ occur as node annotation of a ZK node in the execution, then it holds $(\beta(x), \beta(w)) \in R^{\text{comp}}_{\text{hon}}$.*
2. *In the hybrid execution of $\text{Sim}_2$ 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 $\eta$ mapping terms to bitstrings. $\eta$ depends on the current state of the execution. We will use $\eta$ in both parts of the lemma. So let $t_1, \ldots, t_n$ be the terms sent by the protocol to the simulator so far.

For any term or subterm $t$ that occurs as argument to $\beta$ or output of $\tau$, we define $\eta$ as follows:

- For $t = N^m$ define $\eta(t) := m$.
- For $t = C(t_1, \ldots, t_n, N^m)$ define $\eta(t) := m$ for all $C$ as stated in definition 17.[18]

[18] These are {ek, dk, vk, sk, enc, sig, com, crs, garbCom, garbSig, garbEnc, garb}

- For $t = \mathrm{crs}(N)$ with $N \in \mathbf{N}_P$ define $\eta(t)$ to be the crs produced by the oracle $\mathcal{O}^N_{\mathrm{ZK}}$.
- For $t = \mathrm{ZK}(x, r, M)$ with $M \in \mathbf{N}_P$ define $\eta(t)$ to be proof produced by $\mathcal{O}^N_{\mathrm{ZK}}$ in the computation of $\beta(t)$.
- For $t = N$ with $N \in \mathbf{N}_P$ we distinguish 2 cases. If $t$ does neither occur in a term of the form $\mathrm{crs}(t)$ nor in $\mathrm{ZK}(x, r, t)$ for some $x, r$ then define $\eta(t) := r_N$. Otherwise let $\eta(t)$ be undefined, i.e. $\eta(t) := \bot$.

Note that $\eta$ 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 $\eta$. We distinguish by the type of $\eta$'s output.

- Type nonce. A collision $r_M = r_N$ for $M, N \in \mathbf{N}_P$ occurs with negligible probability. For $N, M \in \mathbf{N}_P$, a collision occurs with negligible probability, because $r_N = r_M$ occurs with negligible probabilityand because a collision with the randomness of $\mathcal{O}^N_{\mathrm{ZK}}$ has negligible probability (otherwise it would be possible to guess that randomness, compute the simulation trapdoor and fake proofs). 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 $\tau$, i.e. $M = N^n$ for some $n \in \{0,1\}^*$. First, let $N$ be a nonce occurred inside $\mathrm{crs}(N)$. Then it holds $\eta(N) = \bot \neq n = \eta(N^n)$.
  Otherwise, if $N$ was used before $n$ was received by the simulator, then $n$ would have been parsed to $N$ by construction of $\tau$. So the first occurrence of $N$ has to be after $n$ was received. But then the adversary guessed a nonce. This can only happen with negligible probability.
- Type decryption key. For the same reasons as in the case of type nonce we only consider the case $\eta(\mathrm{dk}(N)) = \eta(\mathrm{dk}(M))$ for $N \in \mathbf{N}_P, M \in \mathbf{N}_E$. By protocol condition 2, it follows that $\mathrm{dk}(M) = \mathrm{dk}(N^d)$ for some $d \in \{0,1\}^*$, $\mathrm{dk}(N^d)$ was subterm of an output of $\tau$, and $d$ was not output of $\beta$ earlier (otherwise $d$ would have been parsed to $\mathrm{dk}(N)$). So the adversary used either no input or only encryptions plus the encryption key to compute $\mathrm{dk}(N)$. By the CCA property, this can only be the case with negligible probability.
- Type signing key. This case is completely analogue to the decryption key type using the strongly existentially unforgeability instead of the CCA property.
- Type encryption key. As in the previous cases we can only need to consider $\eta(\mathrm{ek}(N)) = \eta(\mathrm{ek}(M))$ for $N \in \mathbf{N}_P, M \in \mathbf{N}_E$. By protocol condition 2, it follows that $\mathrm{ek}(M) = \mathrm{ek}(N^e)$ for some $e \in \{0,1\}^*$. But then $\tau$ parsed $e$ to $\mathrm{ek}(N^e)$, so neither $\mathrm{ek}(N)$ nor $\mathrm{dk}(N)$ was used. This means the adversary guessed an encryption key without having any information about it. This can only happen with negligible probability.
- Type verification key and common reference string. Analogue to the case of encryption key.
- Type zero-knowledge proof. Because $\tau$ 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, signature, and commitment. 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 $\eta$ to a term of this type leads to a unique bitstring which cannot be hit by any other term of this type (by implementation condition 17).
- No type. The only term which has no type is $\mathrm{garb}(t)$ for $t \in \mathbf{T}$. By protocol condition 2 and construction of $\tau$, it has to hold that $t = N^m$ for some $m \in \{0,1\}^*$.

*Proof of part* 1 *of the lemma.*

By Definition 18[19], it suffices to show that if $(x, w) \in R^{\mathrm{sym}}_{\mathrm{hon}}$ then there is a consistent $\eta \in \mathcal{E}$ such that $(\mathrm{img}_\eta(x), \mathrm{img}_\eta(w)) = (\beta(x), \beta(w))$ since $\beta(x) \neq \bot \neq \beta(w)$. We show that the $\eta$ defined above satisfies this criterium. Here, we prove the case for $\mathrm{Sim}_3$. The proof for Sim is analoguous with the only difference in the cases of ZK and crs. Here, the definition of $\eta$ is done as for enc and ek and the proof, as well.

For any term $t$ that can occur in the execution of $\mathrm{Sim}_3$ 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 7 and 13). 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 $\eta$.
- "$t = N$ 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

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

$C \in \{\mathrm{ek}, \mathrm{dk}, \mathrm{vk}, \mathrm{sk}\}$ be the constructor such that $t = C(u)$. Then, it holds that $\mathrm{img}_\eta(t) = A_C(\mathrm{img}_\eta(u)) \overset{(*)}{=} A_C(\beta(u)) = \beta(t)$. Since $u \in \mathbf{N}_P$ and occurs in $C(u)$, it follows that $u$ does neither occur in $\mathrm{crs}(u)$ nor in $\mathrm{ZK}(x, r, u)$ for $x, r \in \mathbf{T}'$ (protocol conditions forbid that these nonces are used more than once). Thus $\mathrm{img}_\eta(u) = r_u = \beta(u)$. Hence equality $(*)$ holds.

- "$t = \mathrm{crs}(N)$ with $N \in \mathbf{N}_P$". By definition $\beta(t)$ produces the crs using $\mathcal{O}^N_{\mathrm{ZK}}$ and $\mathrm{img}_\eta(\mathrm{crs}(N)) = \eta(\mathrm{crs}(N))$ which was defined as $\beta(t)$. Thus, it holds $\mathrm{img}_\eta(t) = \beta(t)$.
- "$t = \mathrm{crs}(N)$ with $N \in \mathbf{N}_E$". This case is analogue to the case $\mathrm{ek}(N)$ with $N \in \mathbf{N}_E$.
- "$t = \mathrm{enc}(u_1, u_2, u_3)$". 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 7). 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 $\tau$, it follows that $t$ was honestly generated. This means there was a sig-computation node that produced $t$. By protocol condition 11 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 $\beta(\mathrm{pair}(u_1, u_2)) = A_{\mathrm{pair}}(\beta(u_1), \beta(u_2)) = A_{\mathrm{pair}}(\mathrm{img}_\eta(u_1), \mathrm{img}_\eta(u_2)) = \mathrm{img}_\eta(\mathrm{pair}(u_1, u_2))$.
- "$t \in \{\mathrm{string}_0(u), \mathrm{string}_1(u), \varepsilon\}$ with $u \in \mathbf{T}$". These cases are analogous to the case $t = \mathrm{pair}(u_1, u_2)$.
- "$t \in \mathrm{ZK}(t_1 \wedge t_1', t_2 \wedge t_2', \mathrm{rand}(N, N'))$ with $t_i, t_i' \in \mathbf{T}$". This case are analogous to the case $t = \mathrm{pair}(u_1, u_2)$ using splitAnd.
- "$t \in \{\mathrm{garb}(u_1)$, $\mathrm{garbSig}(u_1, u_2, u_3)$, $\mathrm{garbEnc}(u_1, u_2)$, $\mathrm{garbCom}(u_1, u_2), \mathrm{com}(u_1, x, u_2)\}$ where $u_i \in \mathbf{T}$ and $\alpha(x) = \bot$." By protocol condition 2 follows that $t$ was generated by $\tau$, i.e. the last argument of $t$ has the form $N^m$ for some $m \in \{0,1\}^*$. By definition of $\beta$, it holds that $\beta(t) = m$. On the other hand, by definition of $\mathrm{img}_\eta$, it holds $\mathrm{img}_\eta(t) = \eta(t) = m$, as well.
- "$\mathrm{ZK}(t_1, t_2, t_3)$ where $\alpha(x) = \bot$ for each $x \in \mathrm{extrWit}(t_1)$". This case is analogous to the case above.

*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 $\mathrm{Sim}_2$, there is some $\eta$ such that $m = \mathrm{img}_\eta(\tau(m))$ holds. Then it follows by definition 18[20] $(m_x, m_w) \in R^{\mathrm{comp}}_{\mathrm{adv}} \implies (\mathrm{img}_\eta(\tau(m_x)), \mathrm{img}_\eta(\tau(m_w))) \in R^{\mathrm{comp}}_{\mathrm{adv}} \implies (\tau(m_x), \tau^*(m_w)) \in R^{\mathrm{sym}}_{\mathrm{adv}}$.

Take the same definition of $\eta$ 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 $\tau^*$, as well, since all cases for $\tau^*$ occur in $\tau$.

- $\tau(m) = N$ for some $N \in \mathbf{N}_P$
  By construction of $\tau$, 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 7. Then $\mathrm{img}_\eta(\tau(m)) = r_N = m$ where the last equality holds because of the definition of $\tau$.
- $\tau(m) = N^m$
  Then by construction of $\eta$ 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 $\eta$ 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 $\tau$ follows that $m$ was earlier output by $\beta$ and thus evaluating $t$ again using $\mathrm{img}_\eta$ 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 $m$.
- $\tau(s) = \{\mathrm{sig}(\mathrm{sk}(M), t, N), \mathrm{com}(t_1, t_2, N)\}$ 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 $\tau$, 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 $N \in \mathbf{N}_P$

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

The same as the case of $\mathrm{ek}(N)$.

- $\tau(m) = \mathrm{crs}(N)$ for some $N \in \mathbf{N}_P$
  By definition of $\tau$ follows that $m$ was output after a call of $\beta$ 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 $\eta(N)$.
- $\tau(m) = \mathrm{ZK}(t_1, t_2, N_2)$ for some $N_1, N_2 \in \mathbf{N}_P$
  By definition of $\eta$ follows that $\mathrm{img}_\eta(\mathrm{ZK}(t_1, t_2, N_2)) = \eta(\mathrm{ZK}(t_1, t_2, N_2)) = m$.
- $\tau(m) = \mathrm{pair}(t_1, t_2)$
  This case follows by the induction hypothesis and the determinism of the implementations.
- $\tau(m) \in \{\mathrm{string}_0(t_1), \mathrm{string}_1(t_1), \varepsilon\}$
  The case of $\varepsilon$ is trivial, since the implementation is deterministic. For the other cases holds that - by definition of $\tau$ - $t_1 = \tau(m')$ 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.
- $\tau(m) \in \{\mathrm{enc}(\mathrm{ek}(M), t, N^m),\ \mathrm{ek}(N^m),\ \mathrm{dk}(N^m),\ \mathrm{garbEnc}(t, N^m),\ \mathrm{com}(t, t_2, N^m),\ \mathrm{garbCom}(t, N^m),$
  $sig(\mathrm{sk}(M), t, N^m),$
  $\mathrm{garbSig}(t, N^m),\ \mathrm{vk}(N^m),\ \mathrm{sk}(N^m),\ \mathrm{crs}(N^m),\ \mathrm{ZK}(x, r, N^m),\ \mathrm{garb}(N^m)\}$ for some $M \in \mathbf{N}_P$
  All of these cases follow immediately by definition of $\eta$ and definition 17.

