2015 IEEE 28th Computer Security Foundations Symposium

Symbolic Malleable Zero-knowledge Proofs

Michael Backes*†, Fabian Bendun*, Matteo Maffei*, Esfandiar Mohammadi*, Kim Pecina‡
* *CISPA, Saarland University, Saarbrücken, Germany*
Email: {backes,bendun,maffei,mohammadi}@cs.uni-saarland.de
† *MPI-SWS, Saarbrücken, Germany*
‡ *peloba UG & Co. KG, Saarbrücken, Germany*
Email: pecina@peloba.de

***Abstract*—Zero-knowledge (ZK) proofs have become a central building block for a variety of modern security protocols. 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.

***Keywords*-computational soundness; equivalence properties; privacy; zero-knowledge proofs**

I. 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 [1]–[3]. 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 [1].

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 [4]), requiring more sophisticated verification techniques, and on the other hand more complex primitives (e.g., secure multi-party computation [5]). In this work, we focus on zero-knowledge proofs [6], which have rapidly become a central building block for a variety of modern privacy protocols, such as verifiable computation [7], [8], e-voting systems [9], anonymous credentials [10], 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 [11] and partly the ZK-SNARKs used in Pinocchio [7] or AD-SNARK [8], 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 [12]–[14]). 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 [15]–[17]. There have even been two corresponding computational soundness results for these abstractions [18], [19]. 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

DOI 10.1109/CSF.2015.35

realizations [18], [19]. 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.

A. 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 [20], [21], 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 [22]).

Outline of the Paper. We first present and discuss our symbolic abstraction of malleable ZK proofs in § II. Then, we review the CoSP framework in § III. Finally, we show computational soundness in § IV. § VI concludes this work and outlines future work.

Proofs and technical details are reported in the full version [23].

II. Symbolic abstraction of malleable ZK proofs

A. The applied pi-calculus

We concisely review the syntax and the operational semantics of the applied pi-calculus [24]. 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.

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 $\mathit{left}(Q) \rightarrow R_{\mathit{left}}$ implies that $Q \rightarrow R$ for some bi-process R with $\mathit{left}(R) \equiv R_{\mathit{left}}$, and symmetrically for $\mathit{right}(Q) \rightarrow R_{\mathit{right}}$ with $\mathit{right}(R) \equiv R_{\mathit{right}}$.*

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 $C[Q] \rightarrow P$, the biprocess P is uniform, then $\mathit{left}(Q)$ is observationally equivalent to $\mathit{right}(Q)$.*

For a complete description of the applied pi-calculus, we refer to [24].

B. Anonymous Webs of Trust

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

[1] We further consider asymmetric encryptions and digital signatures, handled in a standard way [20].

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 vk_B by Bob,[2] represented by his verification key vk_B, everybody that trusts Bob can accept that 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 $\mathsf{vkofsk}(\mathrm{sk}(s)) = \mathrm{vk}(s)$. 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 [25] 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).

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.

[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.

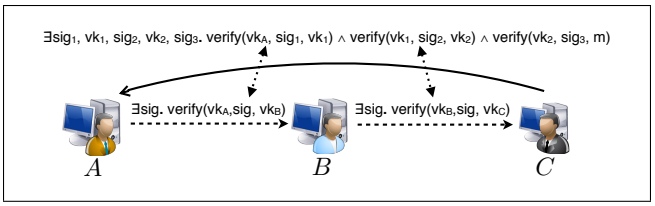

Figure 1. In decentralized anonymous webs of trust, zero-knowledge proofs are exchanged between Alice and Bob, and Bob and Charlie. Since these proofs are reused when proving a trust chain, denoted by the dotted arrows, the proofs have to be re-randomized. Otherwise, Bob, respectively Alice, will be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

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

Figure 2. The syntax of terms

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.

C. 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 π 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., [11]) 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 π as zero-knowledge proof or proof and to the triple $(\pi, \underline{c}, \underline{o})$ as zero-knowledge triple or triple. Furthermore, we let ε 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 vk_B (respectively, 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 [14].