The proof for $\mathrm{Sim}_2$ is the same. Recall, the only difference between $\mathrm{Sim}_3$ and $\mathrm{Sim}_2$ is that $\mathrm{Sim}_3$ does not check if $(x, w) \in R_{\mathrm{hon}}^{\mathrm{comp}}$ any more.

□

**Lemma 12** (No invalid symbolic witnesses)**.** *Assume that $Sim_3$ is DY. Then, in the $\boldsymbol{D}_{\mathrm{trans}}$-hybrid execution of $Sim_3$, for each ZK node with arguments $t_1, t_2, t_3$, it holds that $(t_2, extrWit(t_3)) \in R_{\mathrm{hon}}^{\mathrm{sym}}$ with overwhelming probability.*

*The same holds for Sim if Sim is DY.*

*Proof.* If $\mathrm{Sim}_3$ is DY, then the hybrid execution of $\mathrm{Sim}_3$ 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 $\mathrm{Sim}_3$ is DY, this occurs only with negligible probability.

In the symbolic execution, the property $(t_2, t_3) \in R_{\mathrm{hon}}^{\mathrm{sym}}$ holds since Definition 6 by construction of $R_{\mathrm{hon}}^{\mathrm{sym}}$ and $R_{\mathrm{hon}}^{\mathrm{comp}}$. Thus in the case that the hybrid execution corresponds to a symbolic one, it follows that $(t_2, t_3) \in R_{\mathrm{hon}}^{\mathrm{sym}}$ with overwhelming probability.

The same proof shows the statement for Sim. □

**Lemma 13** (Indistinguishability of Simulators)**.**

*(i) We have*

$$\mathit{TH\text{-}Nodes}_{\boldsymbol{M},\Pi_{\mathsf{p}},\mathit{Sim}}(k) \overset{C}{\approx} \mathit{TH\text{-}Nodes}_{\boldsymbol{M},\Pi_{\mathsf{p}},\mathit{Sim}_f}(k)$$

*(ii) We have that Sim is DY if and only if $Sim_f$ is DY.*

*(iii) Let $\mathit{H\text{-}Nodes}^S_{\boldsymbol{M},\Pi_{\mathsf{p}},\mathit{Sim}_i}(k)$ be trace of the knowledge set $S$ from hybrid execution with $Sim_i$. Then, we have*

$$\mathit{H\text{-}Nodes}^S_{\boldsymbol{M},\Pi_{\mathsf{p}},\mathit{Sim}}(k) \overset{C}{\approx} \mathit{H\text{-}Nodes}^S_{\boldsymbol{M},\Pi_{\mathsf{p}},\mathit{Sim}_f}(k)$$

*Proof of (i) and (ii).* For $x \in \{1, \ldots, 5, f\}$ or $x$ being the empty word. The same way, we denote the event that a simulator $\mathrm{Sim}_x$ is DY in that execution by $\mathrm{DY}_x$. We abbreviate $\mathrm{TH\text{-}Nodes}_{\mathbf{M},\Pi_{\mathsf{p}},\mathrm{Sim}_x}(k)$ as $\mathrm{TH\text{-}Nodes}_x$.

To show the lemma, we will show that

$$(\text{DY}, \text{TH-Nodes}) \overset{C}{\approx} (\text{DY}_1, \text{TH-Nodes}_1)$$
$$(\text{DY}_1, \text{TH-Nodes}_1) \overset{C}{\approx} (\text{DY}_2, \text{TH-Nodes}_2)$$
$$(\text{DY}_2, \text{TH-Nodes}_2) \overset{C}{\approx} (\text{DY}_3, \text{TH-Nodes}_3)$$
$$(\text{DY}_5, \text{TH-Nodes}_5) \overset{C}{\approx} (\text{DY}_6, \text{TH-Nodes}_6)$$
$$(\text{DY}_6, \text{TH-Nodes}_6) \overset{C}{\approx} (\text{DY}_7, \text{TH-Nodes}_7)$$
$$(\text{DY}_7, \text{TH-Nodes}_7) \overset{C}{\approx} (\text{DY}_f, \text{TH-Nodes}_f)$$

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 $(\text{DY}_2, \text{TH-Nodes}_2) \overset{C}{\approx} (\text{DY}_3, \text{TH-Nodes}_3)$
at the end, because we need the intermediate result to prove it.

- $(\text{DY}, \text{TH-Nodes}) \overset{C}{\approx} (\text{DY}_1, \text{TH-Nodes}_1)$

  Transforming Sim to $\text{Sim}_1$ is done by replacing invocations of the ZK algorithms by oracle-queries. We can replace $A_{\text{crs}}(r_N)$ by a $(\texttt{crs})$-query to $\mathcal{O}^N_{\text{ZK}}$ because $N$ is only used inside this crs (protocol condition 13) and the distributions of the implementation and the oracle are the same. Since $\tau(c)$ in $\text{Sim}_1$ not checks whether $c = A_{\text{crs}}(r_N)$ but whether $c$ is the result of some $(\texttt{crs})$-query, the node traces have the same distribution.

  The same holds for the replacement of $A_{\text{ZK}}$ by the $(\mathit{prove}, x, w)$ oracle query to $\mathcal{O}^N_{\text{ZK}}$. The randomness – the fourth argument of the ZK proof – only occurs inside this proof and nowhere else (protocol condition 7), so we can replace it by the oracle's randomness as in the crs case.

  By implementation condition 26, it holds that if $(x, w) \notin R^{\text{comp}}_{\text{hon}}$ the implementation, as well as the oracle, output $\perp$. So $A_{\text{ZK}}$ and the $(\texttt{prove}, x, w)$-query return $\perp$ in the same cases. In the case that $(x, w) \in R^{\text{comp}}_{\text{hon}}$ both compute a proof of $x$ using witness $w$. Thus, it holds $(\text{DY}, \text{TH-Nodes}) \overset{C}{\approx} (\text{DY}_1, \text{TH-Nodes}_1)$.

- $(\text{DY}_1, \text{TH-Nodes}_1) \overset{C}{\approx} (\text{DY}_2, \text{TH-Nodes}_2)$

  In this step we replace $\mathcal{O}_{\text{ZK}}$ by $\mathcal{O}_{\text{Sim}}$ which returns a simulated proof for $x$ if for the input $(x, w)$ it holds that $(x, w) \in R^{\text{comp}}_{\text{hon}}$.

  If we change both simulators to not extract the proof in case of an extraction failure, then the TH-Nodes does not change. The simulator stops after handling extraction failures in any case. By definition of zero-knowledge these two modified cases are indistinguishable (using the fact that the simulator and prover are only invoked if $(x, w) \in R^{\text{comp}}_{\text{hon}}$). Thus $(\text{DY}_1, \text{TH-Nodes}_1)$ and $(\text{DY}_2, \text{TH-Nodes}_2)$ are indistinguishable, too.

- $(\text{DY}_3, \text{TH-Nodes}_3) \overset{C}{\approx} (\text{DY}_4, \text{TH-Nodes}_4)$

  As the randomness can only occur at transformations, letting the oracles draw the randomness yields an indistinguishable execution. Moreover, adding an extra check for the validity of the witness is possible because all proofs have been checked anyway (protocol conditions 17). Hence, the traces are indistinguishable and the Dolev-Yaoness carries over.

- $(\text{DY}_4, \text{TH-Nodes}_4) \overset{C}{\approx} (\text{DY}_5, \text{TH-Nodes}_5)$

  We apply the sound transformation property and obtain the indistinguishability of the traces and hence the preservation of the DYness.

- $(\text{DY}_5, \text{TH-Nodes}_5) \overset{C}{\approx} (\text{DY}_6, \text{TH-Nodes}_6)$

  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 $\text{Sim}_1$. Because $\text{Sim}_5$ does not compute witnesses of zero-knowledge proofs anymore, nonces of encryptions are only used once (by protocol condition 7). 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 $\tau$ accordingly, cf. the replacement of $A_{\text{crs}}$ in $\text{Sim}_1$). In addition we can define $\beta(\text{dk}(N)) := \bot$ because decryption keys are not used as input to $\beta$ (by protocol condition 8 and the use of an oracle for decrypting).

We did neither change the bitstrings that are sent to the adversary nor the way they are parsed. So the property of DY did not change either.

- $(\text{DY}_6, \text{TH-Nodes}_6) \overset{C}{\approx} (\text{DY}_7, \text{TH-Nodes}_7)$

  In the step from $\text{Sim}_6$ to $\text{Sim}_7$, 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 $\tau$ the protocol execution asks only for decryptions of ciphertexts which were not generated by the encryption oracle (since only $\beta$ 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. In order to get indistinguishability the adversary has to be able to use ZK-breaks, DYness and node traces to distinguish both executions. Obviously, it is possible to use DYness and the node traces. For the case of ZK-breaks, we have to require that $R_{\text{adv}}^{\text{sym}}$ is efficiently decidable.

  Thus replacing ENC by $\mathcal{O}_{\text{fake}}$ leads to an indistinguishable execution and hence $(\text{DY}_6, \text{TH-Nodes}_6)$ and $(\text{DY}_7, \text{TH-Nodes}_7)$ are computationally indistinguishable.

- $(\text{DY}_7, \text{TH-Nodes}_7) \overset{C}{\approx} (\text{DY}_f, \text{TH-Nodes}_f)$

  As in the case for $\text{Sim}_5$ and $\text{Sim}_6$, we have the case that after removing the witnesses in $\text{Sim}_5$ the nonces, used as randomness for signatures, are only used (by protocol condition 7) once for signing a message.

  The same holds for verification and signing keys (by protocol condition 1). Thus we can replace signing and computation of verification/signing keys by invocations of $\mathcal{O}_{\text{sig}}$ without changing the distribution of $(\text{DY}, \text{TH-Nodes})$ (since we adopted $\tau$ accordingly, cf. the replacement of $A_{\text{crs}}$ in $\text{Sim}_1$). In a run of the protocol $\beta$ is never applied to $\text{sk}(N)$ (by protocol condition 9 and the use of an oracle for signing), so we can define $\beta(\text{sk}(N)) := \bot$ without changing the distribution of $(\text{DY}, \text{TH-Nodes})$.

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

  We have already proven that $(\text{DY}_{\text{f}}) \overset{C}{\approx} (\text{DY}_3)$. Together with the fact that $\text{Sim}_f$ is DY (Lemma 10), it follows that $\text{Sim}_3$ is DY. By Lemma 12, it follows that $(t_2, t_3) \in R_{\text{hon}}^{\text{sym}}$ for all ZK-nodes with arguments $t_1, \ldots, t_6$ in a $\mathbf{D}_{\text{trans}}$-hybrid execution of $\text{Sim}_3$ (with overwhelming probability). Applying Lemma 11 leads to $(\beta(t_2), \beta(t_3)) \in R_{\text{hon}}^{\text{comp}}$ for a $\mathbf{D}_{\text{trans}}$-hybrid execution of $\text{Sim}_3$ with overwhelming probability. The only difference between $\text{Sim}_2$ and $\text{Sim}_3$ is that $\text{Sim}_2$ checks whether $(\beta(t_2), \beta(t_3)) \in R_{\text{hon}}^{\text{comp}}$ or not. Because this check would succeed with overwhelming probability in $\text{Sim}_3$, it actually succeeds in $\text{Sim}_2$.

  Thus the distribution of $(\text{DY}, \text{TH-Nodes})$ is the same in $\text{Sim}_2$ as in $\text{Sim}_3$.

  □

□

*Proof of (iii).* Let $\Omega_i := \text{H-Nodes}^S_{\mathbf{M},\Pi_{\text{p}},\text{Sim}_i}(k)$. $\Omega$ and $\Omega_1$ are indistinguishable as only a different randomness is used. For $\Omega_1$ and $\Omega_2$ we can reduce the indistinguishability to the zero-knowledge property. We construct a machine $M^D$ that breaks the zero-knowledge property with non-negligible probability if there is a distinguisher $D$ that distinguishes $\Omega_1$ from $\Omega_2$.