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 [26].

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 π 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 cmb. Since a protocol only comprises a finite number of re-randomization operations, the number of nested 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 [14].

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

We use the constructor 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 π 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

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 $o_i := \varepsilon$

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

Table I
SET OF CONSTRUCTOR AND DESTRUCTOR EQUATIONS FOR A ZERO-KNOWLEDGE PROOF THAT CONTAINS n COMMITMENTS.

message m_i as $c_i = \mathsf{com}(m_i, r_i^H, r_i^A)$. Honest participants put ε as attacker randomness.

Modeling symbolic opening information. Analogously to computational opening information, the symbolic opening information consist of the three arguments to the com constructor. If the opening information has been removed because the corresponding committed value has been selectively hidden, it consists of the dedicated value ε.

Modeling symbolic verification. The symbolic zero-knowledge verification destructor verzk has to enforce two properties: first, it needs to assert that the randomness in π 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 π 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:

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

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 ε. The destructor equations are shown in Table I.

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 π itself is re-randomized using $i = 0$.

Modeling symbolic verification. The symbolic zero-knowledge verification destructor verzk consists of two parts: first, it needs to enforce that the proof π 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 π 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 π and in the commitments match. Additionally, the verification must ensure, that the signature verification succeeds. The destructor looks as follows:

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

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 ε. 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 I.

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 π itself is re-randomized using $i = 0$.

Symbolic length. We use the same notion of symbolic length as in [21]. Length is modelled in a Peano arithmetic manner: by the two constructors S and O(S). To every term the length destructor can be applied and returns the symbolic length.

Figure 2 depicts the full syntax of our symbolic model. Table I depicts the destructors connected to zero-knowledge proofs. The destructors for all other terms are as in the basic model of the full version [23] and similar to previous work [20], [21].

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. The transformation and its soundness proof can be found in [23]. 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.

D. 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 § II-A: 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 [27].[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 [28]. ∎

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

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

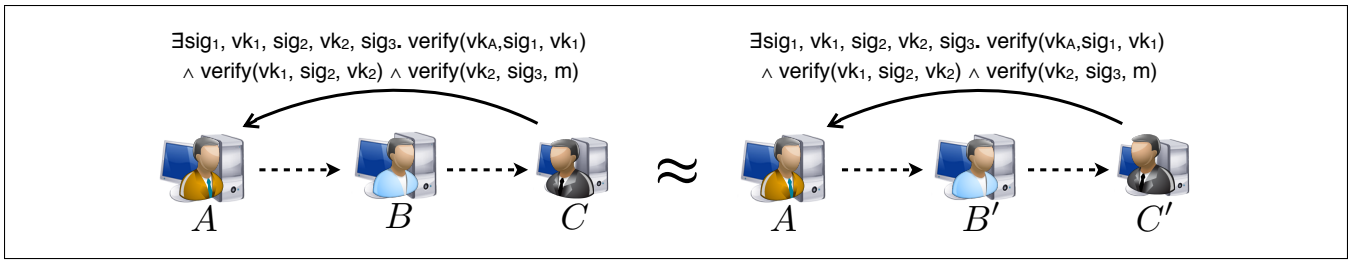

Figure 3. Anonymity game. A dashed arrow from principal I to principal J denotes that I signed the public-key pair of J. A solid arrow from principal I to J denotes that I send a zero-knowledge proof for the shown statement to J.

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 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 § II, 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.

III. Reviewing the CoSP framework for equivalence properties

In this section, we review the basic concepts underlying the CoSP framework for equivalence properties [21]. For technical details, we refer to the previous work [20], [21].