- Draw a random coin $b \leftarrow_R \{0,1\}$.
- Internally, run the hybrid execution with the simulator $\text{Sim}_{b+1}$.
- Forward every prover or simulator call of $\beta$ to the zero-knowledge challenger.
- At the end of the execution, output the trace of the knowledge set $S$ to the distinguisher $D$. If $D$ guesses $\Omega_{b+1}$, $M^D$ guesses $b$; otherwise, $M^D$ makes a random guess $b' \leftarrow_R \{0,1\}$.

$\Omega_2$ and $\Omega_3$ are indistinguishable because in $(i)$ it has been shown that that omitting the additional check does not change the trace, hence also not the knowledge set trace length. Moreover, this additional check does not affect $\tau$, and the result of $\beta$ because $\beta$ is only called on true proofs in $\Omega_2$ and $\Omega_3$.

$\Omega_3$ and $\Omega_4$ again are indistinguishable since only the randomness is now freshly chosen for every call. But the only place at which a nonce is reused is in the witness of a proof. $\mathrm{Sim}_3$ and $\mathrm{Sim}_4$, however, simulate all proofs and therefore do not use the witnesses at all.

The indistinguishability of $\Omega_4$ and $\Omega_5$ can be reduced against the IND-CCA property of the encryption scheme. We construct a machine $M^D$ that breaks the IND-CCA property with non-negligible probability if there is a distinguisher $D$ that distinguishes $\Omega_4$ from $\Omega_5$.

- Draw a random coin $b \leftarrow_R \{0,1\}$.
- Internally, run the hybrid execution with the simulator $\mathrm{Sim}_{b+4}$.
- Forward every encryption call of $\beta$ to the IND-CCA challenger.
- Forward every decryption call of $\tau$, i.e., excluding the cases in which $\tau$ only does a look-up, to the IND-CCA challenger.
- At the end of the execution, output the trace of the knowledge set $S$ to the distinguisher $D$. If $D$ guesses $\Omega_{b+4}$, $M^D$ guesses $b$; otherwise, $M^D$ makes a random guess $b' \leftarrow_R \{0,1\}$.

The indistinguishability of $\Omega_5$ and $\Omega_f$ again follows from the fact that only proofs reuse randomness in the witness. But all proofs are simulated; therefore, letting the signing oracle choose fresh randomness is indistinguishable from using the protocol nonce, which is also fresh by protocol condition 7. □

**No quasi extraction-failure.** We first show that no quasi extraction-failures happen, and then show that even extraction-failures do not happen with more than negligible probability. In order to be able to prove that extraction failures do not happen, we have to apply the extractability. However, the extractability property of the zero-knowledge proofs is not applicable to simulated proofs. Therefore, we have to prove extraction failures for $\mathrm{Sim}_1$, which does not simulate any zero-knowledge proofs. However, without applying the zero-knowledge property we cannot show the symbolic zero-knowledge property, but we can still apply the IND-CCA property of the encryption scheme. Hence, we show a weaker property, namely that the only reason why extraction failures can occur is that the symbolic zero-knowledge property is violated.

Formally, we define in Figure 13 a *witness-knowledge relation* $\vdash_w$, which extends the knowledge relation $\vdash$ with all messages that already occurred in a zero-knowledge proof as a witness. Then, we define a relaxed symbolic extraction algorithm $\mathbf{SymbExtr'}$, which tries to find a valid symbolic witness $w$ such that $S \vdash_w w$ and outputs $\bot$ if it fails. We say that a *quasi-extraction failure* happens if $\mathbf{SymbExtr'}(S, x) = \bot$.

The proof goes along the lines of the proof of extraction failures in the work of Backes, Bendun, and Unruh [BBU13]. We first show that in $\mathrm{Sim}_1$ ZK-Breaks happen with negligible probability (Lemma 14). Then, we show that $\tau$ and $\tau^*$ with overwhelming probability output a term $t$ such that $S \vdash_w t$ (Lemma 18). And finally we show that quasi-extraction failures only happen with negligible probability (Lemma 19).

The heart of this proof lies in the proof of the statement that $\tau$ and $\tau^*$ with overwhelming probability only produce terms $t$ such that $S \vdash_w t$ (Lemma 18). Even though we cannot apply the zero-knowledge property, we can still show that encryptions hide their plaintexts, even if the attacker knows the extraction trapdoor. However, since the zero-knowledge proofs might contain in the witness the randomness with which a ciphertext has been constructed, we cannot naïvely apply the IND-CCA property for showing that encryptions hide their plaintexts. We construct for every protocol nonce $n$ a simulator $\mathrm{Sim}_{cf,n}$ that fakes all ciphertexts that observably contain this nonce, and show that this nonce is not guessable in the execution with $\mathrm{Sim}_{cf,n}$ (Lemma 16). We also show that this simulator is indistinguishable from $\mathrm{Sim}_1$ as long as no randomness of any faked ciphertext is sent as a witness of a proof (Lemma 17). Then, we show that whenever an underivable term is parsed by $\mathrm{Sim}_1$ there is an underivable subterm $t$ that uses a nonce $n$ (Lemma 15) such that $t$ is either in $\mathrm{Sim}_{cf,n}$ not guessable, hence also not in $\mathrm{Sim}_1$, or $t$ is already derivable, if the randomness of a faked ciphertext is already leaked (Lemma 18).

**Witness-knowledge relation.** The cryptographic definition of extractability, does not exclude that the extractor applies also extracts witnesses of ZKPs that have been used as witnesses. Therefore, we need to grant the attacker the oppurtunity to extract witnesses out of ZKPs that are used as a witness in another ZKP. We call this operation extract, and it is defined as follows:

$$\mathrm{extract}(\mathrm{com}(t_1, t_2, t_3)) \quad = \quad t_2$$

**Lemma 14** (No ZK-breaks)**.** *In the hybrid execution with $\mathrm{Sim}_1$, ZK-breaks occur only with negligible probability.*

$$\frac{S \vdash m}{S \vdash_w m} \qquad \frac{S \vdash \mathrm{ZK}(t_2, t_3, t_4)}{S \vdash_w \mathrm{extrWit}(t_3)} \qquad \frac{S \vdash_w \bar{t} \qquad \bar{t} \in \mathbf{T} \qquad F \in \mathbf{C} \cup \mathbf{D}' \qquad \mathrm{eval}_F(\bar{t}) \neq \bot}{S \vdash_w \mathrm{eval}_F(\bar{t})}$$

Figure 13: Deduction rules for the witness knowledge relation: $\mathbf{D}' := \mathbf{D} \cup \{\mathrm{extract}\}$

*Proof.* The simulator $\mathrm{Sim}_1$ does not know the simulation trapdoor, does not have access to a simulation oracle, and only queries the extraction trapdoor once before it halts. We consider the machine $M$ that behaves exactly as $\mathrm{Sim}_1$ and stops before it queries the extraction trapdoor. This machine $M$ serves as a valid adversary against the extractability game, because it never queries the extraction trapdoor. The last part of $\mathrm{Sim}_1$ serves as a valid challenger in the extractability game. A ZK-break occurs if and only if $M$ wins in the extractability game. But we assumed this probability to be negligible; consequently the probability that ZK-breaks occur is negligible. □

**Notation.** Let $S$ be the set of messages that has been sent to the symbolic attacker. We write $M(n)$ for the following event: A bitstring $z$ is sent in a round $\ell$ in which the execution is still alive every bitstring $z$ such that $S_\ell \not\vdash_w \tau(z)$ and $\tau(z)$ contains an underivable subterm that uses the nonce $n$. Definition 15 defines the meaning of an underivable subterms $t$ using a nonces $n$.

**Chosen nonces and encryptions, observably contained terms, being caught, and dead tails.** For the simulator $\mathrm{Sim}_{cf,n}$, $n$ is the *chosen nonce*. We recursively define *chosen encryptions*. An encryption $\mathrm{enc}(t, m, N)$ for which $m$ contains the chosen nonce $n$ as a subterm, is called a *chosen encryption* unless $n$ is in $m$ already contained in a chosen encryptions. In this way, every term has for every occurrence of a chosen nonce at least one chosen encryption. More formally, for a chosen nonce $n$ the term $\mathrm{enc}(t, m, N)$ is a *chosen encryption* if there is a context $C$ such that $C[n] = m$ and there is no pair of contexts $D, D'$ such that $D[\mathrm{enc}(t, D'[n], N)] = m$ and $\mathrm{enc}(t, D'[n], N)$ is already a chosen encryption. Moreover, the randomness symbol $N$ of a chosen encryption $\mathrm{enc}(t, m, N)$ is called *dangerous* from that point on.

Loosely speaking, we say that a $t$ term is (witness) *observably contained* in another term $m$ if changing something in $t$ is always observable in $\beta(t)$ for an attacker that only sees $\beta(m)$ and has access to the extraction trapdoor. Because $\mathrm{Sim}_{cf,n}$ does not simulate the zero-knowledge proofs and only fakes chosen encryptions, it suffices to consider chosen encryptions. More formally, let $t, m \in \mathbf{T}$ be two terms. We say that $t$ is (witness) *observably contained* in $m$ if there are no two context $C, D$ such that $C[\mathrm{enc}(\mathrm{ek}(N), D[t], N')] = m$, $S \not\vdash_w N$, $N, N' \in \mathbf{N}$, and $\mathrm{enc}(\mathrm{ek}(N), D[t], N')$ is a chosen encryption.

We stop the simulation before a treacherous zero-knowledge proof is sent to the attacker. More formally, an invocation of $\beta(t) =: q$ with a zero-knowledge proof $\mathrm{ZK}(\mathrm{crs}(N), t_1, t_2, N')$[21] is called a *catching call* of $\beta$ if the witness $t_1$ observably contains a dangerous randomness symbol $r$ and there is no context $C$ such that $t_1 = C[\mathrm{enc}(\mathrm{ek}(N), m, r)]$. $\mathrm{Sim}_{cf,n}$ halts at the latest point at which the following two conditions are satisfied: $(i)$ a catching call of $\beta$ is about to be performed for the dangerous randomness symbol $r$ and $(ii)$ $\beta$ is called with a chosen encryption with $r$. In the original execution we call the part after a catching call the *dead tail* of the execution.

**The chosen encryption faking simulator $\mathbf{Sim}_{cf,n}$.** For a chosen nonce $n$, we define a faking simulator $\mathrm{Sim}_{cf,n}$ that fakes all chosen encryptions but leaves everything else in the simulation untouched except for halting before a catching call of $\beta$ is computed. We show that the hybrid execution with $\mathrm{Sim}_{cf,n}$ is indistinguishable from the hybrid execution with $\mathrm{Sim}_1$ even if the distinguisher knows the extraction trapdoor; for proving this indistinguishability, we introduce the intermediate simulators $\mathrm{Sim}'_1$, $\mathrm{Sim}'_2$, and $\mathrm{Sim}'_3$.

- We define $\mathrm{Sim}'_2$ like $\mathrm{Sim}_1$, but we change $\beta$ and $\tau$ to use encryption oracles for all honestly generated encryptions instead of computing $A_{\mathrm{enc}}$, $A_{\mathrm{dec}}$, $A_{\mathrm{ek}}$, $A_{\mathrm{dk}}$. More precisely, assume an oracle $\mathcal{O}_{\mathrm{enc}}$ that internally picks $(\mathrm{ek}, \mathrm{dk}) \leftarrow \mathrm{KeyGen}_{\mathrm{enc}}(1^\eta)$ and that responds to three kinds of queries: Upon an $(\mathtt{ek})$-query, it returns ek. Upon a $(\mathtt{enc}, m)$-query, it returns $\mathrm{ENC}(\mathrm{ek}, m)$. Upon a $(\mathtt{dec}, c)$-query, it returns $\mathrm{DEC}(\mathrm{dk}, c)$. $\mathrm{Sim}'_2$ maintains an instance $\mathcal{O}^N_{\mathrm{enc}}$ for each $N \in \mathbf{N}_P$. Then $\mathrm{Sim}'_2$ computes $\beta(\mathrm{ek}(N))$ with $N \in \mathbf{N}_P$ as $\beta(\mathrm{ek}(N)) := \mathcal{O}^N_{\mathrm{enc}}(\mathtt{ek})$. And it computes $\beta(\mathrm{enc}(\mathrm{ek}(N), t, M))$ with $N, M \in \mathbf{N}_P$ as

[21] We stress that $t$ might also be a transformed proof.

$\beta(\mathrm{enc}(\mathrm{ek}(N),t,M)) := \mathcal{O}^N_{\mathrm{enc}}(\mathtt{enc},\beta(t))$. And it computes $\beta(\mathrm{dk}(N)) := \bot$. And in the computation of $\tau(c)$ for $c$ of type ciphertext, the computation of $A_{\mathrm{dec}}(A_{\mathrm{dk}}(r_N),c)$ is replaced by $\mathcal{O}^N_{\mathrm{enc}}(\mathtt{dec},c)$.

- We define $\mathrm{Sim}'_3$ like $\mathrm{Sim}'_2$, except that we replace the oracle $\mathcal{O}_{\mathrm{enc}}$ by an oracle $\mathcal{O}_{\mathrm{fake}}$. That oracle behaves like $\mathcal{O}_{\mathrm{enc}}$, except that upon an $(\mathtt{enc},x)$-query, it returns $\mathrm{ENC}(\mathrm{ek},0^{|x|})$.
- We define $\mathrm{Sim}_{cf,n}$ like $\mathrm{Sim}'_3$, but we change $\beta$ to use signing oracles instead of computing $A_{\mathrm{vk}}, A_{\mathrm{sk}}, A_{\mathrm{sig}}$. More precisely, we assume an oracle $\mathcal{O}_{\mathrm{sig}}$ that internally picks $(\mathrm{vk},\mathrm{sk}) \leftarrow \mathrm{KeyGen}_{\mathrm{sig}}(1^\eta)$ and that responds to two kinds of queries: Upon a $(\mathtt{vk})$-request, it returns vk, and upon a $(\mathtt{sig},m)$-request, it returns $\mathrm{SIG}(\mathrm{sk},m)$. $\mathrm{Sim}_{cf,n}$ maintains an instance $\mathcal{O}^N_{\mathrm{sig}}$ for each $N \in \mathbf{N}_P$. Then $\mathrm{Sim}_{cf,n}$ computes $\beta(\mathrm{vk}(N))$ with $N \in \mathbf{N}_P$ as $\beta(\mathrm{vk}(N)) := \mathcal{O}^N_{\mathrm{sig}}(\mathtt{vk})$. And $\beta(\mathrm{sk}(N))$ with $N \in \mathbf{N}_P$ as $\beta(\mathrm{sk}(N)) := \bot$. And $\beta(\mathrm{sig}(\mathrm{sk}(N),t,M))$ with $N, M \in \mathbf{N}_P$ as $\beta(\mathrm{sig}(\mathrm{sk}(N),t,M)) := \mathcal{O}^N_{\mathrm{sig}}(\mathtt{sig},\beta(t))$.

We need to further characterize the underivable witnesses, i.e., the subterms that are underivable, given a witness $w$ such that $S \not\vdash_w w$.

**Lemma 15** (Witness-underivable subterms)**.** *In a given step of the $\boldsymbol{D}_{\mathrm{trans}}$-hybrid execution of $Sim_{cf}$, let $S$ be the set of messages sent from $\Pi^T$ to $Sim_{cf}$ before.*

*Let $u' \in \boldsymbol{T}$ be the message sent from $Sim_{cf}$ to the protocol in that step. Let $C$ be a context and $u \in \boldsymbol{T}$ such that $u' = C[u]$ and $S \not\vdash_w u$.*

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

$$\begin{aligned} D ::= \Box \mid\ & pair(t,D) \mid pair(D,t) \mid enc(ek(N),D,t) \\ \mid\ & enc(D,t,M) \mid sig(sk(M),D,M') \\ \mid\ & com(D,r,N) \mid com(t,D,M'') \\ \mid\ & ZK(t,t',M) \mid ZK(D,t',M) \mid ZK(t',D,M) \\ \mid\ & garbEnc(D,M) \mid garbSig(D,M) \mid garbZK(D,M) \\ \mid\ & garbZK(M,D) \end{aligned}$$

*with $N \in \boldsymbol{N}_P, M, M' \in \boldsymbol{N}_E, M'' \in \boldsymbol{N}_P \cup \boldsymbol{N}_E, t, t' \in \boldsymbol{T}$ and $u = D[t_{bad}]$ such that $S \not\vdash_w t_{bad}$ and such that one of the following holds:*

1. $t_{bad} \in \boldsymbol{N}_P$
2. $t_{bad} = enc(p,m,N)$ *with* $N \in \boldsymbol{N}_P$
3. $t_{bad} = sig(k,m,N)$ *with* $N \in \boldsymbol{N}_P$
4. $t_{bad} = ZK(x,r,N)$ *with* $N \in \boldsymbol{N}_P$ *and* $crsof(ZK(x,r,N)) = crs(M)$ *and* $M \in \boldsymbol{N}_P$
5. $t_{bad} = sig(sk(N),m,M)$ *with* $N \in \boldsymbol{N}_P$, $M \in \boldsymbol{N}_E$
6. $t_{bad} = crs(N)$ *with* $N \in \boldsymbol{N}_P$
7. $t_{bad} = ek(N)$ *with* $N \in \boldsymbol{N}_P$
8. $t_{bad} = vk(N)$ *with* $N \in \boldsymbol{N}_P$
9. $t_{bad} = sk(N)$ *with* $N \in \boldsymbol{N}_P$
10. $t_{bad} = dk(N)$ *with* $N \in \boldsymbol{N}_P$

*We say that an underivable subterm $t_{bad}$* uses the nonce $N$.

*Proof.* We prove the lemma by structural induction on $u$. Recall that only upon a message $m \in \{0,1\}^*$ from the attacker $\mathrm{Sim}_{cf}$ sends a message $\tau(m)$. There are the following cases:

**Case 1:** " $u \in \{\mathrm{ek}(N),\mathrm{vk}(N),\mathrm{crs}(N),\mathrm{dk}(N),\mathrm{sk}(N)\}$ with $N \notin \mathbf{N}_P$"

Let $u = C(N)$ for $C \in \{\mathrm{ek},\mathrm{vk},\mathrm{crs},\mathrm{dk},\mathrm{sk}\}$. By protocol conditions 1 and 13 each $C$-node has as annotation a nonce from $\mathbf{N}_P$. Therefore $u$ cannot be honestly generated, that means there is some $e \in \{0,1\}^*$ such that $\tau(e) = u$ and $u$ has the form $C(N^e)$. But then $S \vdash u$ contradicting the premise of the lemma.

**Case 2:** " $u \in \{\mathrm{ek}(N),\mathrm{vk}(N),\mathrm{crs}(N),\mathrm{dk}(N),\mathrm{sk}(N)\}$ with $N \in \mathbf{N}_P$"

Then the claim is fulfilled with $D := \Box$ and $t_{bad} = u$.

**Case 3:** "$u = \mathrm{garb}(u_1)$"

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

**Case 4:** "$u = \text{garbEnc}(u_1, u_2)$ or $u = \text{garbSig}(u_1, u_2)$"

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

**Case 5:** "$u = \text{com}(u_1, u_2, N)$"

By protocol condition 13, we know that there is a $c \in \{0,1\}^*$ such that $\tau(c) = \text{com}(\text{crs}(N), t, N^c)$. Hence, the lemma is satisfied by $t_{bad} = \text{com}(\text{crs}(N), t, N^c)$ and $D = \Box$.

By applying $\text{extract}(\text{com}(u_1, u_2, M)) = u_2$, we know that by induction hypothesis there is a $D$ such that $S \vdash D[t_{bad}]$. Consequently, the lemma is satisfied by $D' = \text{com}(u_1, D, M)$.

**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 $D_1 \coloneqq \text{pair}(D, M)$ or $D_2 \coloneqq \text{pair}(M, D)$ is the context for the term $u$ depending on $i$ with the same term $t_{bad}$.

**Case 7:** "$u = \varepsilon$"

This case cannot happen because $S \vdash \varepsilon$, 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 $u_1 = \varepsilon$ 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 $\tau$.

**Case 10:** "$u = N$ with $N \in \mathcal{N}$"

The context $D \coloneqq \Box$ 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 $D = \Box$.

**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 7. Therefore there is a $c \in \{0,1\}^*$ with $\tau(c) = \text{enc}(\text{ek}(M), u_2, N^c) = u$ for some $M \in \mathbf{N}_P$. Apply the induction hypothesis to $u_1$ getting $t_{bad}$ and context $D$ we can define $D' \coloneqq \text{enc}(D, u_2, N^c)$ fulfilling the claim of the lemma with $t_{bad}$.

**Case 14:** "$u = \text{enc}(u_1, u_2, u_3)$ 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 7. Therefore there is an $c \in \{0,1\}^*$ with $\tau(c) = \text{enc}(\text{ek}(M), u_2, N^c) = u$ 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 $D' \coloneqq \text{enc}(\text{ek}(M), D, N^c)$ together with $t_{bad}$ satisfies the lemma.

**Case 15:** "$u = \text{sig}(u_1, u_2, N)$ with $N \in \mathbf{N}_P$"

Use context $D \coloneqq \Box$ and $t_{bad} = u$.

**Case 16:** "$u = \text{sig}(\text{sk}(N), u_1, u_3)$ 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 $D \coloneqq \Box$ and $t_{bad} = u$ proves the claim.

**Case 17:** "$u = \mathrm{sig}(u_1, u_2, u_3)$ and $u_3 \notin \mathbf{N}_P$ and $u_1$ is not of the form $\mathrm{sk}(N)$ with $N \in \mathbf{N}_P$"

Since $u_3 \notin \mathbf{N}_P$ we get by protocol condition 7 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:** "$u = \mathrm{ZK}(u_1, u_2, N)$ with $N \in \mathbf{N}_P$"

Defining $t_{bad} = u$ and $D := \Box$ suffices.

**Case 19:** "$u = \mathrm{ZK}(u_1, u_2, N)$ with $N \in \mathbf{N}_E$"

Since $\tau$ uses randomnessTree for $u_2$ and $N \in \mathbf{N}_E$, we know that $S \nvdash u_1$.

In this case we use the induction hypothesis on $\mathrm{getPub}(u) = u_1$ to get the term $t_{bad}$ and a context $D$. Then using $t_{bad}$ and $D' := \mathrm{ZK}(D, u_2, N)$ satisfies the lemma.

**Case 20:** "$u = \mathrm{ZK}(u_1, u_2, N)$ with $N \in \mathbf{N}_E$"

This case cannot occur because $u$ is not in the range of $\tau$.

**Case 21:** "$u = \hat{f}(t_1, \ldots, t_n, x, w, N)$"

By the definition of $\tau$, we know that $\tau(m) = u$ only if $m$ was already sent by the protocol; hence already $S \vdash u$ holds true and this case cannot occur.

□

We show that for any MZK-safe protocol $\Pi$ and any ppt attacker $A$ the nonce $n$ is not guessable in the execution with the $n$-faking simulator $\mathrm{Sim}_{cf,n}$.

**Lemma 16.** *Let $\Pi$ be an arbitrary MZK-safe protocol and $A$ be an arbitrary ppt machine. In the $\boldsymbol{D}_{\mathrm{trans}}$-hybrid execution with $Sim_{cf,n}$, $\Pi$, $A$, the probability that $M(n)$ happens is negligible in the security parameter.*

*Proof.* Assume that the adversary sends with non-negligible a bitstring $z$ in a round $\ell$ in which the execution is still alive, and $S_\ell \nvdash_w \tau(z)$ and $\tau(z)$ contains a chosen underivable subterm $t$ that uses the chosen nonce $n$. Therefore, only two cases can occur. First, $t$ has not been sent so far, and second $t$ has only been sent along with an encryption. In the second case, however, we know by the construction of $\mathrm{Sim}_{cf,n}$ that the encryption has been faked; in particular, $t$ has never been constructed, i.e., $\beta$ has never been applied to $t$. We show that for every possible underivable subterm that uses $n$ the assumption leads to a contradiction.