Symbolic Model. In CoSP, symbolic abstractions of protocols and, more importantly, of the attacker, are formulated in a symbolic model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D})$: a set of free functions $\mathbf{C}$, a countably infinite set $\mathbf{N}$ of nonces, a set $\mathbf{T}$ of terms, and a set $\mathbf{D}$ of partial mappings from terms to terms (called destructors). To unify notation, we introduce $\text{eval}_F(t)$: if F is a constructor, $\text{eval}_F(\underline{t}) := F(\underline{t})$ if $F(\underline{t}) \in \mathbf{T}$ and $\text{eval}_F(\underline{t}) := \bot$ otherwise. If F is a nonce, $\text{eval}_F() := F$. If F is a destructor, $\text{eval}_F(\underline{t}) := F(\underline{t})$ if $F(\underline{t}) \neq \bot$ and $\text{eval}_F(\underline{t}) := \bot$ otherwise.

Protocols. In CoSP, protocols are represented as infinite trees with the following nodes: *computation nodes* are used for drawing fresh nonces, and applying constructors and destructors; *input* and *output nodes* are used for sending and receiving operations; *control nodes* are used for allowing the attacker to schedule 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 constructor or destructor F on a term t fails, i.e., if $\text{eval}_F(t) = \bot$. Nodes have explicit *references* to other nodes whose terms they use.

Bi-protocols. For the definition of symbolic indistinguishability, we represent pairs of protocols as so-called bi-protocols, which are pairs of protocols that are encoded in one extended CoSP protocol tree. Bi-protocols are pairs of protocols that only differ in the messages they operate on.

Symbolic operations. As a next step, we model the capabilities of the symbolic attacker. We capture formalize the tests or operations that the attacker can perform on protocol messages with so-called *symbolic operations*. A symbolic operation is (similar to a CoSP tree) a finite tree, whose nodes are labeled with constructors, destructors, or nonces from the symbolic model M, or formal parameters, e.g., x_i, denoting pointers to the ith protocol-message. There is a natural evaluation function for a symbolic operation O and a list $\underline{t}$ of terms that the attacker received so far (the *view*).

Symbolic execution. A symbolic execution is a path through a protocol tree.

Definition 2 (Symbolic Execution)**.** *Let a symbolic model* $\mathsf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D})$ *and a CoSP protocol* Π *be given. A* symbolic view *of the protocol* Π *is a (finite) list of triples* (V_i, ν_i, f_i) *satisfying the following conditions. Initially, we have* $V_1 = \varepsilon$, ν_1 *is the root of* Π*, and* f_1 *is an empty partial function, mapping node identifiers to terms. For every two consecutive tuples* (V, ν, f) *and* (V', ν', f') *in the list, let* $\underline{\tilde{\nu}}$ *be the nodes referenced by* ν *and define* $\underline{\tilde{t}}$ *through* $\tilde{t}_j := f(\tilde{\nu}_j)$*. Figure 4 depicts a case distinction over* ν *for defining valid successors* ν', f', V'*.*

$\text{SViews}(\Pi)$ *is set of all symbolic views of* Π*.* V_{Out} *is the list of the terms* t *contained in* $(\texttt{out}, t_i) \in V$*.* $V_{\textit{Out-Meta}}$ *is the list the terms* l *contained in* $(\texttt{control}, (l, l')) \in V$*.* V_{In} *(the* attacker strategy*) is the list of terms that contains only entries from* V *of the form* $(\texttt{in}, (*, O))$ *or* $(\texttt{control}, (*, l'))$*, where the first termhas been masked with the symbol* $*$*.* $[V_{In}]_{\text{SViews}(\Pi)}$ *is the equivalence class of all views* $U \in \text{SViews}(\Pi)$ *with* $U_{In} = V_{In}$*.*

Symbolic knowledge. The *symbolic knowledge* of the at-

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

Figure 4. Symbolic execution

tacker comprises the results of all the symbolic tests the attacker can perform on the messages output by the protocol. Given a view V with $|V_{Out}| = n$, we define the *symbolic knowledge* K_V as a function from symbolic operations on $\mathbf{M}$ to $\{\top, \bot\}$, where $\top$ comprises all results of $\text{eval}_O(V_{Out})$ that are not $\bot$.

Equivalent views. As preparation for symbolic indistinguishability, we define two views two be *equivalent*, denoted as $V \sim V'$, if they (i) have the same structure (i.e., the same order of $\texttt{out}, \texttt{in}, \texttt{control}$ entries), (ii) have the same out-metadata (i.e., $V_{Out\text{-}Meta} = V'_{Out\text{-}Meta}$), and (iii) lead to the same knowledge (i.e., $(K_V = K_{V'})$).

Defining symbolic indistinguishability. 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.

1) Computational Indistinguishability: On the computational side, the constructors and destructors in a symbolic model are realized with cryptographic algorithms, which we call computational implementations.

Computational Implementation. A *computational implementation* is a family $A = (A_x)_{x \in \mathbf{C} \cup \mathbf{D} \cup \mathbf{N_P}}$ of deterministic polynomial-time algorithms A_F for each constructor or destructor $F \in \mathbf{C} \cup \mathbf{D}$ well as a probabilistic polynomial-time (ppt) algorithm A_N for drawing protocol nonces $N \in \mathbf{N}$.

Computational Execution. The *computational execution* of a protocol is a randomized interactive machine, called the *computational challenger*, that runs against a ppt attacker E. The transcript of the execution contains the computational counterparts of a symbolic view. The computational challenger is a probabilistic machine that traverses the protocol tree and interacts with the attacker: at a computation node the corresponding algorithm is run and depending on whether the algorithm succeeds or outputs $\bot$, 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 control node the attacker sends a command that specifies which branch to take.

Computational Indistinguishability. We use *termination-insensitive computational indistinguishability* [29] (tic-indistinguishability) to capture that two protocols are computationally indistinguishable. In comparison to standard notion of indistinguishability, tic-indistinguishability does not require the interactive machines to be polynomial-time; instead, it only considers decisions that were made for polynomially-bounded prefix of the interaction (where, both, the attacker's and the protocol's steps are counted). We write the fact that a machine M terminates after n steps with the output a as $M \Downarrow_n a$.

Definition 3 (Tic-indistinguishability [29])**.** *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. Two machines A and B are* tic-indistinguishable for a machine E *($A \approx^E_{tic} B$) if for all polynomials p, there is a negligible function μ 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. We call A and B tic-indistinguishable *($A \approx_{tic} B$) if $A \approx^E_{tic} B$ for all polynomial-time machines E.*

With the notion of tic-indistinguishability, we define a bi-protocol to be *computationally indistinguishable* if the corresponding challengers are tic-indistinguishable for every ppt attacker E.

Finally, we are ready to define *computational soundness*, which means that symbolic indistinguishability implies computational indistinguishability.

Definition 4 (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 M if for every $\Pi \in \mathrm{P}$, we have that Π is computationally indistinguishable whenever Π is symbolically indistinguishable.*

A. 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 [21], 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 Π, 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 Π and checks at run-time whether Π 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; hence, we refer to the work of Backes, Mohammadi, and Ruffing [21] for the actual construction of $\mathsf{Mon}(\Pi)$ and define here only the final properties that we require in order to reduce observational equivalence to trace equivalence.

Definition 5 (Distinguishing self-monitors)**.** *Let* M *be a symbolic model and* A *a computational implementation of* M*. Let* Π *be a bi-protocol and* $\mathsf{Mon}(\Pi)$ *its self-monitor. Let* $e \in \{\mathsf{bad\text{-}knowledge}, \mathsf{bad\text{-}branch}\}$ *and* $n_{\mathsf{bad\text{-}knowledge}}$ *denote the node type output node and* $n_{\mathsf{bad\text{-}branch}}$ *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* Π *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* Π_i *is symbolically indistinguishable,* bad *does symbolically not occur in* $\mathsf{Mon}(\Pi_{i-1})$*, and the* i*th node in* Π_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 negligible probability,* Π_{i-1} *is computationally indistinguishable, and the* i*th node in* Π_i *is of type* n_e*, then* Π_i *is computationally indistinguishable.*

We say that a M *and a protocol class* allow for self-monitoring *if for every bi-protocol* Π *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*[5] *class of bi-protocols. If* M *and* P *allow for self-monitoring (in the sense of Definition 5), 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.*

IV. 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 protocols for which we can show computational soundness (Section IV-A). We then introduce necessary implementation conditions for computationally sound realizations (Section IV-B). Finally, we conduct the actual proof (Section IV-C). 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.