**Case** 1: $t_{bad} \in \{n, \mathrm{enc}(p, m, n), \mathrm{sig}(k, m, n), \mathrm{ZK}(x, r, n)\}$
$\mathrm{crs}(n), \mathrm{ek}(n), \mathrm{vk}(n)$ with $M, n \in \mathbf{N}_P$.
If $\beta$ has never been applied on $t_{bad}$, we know by the definition of $\tau$ that $n \notin \mathbf{N}_P$.

**Case** 2: $t_{bad} = \mathrm{sig}(\mathrm{sk}(n), m, M)$ with $n \in \mathbf{N}_P$, $M \in \mathbf{N}_E$.
The same argumentation as for Case 1 holds for $t_{bad} = \mathrm{sig}(\mathrm{sk}(n), m, M)$ and $n \in \mathbf{N}_P$, $M \in \mathbf{N}_E$ with the only difference that the reason why $t_{bad}$, even though unused, is unguessable is that otherwise the attacker would induce a ppt machine that breaks the unforgeability property of the signature scheme.

**Case** 3: $t_{bad} = \mathrm{sk}(n)$ with $n \in \mathbf{N}_P$.
The same argumentation as for Case 1 holds for $t_{bad} = \mathrm{sk}(n)$ and $n \in \mathbf{N}_P$; however, we additionally have to consider the case that the attacker is able to compute the signature key from the verification key. In this case, however, we can use the attacker to break the unforgeability property, which is a contradiction.

**Case** 4: $t_{bad} = \mathrm{dk}(n)$ with $n \in \mathbf{N}_P$.
The same argumentation as for Case 1 holds for $t_{bad} = \mathrm{dk}(n)$ and $n \in \mathbf{N}_P$; however, we additionally have to consider the case that the attacker is able to compute the decryption key from the encryption key. In this case, however, we can use the attacker to break the IND-CCA property, which is a contradiction.

□

In the following proofs, we often consider the $\mathbf{D}_{\text{trans}}$-hybrid execution of a protocol $\Pi^C$ with a simulator $\text{Sim}'_{i,n}$. We extend the $\mathbf{D}_{\text{trans}}$-hybrid execution by letting H-Nodes$'$ output the extraction trapdoors of all CRS and stops before a treacherous zero-knowledge proof is sent. This scenario defines a probability space on traces, which we denote as

$$\Omega_{i,n} := (t, \text{extd}) \leftarrow \text{TH-Nodes}'_{\text{Sim}'_{i,n},\Pi^C}$$

where extd are the extraction trapdoors of all protocol CRS, $\text{Sim}'_{1,n} := \text{Sim}_1$, with the chosen nonce $n$, and $\text{Sim}'_{cf,n} := \text{Sim}_{cf,n}$. Moreover, we write $S_\ell$ for the set of messages that after the $\ell$th round has been sent to $\text{Sim}'_{i,n}$.

**Lemma 17.** *Recall that p is the polynomial of the attacker. Then, the following difference is negligible in the security parameter*

$$\mid Pr[\Omega_{cf,n} : M(n)] - Pr[\Omega_{1,n} : M(n)] \mid$$

*Proof.* Let $\ell \in \mathbb{N}$ be arbitrary but fixed. By protocol condition 7 nonces that are used in $\Omega_{1,n}$ as randomness for encryptions calls can only be further used in the witness of a zero-knowledge proof. However, for chosen encryptions, we abort before the attacker learns both, the faked ciphertext and the proof about the faked ciphertext. Since only chosen encryptions are faked $\Omega_{1,n}$ is indistinguishable from $\Omega_{2,n}$ even for distinguishers that know the extraction trapdoors extd.

By the IND-CCA property of the encryption scheme (implementation condition 8) $\Omega_{2,n}$ is indistinguishable from $\Omega_{3,n}$ even for distinguishers that know the extraction trapdoors extd.

By protocol condition 7 nonces that are used in $\Omega_{3,n}$ as randomness for signing calls can not be used anywhere else, in particular not as a witness in a zero-knowledge proof. Therefore, $\Omega_{3,n}$ is indistinguishable from $\Omega_{cf,n}$ even for distinguishers that know the extraction trapdoors extd.

Since $M(n)$ is efficiently computable for a machine that has access to an extraction oracle, the probability that $M(n)$ holds can only differ by a negligible amount in $\Omega_{1,n}$ and $\Omega_{cf,n}$. □

Finally, we are able to show that all invocations of $\tau$ and $\tau^*$ in $\text{Sim}_1$ adhere the witness knowldge relation.

**Lemma 18** ($\text{Sim}_1$ adheres $\vdash_w$)**.** *For any invocation $\tau(m)$ of $\tau$ or $\tau^*(m)$ of $\tau^*$ in the $\mathbf{D}_{\text{trans}}$-hybrid execution of $\text{Sim}_f$, 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 $S \vdash_w \tau(m)$ or $S \vdash_w \tau^*(m)$, respectively.*

*Proof.* Assume that with non-negligible probability the attacker sends a bitstring $z$ in a round $\ell$ such that $S_\ell \nvdash_w \tau(z)$. Applying Lemma 15, we know that $\tau(z)$ has the following underivable subterms:

**Case** 1: $t_{bad} \in \{N, \text{enc}(p, m, N), \text{sig}(k, m, N),$
$\text{ZK}(x, r, N), \text{crs}(N), \text{ek}(N), \text{vk}(N)\}$ with $M, N \in \mathbf{N}_P$.
If $t_{bad}$ has not been sent to the attacker so far, $\beta(t_{bad})$ has not been called so far. By definition of $\tau$ (resp., $\tau^*$), however, we then have $N, M \notin \mathbf{N}_P$.

If $t_{bad}$ is contained in a term that has already been sent to the attacker, $t_{bad}$ can only have occurred below an encryption. Assuming that $N$ is the chosen nonce, we are either in the living part of the execution or in the dead tail. If we are in the dead tail, because of the protocol condition 7 the attacker already witness-knows the plaintext of a chosen encryption. By the definition of chosen encryptions, we know that the attacker then also witness-knows $t_{bad}$, which is a contradiction.

If $z$ has been sent in the living part of the execution, we know by Lemma 16 that in the execution with $\text{Sim}_{cf,N}$ the attacker cannot send a bitstring $z$ that such that $\tau(z)$ contains an underivable subterm $t_{bad}$, which uses this chosen nonce $N$, with more than negligible probability. By Lemma 17, we know that the same property holds for $\text{Sim}_1$. This property, however, is a contradiction to the assumption that $z$ has been sent with non-negligible probability.

**Case** 2: $t_{bad} = \text{sig}(\text{sk}(N), m, M)$ with $N \in \mathbf{N}_P$, $M \in \mathbf{N}_E$.
The same argumentation as for Case 1 holds for $t_{bad} = \text{sig}(\text{sk}(N), m, M)$ and $N \in \mathbf{N}_P$, $M \in \mathbf{N}_E$ with the only difference that the reason why $t_{bad}$, even though unused, is unguessable is that otherwise the attacker would induce a ppt machine that breaks the unforgeability property of the signature scheme.

**Case** 3: $t_{bad} = \mathrm{sk}(N)$ with $N \in \mathbf{N}_P$.
The same argumentation as for Case 1 holds for $t_{bad} = \mathrm{sk}(N)$ and $N \in \mathbf{N}_P$; however, we additionally have to consider the case that the attacker is able to compute the signature key from the verification key. In this case, however, we can use the attacker to break the unforgeability property, which is a contradiction.

**Case** 4: $t_{bad} = \mathrm{dk}(N)$ with $N \in \mathbf{N}_P$.
The same argumentation as for Case 1 holds for $t_{bad} = \mathrm{dk}(N)$ and $N \in \mathbf{N}_P$; however, we additionally have to consider the case that the attacker is able to compute the decryption key from the encryption key. In this case, however, we can use the attacker to break the IND-CCA property, which is a contradiction.

□

In order to be able to prove that ZK-failures cannot happen with more than negligible probability, we first need to show that the relaxed symbolic extraction $\mathbf{SymbExtr}'$ always proceeds. The relaxed symbolic extraction only checks whether the resulting candidate for a witness $w$ is witness-derivable, i.e., $S \vdash_w w$.

**Lemma 19** (No quasi-extraction failures)**.** *In a $\boldsymbol{D}_{\mathrm{trans}}$-hybrid execution with the simulator $Sim_1$ the following holds: A quasi-extraction failure only happens with negligible probability.*

*Proof.* Assume a quasi-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. Therefore, $z$ 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 $\tau$ would not invoke the symbolic extraction). Let $N \in \mathbf{N}_P$ be defined by $\mathrm{crs}(N) = \tau(A_{\mathrm{crs}}(z))$. Let $m_x := A_{\mathrm{getPub}}(z)$, $x := \tau(m_x)$, $m_w := E(m_x, z, \mathrm{extd}_N)$[22]. Let $S$ denote the set that the protocol already sent to the simulator in this execution.

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

1. $(x, w) \notin R^{\mathrm{sym}}_{\mathrm{adv}}$
2. $S \nvdash_w w$
3. $(x, w) \in R^{\mathrm{sym}}_{\mathrm{adv}}$ and $S \vdash_w w$ but $\mathbf{SymbExtr}'(S, x) = \bot$

We prove for each case that it only occurs with negligible probability, which leads to a contradiction to the assumption that quasi-extraction failures occur with non-negligible probability.

**Case** 1: $(x, w) \notin R^{\mathrm{sym}}_{\mathrm{adv}}$.
This constitutes a ZK-Break, which is a contradiction to Lemma 14.

**Case** 2: $S \nvdash_w w$.
By Lemma 18 this case only happens with negligible probability.

**Case** 3: $(x, w) \in R^{\mathrm{sym}}_{\mathrm{adv}}$ and $S \vdash_w w$ but $\mathbf{SymbExtr}'(S, x) = \bot$.
By definition of $\mathbf{SymbExtr}'$ the symbol $\bot$ is only returned if there is no $w$ such that $(x, w) \in R^{\mathrm{comp}}_{\mathrm{adv}}$ and $S \vdash_w w$. So this case cannot occur.

□

**No extraction-failure.** In order to conclude that an extraction-failure does not happen in Sim, we first show that in $\mathrm{Sim}_f$ no extraction-failure happens. And then, we apply the indistinguishability of the knowledge traces (Lemma 13) of Sim and $\mathrm{Sim}_f$.

**Lemma 20** (No extraction failures - FMZK model)**.** *For the FMZK model in a hybrid execution of $Sim_f$ holds: An extraction failure can only occur with negligible probability.*

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

*Proof.* Assume an extraction failure occurs with non-negligible probability. Then, a bitstring $z$ has been sent by the attacker such that $x = \tau(A_{\text{getPub}}(z))$ and $\mathbf{SymbExtr}(S, x) = \bot$ but $\mathbf{V}(A_{\text{getPub}}(z), z, \text{crs}) = 1$. By Lemma 19 and Lemma 13, we know that in $\text{Sim}_f$ quasi-extraction failures only happen with negligible probability. Consequently, there is a $w \in \mathbf{SymbExtr}'(x)$ such that $\bot \neq w$, and by definition of $\mathbf{SymbExtr}'$ also $S \vdash_w w$ holds true. We observe that $S \vdash_w w$ if and only if there is a transformation $C[\bullet_1, \ldots, \bullet_n]$ such that for some $t_1, \ldots, t_n \in S$ we have $C[t_1, \ldots, t_n] = w$, where $C$ is a context using constructors and destructors from $\mathbf{C} \cup \mathbf{D} \cup \{\text{extract}\}$.

Let $D$ be a context such that $D[\text{ZK}(t_1, r_1, N_1), \ldots, \text{ZK}(t_n, r_n, N_n)] = \text{ZK}(t, r, N^z) = \tau(z)$ and $\text{ZK}(t_1, r_1, N_1)$, $\ldots, \text{ZK}(t_n, r_n, N_n) \in S$. We show that $D$ is with overwhelming probability zero-knowledge preserving.

Next, we show the following: for every proof $z$ such that $\mathbf{V}(A_{\text{getPub}}(z), z, \text{crs}) = 1$ that the attacker sends, there is a $w \in \mathbf{SymbExtr}'(S, \tau(A_{\text{getPub}}(z)))$ such that $w \neq \bot$, a ZK-transformation $f_{f_1,f_2}$ over $\mathbf{D}, \mathbf{D} \cup \{\text{extract}\}$, and there are $\text{ZK}(t_2^{(i)}, t_3^{(i)}, t_4^{(i)}) \in S$, $i \in [n]$, such that $\tau(A_{\text{getPub}}(z)) = f_1(t_2^{(1)}, \ldots, t_2^{(n)})$ and $w = f_2(t_3^{(1)}, \ldots, t_3^{(n)})$.

By Lemma 13, we know that the attacker can also in the hybrid execution with Sim construct such a message with non-negligible probability. The soundness property of the zero-knowledge proof system implies that $(x, w) \in R_{\text{adv}}^{\text{comp}}$ holds true with overwhelming probability. Lemma 11 in turn implies that $(x, w) \in R_{\text{adv}}^{\text{sym}}$. Therefore, there are witnesses $w_1, \ldots, w_n$ such that the transformation that the attacker applied produces a valid proof for $(x, w)$. We further know that for the statement $x_1, \ldots, x_n$ there is at least one tuple of witnesses $w_1', \ldots, w_n'$ such that the transformation of the attacker does not produce a valid witness $(x, w') \notin R_{\text{adv}}^{\text{sym}}$.[23] But then, we can construct a machine $M$ that emulates the entire execution (with Sim) twice and in one case uses $(w_1, \ldots, w_n)$ for the proofs and in the other case uses $(w_1', \ldots, w_n')$. $M$ outputs 1 if the transformed proof is true and 0 if the transformed proof is wrong. If the probability of the attacker to produce such an above-described transformation is non-negligible, the probability of $M$ distinguishing honest proofs from simulated proofs is non-negligible as well. This, however, contradicts the zero-knowledge property.

Next, we show that every such context $D$ that is zero-knowledge preserving does not use extract. We show by induction on $n$ that for all contexts $D$ of depth $n$ there is a context $\tilde{D}$ such that eval $D[M] =$ eval $\tilde{D}[M]$ for all $M \in T$. The lemma directly following from this statement.

For $n = 0$ the context cannot contain extract. For $n > 0$ assume that the statement holds for all $n' < n$. Then, we either have $D[\bullet] \neq \text{extract}[\tilde{D}_1[\bullet]]$ or $D[\bullet] = \text{extract}[\tilde{D}_1[\bullet]]$. In the first case, the statement for $n$ directly follows from the induction hypothesis. In the second case, observe that the statement cannot imply that the witness is a commitment, hence $D[\text{com}(c, t, r)] \neq \bot$ (for any $c, t, r$) and $D[t'] = \bot$ for any $t'$ that is not a commitment. Hence, $D$ is not zero-knowledge preserving. □

**Lemma 21** (No extraction failures - CMZK model)**.** *In the CMZK model in a hybrid execution of $\text{Sim}_f$ holds: An extraction failure can only occur with negligible probability.*

*sketch.* By Lemma 20, we know that an extraction failure up to malleability can only occur with negligible probability. So, the only events that we did not exclude yet are the successful verification of a proof $\hat{z}'$ such that $\tau(\hat{z}') = \text{setPub}(\text{ZK}(t, r), t\{c'/c\})$, where $\text{ZK}(t, r) = \tau(\hat{z})$ and $c' = \text{com}(\text{com}(crs, f(m), r')) = \text{applyF}(c, x)$ (for some $r' \in \mathbf{N}$) and $c = \text{com}(crs, m, r)$. We call this event $Q$.

Let $vk$ be the verification key from the CRS. Recall that the statement is

$$(x, w) \in R \vee (\hat{\text{ver}}_{\text{sig}}(\tau(vk), sig, x') = t \wedge x = T_x(x') \wedge T_x \in \mathcal{T}).$$

Since the proof is simulated by $\text{Sim}_f$ and by the extractability property, we know that with overwhelming probability the extracted signature $s$ successfully verifies with $vk$. However, this is a contradiction to applyF $\notin \mathcal{T}$and to the the universal unforgeability of the signature scheme. Hence, $Q$ could not have happened with non-negligible probability.

This argument can easily be generalized for any context that visibly applies applyF. □

**The indistinguishability of Sim.** Finally, applying Lemma 20 and Lemma 11, we are able to conclude that Sim with the $\mathbf{D}_{\text{trans}}$-transparent hybrid execution is indistinguishable from the real computational execution.

We show that given an attacker $\mathcal{A}$, a protocol $\Pi_{\mathsf{p}}$, a polynomial $p$ the traces of the hybrid execution with $M$ and Sim are indistinguishable from the traces of the computational execution Nodes with the implementation $A$ and the attacker $\mathcal{A}$.

[23] If there would not be such a witness, then there would be a transformation that behaves on all zero-knowledge proofs with the statements $x_1, \ldots, x_n$ as the transformation of the attacker and on all other statements simply outputs a trivial true proof.

**Lemma 22.** *Sim is indistinguishable for $\boldsymbol{M}, \Pi, A, E$ and for every polynomial $p$.*

*Proof.* With overwhelming probability it's the case that the messages $r_N$, $\mathrm{enc}(\ldots, N)$, $\mathrm{sig}(\ldots, N)$, $\mathrm{ZK}(\ldots, N)$ are different for all $N \in \mathbf{N}_P$. Furthermore, by Lemma 20, we know that no extraction-failures happen with Sim with overwhelming probability; hence, Sim does not abort. So we can restrict to that case for indistinguishability.

For proving indistinguishability of Sim for $\mathbf{M}, \Pi, A, E$ we need to show the following claims.

**Claim 22.1.** *In the $\hat{F}$-transparent hybrid execution of Sim holds $\forall m \in \{0,1\}^* : \beta(\tau(m)) = m$.*

**Claim 22.2.** *In the $\hat{F}$-transparent hybrid execution, for any term $t$ stored at a node $\nu$ holds, $\beta(t) \neq \bot$.*

**Claim 22.3.** *For all terms $t$ that occur in the $\hat{F}$-transparent hybrid execution holds $\tau(\beta(t)) = t$.*

**Claim 22.4.** *In the $\hat{F}$-transparent hybrid execution, at any computation node $\nu = \nu_i$ with constructor or destructor $F$ and arguments $\tilde{\nu_1}, \ldots, \tilde{\nu_n}$. Let $t_j = f_i^C(\tilde{\nu_j})$. Then holds: $\beta(eval_F(\underline{t})) = A_F(\beta(t_1), \ldots, \beta(t_n))$.*

In the proof we will only use the first and the last claim. Claim 22.1 and Claim 22.3 follow from the definition of $\tau$ and $\beta$ since $\tau$ first checks whether a bitstring is the result of a previous $\beta$ call on some term $t$. If so, $\tau$ outputs this term $t$. The argumentation can be carried out via an induction over the sum of the depths of the recursion calls to $\beta$ and $\tau$ and by performing an extensive case distinction over the definition of $\beta$ and $\tau$.

Claim 22.2 can be shown via an induction over the recursion depth of $\beta(t)$. In the base case, $t$ can only be a nonce. Since $A_t$ implements by implementation condition 3 a uniform distribution on $\{0,1\}^k$ we have $\Pr[m \leftarrow A_t : m = \bot] = 0$. In the induction step, first observe that $\underline{t}$ cannot contain $\bot$ by the definition of a $\hat{F}$-transparent hybrid execution. Consequently, either $t = f(\underline{t})$ for a constructor $f \in \mathbf{C}$ or $t = \hat{f}(\underline{t})$ for a protocol ZK-transformation $f \in \hat{F}$. We stress that $f$ cannot be an attacker ZK-transformation, i.e., $f \in \mathbf{D}_{\mathrm{trans}} \setminus \hat{F}$, since transformed proofs are parsed as attacker-generated ZK proofs.

If $t = f(\underline{t})$ and $f$ is a constructor, we know by the induction hypothesis that all possible $\beta(\underline{t})$ calls do not return $\bot$. By the implementation conditions and the protocol conditions, we therefore conclude that in this case the probability that $\beta(f(\underline{t})) = \bot]$ is negligible.[24] If $t = \hat{f}(\underline{t})$ and $f \in \hat{F}$ is a protocol ZK-transformation, then we know by the strong derivation privacy [CKLM12] and the correctness and zero-knowledge property of the implementation of ZK proofs (Definition 14) that with overwhelming probability $A_{\hat{f}}$ does not fail if applied to verifiable proofs. Furthermore, we know by protocol conditions 17 that ZK-transformations are only applied to verifiable proofs.

The last Claim 22.4 follows from the definition of $\beta$ since for all calls $\beta(F(\underline{t}))$ in which the arguments are the result of a node $\beta(F(\underline{t}))$ is defined as $A_F(\underline{\beta(t_1)})$.[25] The interesting case is $F = \mathrm{ver}_{\mathrm{zk}}$. At that point we need to show that for an honestly generated $\overline{\mathrm{ZK}}$ proof with a statement $x$ and the witness $w$ we have that $(x, w) \in R_{\mathrm{hon}}^{\mathrm{sym}}$ if and only if its implementation is valid, i.e., $(\beta(x), \beta(w)) \in R_{\mathrm{hon}}^{\mathrm{comp}}$. By the Dolev-Yaoness of Sim, we know that $(x, w) \in R_{\mathrm{hon}}^{\mathrm{sym}}$ with overwhelming probability. By Lemma 11, we know that then also $(\beta(x), \beta(w)) \in R_{\mathrm{hon}}^{\mathrm{comp}}$.

Fix the randomness of the protocols nondeterministic nodes, the nonces, and the adversaries random tape.

Let $\nu_i, f_i$ be the nodes and functions as in the computational trace and $\nu_i^C, f_i^C$ be the ones in the hybrid trace. Let $S_i$ be the state of the adversary before execution of the $i-th$ node and $S_i^C$ the corresponding state of the adversary in the hybrid execution. We show by induction in $i$ that $\nu_i = \nu_i^C$, $f_i = \beta \circ f_i^C$, and $S_i = S_i^C$.

Base case $i = 1$. The adversary $E$ is in its starting configuration,i.e. $s_1 = s_1^C$, the node mapping function $f$ is totally undefined, $f_1 = f_1^C =$ and the current node is the root of the protocol, $\nu_1 = \nu_1^C$.

Induction hypothesis: For all $j \leq i$ holds $\nu_j = \nu_j^C$, $f_j = \beta \circ f_j^C$ and $s_j = s_j^C$.

Induction step: $i \to i + 1$

Distinguish by the type of the nodes.

1. If $\nu_i = \nu_i^C$ is a computation node annotated with constructor or destructor.

   Let $F/n$ be the annotated constructor or destructor and $\tilde{\nu_1}, \ldots, \tilde{\nu_n}$ be the annotations of $\nu_i$ and $\tilde{\nu_1^C}, \ldots, \tilde{\nu_n^C}$ the of $\nu_i^C$. By induction hypothesis follows that $\tilde{\nu_k} = \tilde{\nu_k^C}$ for $1 \leq k \leq n$. So let $\tilde{m_i} = f_i(\tilde{\nu_i})$, denote the vector $(\tilde{m_1}, \ldots, \tilde{m_n})$ by $\underline{\tilde{m}}$. Analogue are $\tilde{t_i}$, $\underline{\tilde{t}}$ defined for $\tilde{\nu_i^C}$ and $f_i^C$.

[24] This can be shown by an extensive case distinction on $f$.

[25] Again, this claim can be shown by an induction over the recursion depth of the $\beta$ call and by an extensive case distinction.

In the hybrid execution we compute $\text{eval}_F(\tilde{\underline{t}})$. By Claim 22.4 holds $\beta(\text{eval}_F(\tilde{\underline{t}})) = A_F(\beta(t_1), \ldots, \beta(t_n))$. Using the definition of $t_j$ we get $\beta(t_j) = (\beta \circ f_i^C)(\nu_j^C)$. Now we can apply the induction hypothesis and replace $(\beta \circ f_i^C)$ by $f_i$ and $\nu_j^C$ by $\nu_j$, i.e. $(\beta \circ f_i^C)(\nu_j^C) = f_i(\nu_j)$. All together leads to the equality $\beta(\text{eval}_F(\tilde{\underline{t}})) = A_F(\tilde{\underline{m}})$. The right hand side is the outcome of the computational execution of the node $\nu_i$. So the left side is defined if and only if the right side is, and we get $\nu_{i+1}, \nu_{i+1}^C$ is the yes-successor of $\nu_i$ if the term is defined and the no-successor otherwise, i.e. $\nu_{i+1} = \nu_{i+1}^C$. Since $f_{i+1}(\nu_i) = A_F(\tilde{\underline{m}})$ and $f_{i+1}^C(\nu_i^C) = \text{eval}_F(\tilde{\underline{t}})$ we have seen that $f_{i+1} = (\beta \circ f_{i+1}^C)$ still holds. In both executions the adversary does not learn anything new, i.e. $S_{i+1} = S_i = S_i^C = S_{i+1}^C$.

2. If $\nu_i = \nu_i^C$ is a computation node annotated with nonce.

   Let $N$ be the annotated nonce. Since $N \in \mathbf{T}$ we get $\text{eval}_N() = N \neq \bot$, so $\nu_{i+1}^C$ is the yes-successor of $\nu_i^C$. In the computational case always the yes-successor is taken, therefore $\nu_{i+1} = \nu_{i+1}^C$. The adversary is not activated, so $S_{i+1} = S_i = S_i^C = S_{i+1}^C$. For the functions $f_{i+1}$ and $f_{i+1}^C$ we get $f_{i+1}(\nu_i) = r_N = \beta(N) = \beta(f_{i+1}^C(\nu_i^C))$. So the claim is true for $i+1$.

3. If $\nu_i = \nu_i^C$ is a input node.

   Input nodes have a unique successor, so $\nu_{i+1} = \nu_{i+1}^C$ is this successor. In the computational as in the hybrid case the adversary is asked for input. Since $S_i = S_i^C$, i.e. in both cases the adversary has the same state, both computations are equal, so the resulting state $S_{i+1}$ and $S_{i+1}^C$, too, and the adversary responds with $m$ in the computational and $m^C$ in the symbolic case, such that $m = m^C$. In the hybrid case the simulator forwards $t^C := \tau(m^C)$ to the protocol execution. $f_{i+1}(\nu_i) := m$ and $f_{i+1}(\nu) := f_i(\nu)$ for $\nu \neq \nu_i$ by definition in the computational case and $f_{i+1}^C(\nu_i) := t^C$ and $f_{i+1}^C(\nu) = f_i^C(\nu)$ for $\nu \neq \nu_i$ in the hybrid case. So by induction hypothesis $f_{i+1}(\nu) = (\beta \circ f_{i+1}^C)(\nu)$ for $\nu \neq \nu_i$ and $(\beta \circ f_{i+1}^C)(\nu_i) = \beta(t^C) = \beta(\tau(m^C)) \overset{(*)}{=} m^C = m = f_{i+1}(\nu_i)$, where $(*)$ holds because of Claim 22.1. Therefore the invariant $f_{i+1} = (\beta \circ f_{i+1}^C)$ holds, too.

4. If $\nu_i = \nu_i^C$ is a output node.

   An output node has a unique successor, so $\nu_{i+1} = \nu_{i+1}^C$ is the unique successor of $\nu_i$. The functions $f_i$ and $f_i^C$ doesn't change in this step, so by induction hypothesis holds $f_{i+1} = f_{i+1}^C$, too. Let $\tilde{\nu}$ be the node in the annotation of $\nu_i$. In the computational case $m := f_i(\tilde{\nu})$ is sent to the adversary $E$. In the hybrid case the simulator receives $t^C := f_i^C(\tilde{\nu})$ and forwards $m^C := \beta(t^C)$ to the adversary. By induction hypothesis ($f_i = (\beta \circ f_i^C)$ follows $m^C = m$, and therefore is the state of the adversary the same afterwards, i.e. $S_{i+1} = S_{i+1}^C$.

5. If $\nu_i = \nu_i^C$ is a control node.

   Let $l$ be the annotated out-metadata of node $\nu_i$. In the computational case $l$ is sent to the adversary $E$ and in the hybrid case it's sent to Sim which forwards it to $E$. Since $S_i = S_i^C$ the computation of the adversary is the same and therefore is $S_{i+1} = S_{i+1}^C$ and $E$ returns $l'$ in both cases. So the chosen successor node is the same in both cases, i.e. $\nu_{i+1} = \nu_{i+1}^C$. Since the functions $f_i$ and $f_i^C$ stay untouched, follows $f_{i+1} = f_{i+1}^C$.

6. If $\nu_i = \nu_i^C$ is a nondeterministic node.

   The adversary is in both cases not invoked, so $S_{i+1} = S_i = S_i^C = S_{i+1}^C$. The same holds for the mappings $f_{i+1}$ and $f_{i+1}^C$. Since we fixed the random of nodes and $\nu_i = \nu_i^C$ both choose the same successor and therefore $\nu_{i+1} = \nu_{i+1}^C$.

So $\text{Nodes}^p_{\mathbf{M},A,\Pi_\text{p},E} = \text{H-Nodes}_{\mathbf{M},\Pi_\text{p},\text{Sim}}(k)$ if $r_N$, $\text{enc}(\ldots, N)$, $\text{sig}(\ldots, N)$, $\text{ZK}(\ldots, N)$ are different for all $N \in \mathbf{N}_P$. Since this is the case with overwhelming probability they are indistinguishable. □

**Theorem 7.** *Any computational implementation of the MZK-safe model* $\boldsymbol{M}$ *(see Section A) that satisfies the implementation conditions for MZK-safe protocols (see Section 4.2) is computationally sound for the class of MZK-safe protocols (see Section 4.1).*

*Proof.* By Lemma 10 and Lemma 13 we get by transitivity of indistinguishability that Sim is DY. By Lemma 22 we conclude that Sim is indistinguishable for $\mathbf{M}, \Pi, A, E$ and for every polynomial $p$ and every MZK-safe protocol, where $A$ satisfies the implementation conditions for MZK-safe protocols (see Section 4.2). Consequently, Sim is a good simulator with respect to the $\hat{F}$-transparent hybrid execution, and Lemma 7 implies that any implementation $A$ of the MZK-safe model satisfying the implementation conditions is computationally sound for the class of MZK-safe protocols. □

# F Self-Monitoring

In this section, we prove that the CMZK model allows for self-monitoring. Most of the results are analogous to the the original work on self-monitoring [BMR14a]. Hence, we concentrate on the parts where the proofs differ.

**Theorem 8.** *Let* $\mathsf{M}$ *be the CMZK symbolic model, and let MZK-safe' be the class of uniformity-enforcing bi-protocols. A is also a computationally sound implementation with respect to equivalence properties.*

*Proof.* The theorem directly follows from the symbolic and computational self-monitoring property of the branching and the knowledge monitor (see below) and Theorem 1 from the previous work [BMR14a]. □

## F.1 Branching monitor

For the branching monitor, we discuss the extensions in the construction of the branching monitor and the steps that are different from the original work on self-monitoring [BMR14a].

### F.1.1 Constructing the monitor

The branching monitor is constructed exactly as in the original work on self-monitoring [BMR14a] except that the concrete tests, i.e., the algorithm construct shape. The set of binary and unary destructors is given by the additional destructor rules below and the destructor rules for the CMZK model.

**Additional destructor rules.** We do not need more additional destructor rules than in the original work on self-monitoring [BMR14a]. We only need the additional virtual destructor $\mathrm{dec}'$, which is not a real destructor but a lookup for protocol-created ciphertexts with the public key of the attacker.

**Construct Shape.** The algorithm construct shape works as in the original work on self-monitoring except that the additional rules for the parsing function $\tau$ in the CoSP simulator for trace properties are additionally translated to constructor and destructor calls.

### F.1.2 Extended symbolic model

The extended symbolic model goes along the lines of the previous work. We extend the message type as in the previous work. We use the extended symbolic model to show that the branching monitor satisfies computational and symbolic self-monitoring.

**Extended message type.** The extension to the grammar of the message types is the same as in the original work on self-monitoring [BMR14a]. We only need the additional constructors skofvk /1, plaintextOf /2, nonceOf /1.

### F.1.3 Symbolic and computational self-monitoring

The symbolic self-monitoring property directly follows from the construction of the branching monitor since we only use operations that are also available to the symbolic attacker.

**Lemma 1.** *[Symbolic self-monitoring of the branching monitor] Let* $\Pi$ *be a bi-protocol from the protocol class* $MZK - safe$*, and* $\mathsf{M}$ *be the symbolic model from Section A. The branching monitor satisfies symbolic self-monitoring (see Definition 13).*

*Proof.* The proof is verbatim the same as in the previous work [BMR14a, Lemma 5 and 6]. □

**Lemma 2.** *[Computational self-monitoring of the branching monitor] The branching monitor satisfies computational self-monitoring (see Definition 13).*

*Proof.* The proof of the computational self-monitoring property from the previous work [BMR14a, Lemma 9] applies verbatim to our setting. The only part in the proof that is specific to the symbolic model and its implementation is Claim 4 where it is shown that the difference of the extended symbolic model and the symbolic model is insignificant. Since our extended symbolic model uses the same extension as in the previous work, the same proof for Claim 4 applies for the CMZK model. □

## F.2 Knowledge monitor

The knowledge monitor is constructed as in the previous work. The only difference are the adjustments to the algorithm construct shape, which are the same as for the branching monitor.

**Lemma 3.** *[Symbolic self-monitoring of the knowledge monitor] Let* $\Pi$ *be a bi-protocol. Let* $i \in \mathbb{N}$*. Let* $\Pi'_i$ *be the self-monitor for* $\Pi_i$*. For all* $i \in \mathbb{N}$ *the following holds. If there is an attacker strategy such that in* $\Pi'_i$ *the event* $\mathsf{bad\text{-}knowledge}$ *occurs but in* $\Pi'_{i-1}$ *the event* $\mathsf{bad}$ *does not occur and* $\Pi_{i-1}$ *is symbolically indistinguishable, then* $\Pi_i$ *is symbolically distinguishable because of knowledge.*

*Proof.* The symbolic self-monitoring property immediately follows from the previous work. The reasoning is similar to the proof of the symbolic self-monitoring of the branching monitor. See Lemma 10 of [BMR14a] for more details. □

### F.2.1 Computational self-monitoring

Most of the proof for computational self-monitoring is about reducing the problem to a simpler case. These reductions are generic and apply verbatim from the previous work [BMR14a].

**Re-using the indistinguishability of the faking simulator.** In the previous work, we show [BMR14a, Lemma 12] that the indistinguishability of the traces of the execution and the hybrid execution with the faking simulator $Sim_f$ implies the indistinguishability of the attacker-views of the execution and the hybrid execution with the faking simulator $Sim_f$. The proof does not use any specific property of the simulator $Sim$ or the faking simulator $Sim_f$. Hence, the same results also holds in our case.

**Uniqueness of a symbolic operation.** We can show that whenever two shapes, i.e., resulting symbolic operations from the algorithm construct shape, are different, the corresponding messages are different as well. This uniqueness of the symbolic operations follows along the same lines as in the previous work [BMR14a, Lemma 15] since the construction of construct shape is analogous.

**Perfect distinguishability if no alarm is raised.** Since we can re-use the indistinguishability of the faking simulator (see above), all honest encryptions are faked, all honest proofs are simulated and all honest commitments for which the unveil information is not public are faked. In this setting, for all $i \in \mathbb{N}$, if after $i-1$ steps the left and right bi-protocol are perfectly indistinguishable, given that the self-monitor does not raise an alarm, then also after $i$ steps the left and right bi-protocol are perfectly indistinguishable, as long as the self-monitor does not raise an alarm.

The proof is performed by a big case distinction over the type of symbolic operation that construct shape outputs after $i$ steps (see Lemma 21 in [BMR14a]). Three cases are different from the previous work:[26] (*i*) the symbolic operation represents a CRS, in which case all information is public; (*ii*) the symbolic operation represents a commitment, in which case all commitments are either faked or their unveil information is known; (*iii*) the symbolic operation represents a zero-knowledge proof, in which case either the proof is simulated or does not contain secret witnesses or is attacker-generated.

**Lemma 4.** *[Computational self-monitoring of knowledge monitor] The parametric CoSP protocol* $f_{\mathsf{bad\text{-}knowledge},\Pi}$ *satisfies computational self-monitoring (see Definition 13).*

[26] Actually, we use in the proof the symbolic distinguishability for this proof, but this follows by the virtually the same arguments as in Lemma 19 of [BMR14a].

*Proof.* With this result at hand, the computational self-monitoring follows with exactly the same argumentation as in Lemma 23 of [BMR14a]. □

# G Postponed Definitions

## G.1 Time-insensitive indistinguishability

We rely on the notion of termination-insensitive computational indistinguishability (tic-indistinguishability) [Unr11] to capture that two protocols are indistinguishable in the computational world. In comparison to standard computational indistinguishability, tic-indistinguishability on one hand does not require the protocols (formally, the interactive machines) to be polynomial-time, but on the other hand it solely considers decisions that were made after a polynomially-bounded prefix of the interaction (where, both, the adversary's and the protocol's steps are counted). If after an activation, (say) the second protocol does not output anything within a polynomial numbers of steps, then the pair of protocols will be considered indistinguishable no matter how the first protocol will behave in this case. We write the fact that a machine $M$ terminates after $n$ steps with the output $a$ as $M \Downarrow_n a$.

**Definition 20** (Tic-indistinguishability)**.** *Given two machines $M, M'$ and a polynomial $p$, we write* $\Pr[\langle M \mid M' \rangle \Downarrow_{p(k)} x]$ *for the probability that the interaction between $M$ and $M'$ terminates within $p(k)$ steps and $M'$ outputs $x$. We call two machines $A$ and $B$* termination-insensitively computationally indistinguishable for a machine $E$ ($A \approx^E_{tic} B$) *if for for all polynomials $p$, there is a negligible function $\mu$ such that for all $z, a, b \in \{0,1\}^*$ with $a \neq b$,*

$$\Pr[\langle A(k) \mid E(k,z) \rangle \Downarrow_{p(k)} a] + \Pr[\langle B(k) \mid E(k,z) \rangle \Downarrow_{p(k)} b] \leq 1 + \mu(k).$$

*Here, $z$ represents an auxiliary string. Additionally, we call $A$ and $B$* termination-insensitively computationally indistinguishable ($A \approx_{tic} B$) *if we have $A \approx^E_{tic} B$ for all polynomial-time machines $E$.*

## G.2 Definition of $\mathsf{Mon}(\Pi)$

Formally, we require that the bi-protocols are *uniformity-enforcing*, i.e., when the left and the right protocol of the bi-protocol $\Pi$ take different branches, the attacker is informed. Since taking different branches is only visible after a control node is reached, we additionally require that computation nodes are immediately followed by control nodes.

**Definition 21** (Uniformity-enforcing)**.** *A class* P *of CoSP bi-protocols is* uniformity-enforcing *if for all bi-protocols $\Pi \in$ P:*

1. *Every control node in $\Pi$ has unique out-metadata.*
2. *For every computation node $\nu$ in $\Pi$ and for every path rooted at $\nu$, a control node is reached before an output node.*

All embeddings of calculi the CoSP framework described so far, namely those of the applied pi-calculus [BHU09] and RCF [BMU10], are formalized such that protocols written in these calculi fulfill the second property. We stress that it is straightforward to extend them to fulfill the first property by tagging the out-metadata with the address of the node in the protocol tree.

The self-monitor $\mathsf{Mon}(\Pi)$ of a bi-protocol $\Pi$ behaves like one of the two variants of the bi-protocol $\Pi$, while additionally simulating the opposite variant such that $\mathsf{Mon}(\Pi)$ itself is able to detect whether $\Pi$ would be distinguishable. (For instance, one approach to detect whether $\Pi$ is distinguishable could consist of reconstructing the symbolic view of the attacker in the variant of $\Pi$ that is not executed by $\mathsf{Mon}(\Pi)$.) At the beginning of the execution of the self-monitor, the attacker chooses if $\mathsf{Mon}(\Pi)$ should basically behave like left($\Pi$) or like right($\Pi$). We denote the chosen variant as $b \in \{\text{left}, \text{right}\}$ and the opposite variant as $\overline{b}$. After this decision,

$\mathsf{Mon}(\Pi)$ executes the the $b$-variant $b(\Pi)$ of the bi-protocol $\Pi$, however, enriched with the computation nodes and the corresponding output nodes of the opposite variant $\overline{b}(\Pi)$.[27]

The goal of the self-monitor $\mathsf{Mon}(\Pi)$ is to detect whether the execution of $b(\Pi)$ would be distinguishable from $\overline{b}(\Pi)$ at the current state. If this is the case, $\mathsf{Mon}(\Pi)$ raises the event bad, which is the disjunction of two events bad-branch and bad-knowledge.

The event bad-branch corresponds to the case that the left and the right protocol of the bi-protocol $\Pi$ take different branches. Since uniformity-enforcing protocols have a control node immediately after every computation node , the attacker can always check whether $b(\Pi)$ and $\overline{b}(\Pi)$ take the same branch. We require (in Definition 13) the existence of a so-called *distinguishing self-monitor* $f_{\mathsf{bad\text{-}branch},\Pi}$ that checks whether each destructor application in $b(\Pi)$ succeeds if and only if it succeeds in $\overline{b}(\Pi)$; if not, the distinguishing self-monitor $f_{\mathsf{bad\text{-}branch},\Pi}$ raises bad-branch.

The event bad-knowledge captures that the messages sent by $b(\Pi)$ and $\overline{b}(\Pi)$ (via output nodes, i.e., not the out-metadata) are distinguishable. This distinguishability is only detectable by a protocol if the constructors and destructors, which are available to both the protocol and the symbolic attacker, capture all possible tests. We require (in Definition 13) the existence of a distinguishing self-monitor $f_{\mathsf{bad\text{-}knowledge},\Pi}$ that raises bad-knowledge in $\mathsf{Mon}(\Pi)$ whenever a message, sent in $\Pi$, would allow the attacker to distinguish $b(\Pi)$ and $\overline{b}(\Pi)$.

**Parameterized CoSP Protocols** For a bi-protocol $\Pi$, we formalize the distinguishing self-monitors $f_{\mathsf{bad\text{-}knowledge},\Pi}$ and $f_{\mathsf{bad\text{-}branch},\Pi}$ with the help of *parameterized CoSP protocols*, which have the following properties: Nodes in such protocols are not required to have successors and instead of other nodes, also formal parameters can be referenced. A parameterized CoSP protocol is intended to be plugged into another protocol; in that case the parameters references must be changed to references to nodes.

**Definition 22** (Self-monitor)**.** *Let $\Pi$ be a CoSP bi-protocol. Let $f_{\mathsf{bad\text{-}knowledge},\Pi}$ and $f_{\mathsf{bad\text{-}branch},\Pi}$ be functions that map execution traces to parameterized CoSP protocols*[28] *whose leaves are either* ok*, in which case they have open edges, or nodes that raise the event* bad-knowledge*, or* bad-branch *respectively.*

*Recall that nodes $\nu$ of $\Pi$ have bi-references (as defined in Definition 3) consisting of a left reference (to be used in the left protocol) and a right reference. We write left($\nu$) for the node with only the left reference and right($\nu$) analogously. For each node $\nu$ in $\Pi$, let $tr_\nu$ be the execution trace of $\Pi$ that leads to $\nu$, i.e., the list of node and edge identifiers on the path from the root of $\Pi$ to $\nu$, including $\nu$. The* self-monitor $\mathsf{Mon}(\Pi)$ *protocol is defined as follows:*

*Insert before the root node a control node with two copies of $\Pi$, called the left branch (with $b := left$) and the right branch (with $b := right$). Apply the following modifications recursively for each node $\nu$, starting at the root of $\Pi$:*

1. *If $\nu$ is a* computation node *of $\Pi$, replace $\nu$ with $f_{\mathsf{bad\text{-}branch},\Pi}(b, tr_\nu)$. Append two copies left($\nu$) and right($\nu$) of the the computation node $\nu$ to each open edge of an* ok*-leaf. All left references that pointed to $\nu$ point in $\mathsf{Mon}(\Pi)$ to left($\nu$), and all right references that pointed to $\nu$ point in $\mathsf{Mon}(\Pi)$ to right($\nu$). The successor of right($\nu$) is the subtree rooted at the successor of $\nu$.*

2. *If $\nu$ is an* output node *of $\Pi$, replace $\nu$ with $f_{\mathsf{bad\text{-}knowledge},\Pi}(b, tr_\nu)$. Append the sequence of the two output nodes left($\nu$) (labeled with* `left`*) and right($\nu$) (labeled with* `right`*) to each open edge of an* ok*-leaf. All left references that pointed to $\nu$ point in $\mathsf{Mon}(\Pi)$ to left($\nu$), and all right references that pointed to $\nu$ point in $\mathsf{Mon}(\Pi)$ to right($\nu$). The successor of right($\nu$) is the subtree rooted at the successor of $\nu$.*

**Shortened Protocols $\Pi_i$** Computational soundness for uniformity of bi-protocols (see Theorem 3) follows from two properties of the distinguishing self-monitors: *symbolic self-monitoring* and *computational self-monitoring*. Symbolic self-monitoring states that whenever a bi-protocol $\Pi$ is indistinguishable, the corresponding distinguishing self-monitor in $\mathsf{Mon}(\Pi)$ does not raise the event bad. Computational self-monitoring,

[27] This leads to the fact that whenever there is an output node in $\Pi$, there are two corresponding output nodes in $\mathsf{Mon}(\Pi)$, which contradicts the goal that the interface of $\Pi$ and $\mathsf{Mon}(\Pi)$ should be the same towards the attacker. However, this technicality can be dealt with easily when applying our method. For example, in the computational proof for our case study, we use the self-monitor in an interaction with a filter machine that hides the results of the output nodes of $\overline{b}(\Pi)$ to create a good simulation towards the computational attacker, whose goal is to distinguish $\Pi$. The filter machine is then used as a computational attacker against $\mathsf{Mon}(\Pi)$.

[28] These functions are candidates for distinguishing self-monitors for bad-knowledge and bad-branch, respectively, for the bi-protocol $\Pi$, as defined in Definition 13.

in turn, states that whenever the distinguishing self-monitor in $\mathsf{Mon}(\Pi)$ does not raises the event $\mathsf{bad}$, then $\Pi$ is indistinguishable.

Since the statement (Theorem 3) is proven by induction over the nodes in a bi-protocol, we introduce a notion of *shortened protocols* in the definition of distinguishing self-monitors. For a (bi-)protocol $\Pi$, the shortened (bi-)protocol $\Pi_i$ is for the first $i$ nodes exactly like $\Pi$ but that stops after the $i$th node that is either a control node or an output node.[29]

[29] Formally, the protocol only has an infinite chain of control nodes with single successors after this node.