A. 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. 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 a 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

[5]The property of uniformity-enforcing is basically a syntactic property and required to construct $\mathsf{Mon}(\Pi)$. It requires that the attacker is informed whenever the left and the right protocol of the bi-protocol Π take different branches (see [21] for details).

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 proof 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 [26], [30] applies to our setting as well. If we include protocols that use attacker randomness or re-use previous protocol randomness for re-randomization, it turns out that we run into similar problems as with XOR: the attacker would be 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, which eliminates any influence of the attacker and thereby remedies the above-mentioned problems.

B. 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.[6] More specifically, we require that $A_{\mathrm{vkof}}(m) \neq \bot$, $A_{\mathrm{ekof}}(m) \neq \bot$, and $A_{\mathrm{crsof}}(m) \neq \bot$ 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_{ek}, A_{dk}, A_{enc}, A_{dec} constitute an IND-CCA secure encryption scheme. Moreover, we require that whenever $A_{\mathrm{dec}}(\mathrm{dk}_N, c)$ succeeds, $A_{\mathrm{ekof}}(c) = \mathrm{ek}_N$ outputs the corresponding public key, and that $A_{\mathrm{ek}}(r) = A_{\mathrm{ekofdk}}(A_{\mathrm{dk}}(r))$. For the signature algorithms A_{sk}, A_{vk}, A_{sig}, $A_{\mathrm{ver_{sig}}}$ we require that they constitute a CMA-existentially unforgeable signature scheme. We require that A_{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 $A_{\mathrm{ZK}}, A_{\mathrm{ver_{zk}}}, A_{\mathrm{crs}}, A_{\mathrm{com}}$ constitute a non-interactive argument of knowledge (NIZKAoK), and the same commitment algorithms $A_{\mathrm{crs}}, A_{\mathrm{com}}, A_{\mathrm{open}}$ belong to an extractable non-interactive computationally hiding and binding commitment scheme. We also assume an algorithm A_{getPub} such that $A_{\mathrm{getPub}}(A_{\mathrm{ZK}}(t, r)) = r$ and we assume that the protocol transformations setPub, and$_{\mathrm{ZK}}$, splitAnd, or$_{\mathrm{ZK}}$, commute, rer$_{\mathrm{zk}}$, rer$_{\mathrm{com}}$ 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 $R_{\mathrm{hon}}^{\mathrm{comp}}$ for the protocol is then defined as follows:

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

Controlled malleability. As shown by Fuchsbauer [32, 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:

[6] This requirement is similar in spirit as the recent, more general, result by Mödersheim and Katsoris [31] that characterizes which message formats can be soundly abstracted in Dolev-Yao model.

controlled-malleable simulation sound extractability [33], 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 [11], after the generic construction from [33] is applied to it, and the malleable SNARKs of Chase, Kohlweiss, Lysyanskaya, and Meiklejohn [34].

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.

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 [33].

Efficient statements. In contrast to previous work on computationally symbolic ZK proofs [18], [19], the FMZK and the CMZK model have computationally sound realizations that allow efficient statements for encryption schemes and signature [11], [35], [36]. 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 [33]. The constructions of malleable zero-knowledge proofs in [33], [34] that satisfy our implementation conditions are also shown to satisfy this strong derivation privacy property.

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

The pure Groth-Sahai scheme, without any provision for controlled malleability, enables even more transformations, as pointed out by Fuchsbauer in his PhD Thesis. However, it is not clear whether these transformations (such as squaring the witness) are useful in practice.

C. 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 β and the parsing function τ. The construction function β 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 β over terms, e.g., $\beta(\text{enc}(t_1, t_2, t_3)) := A_{\text{enc}}(\beta(t_1), \beta(t_2), \beta(t_3))$.

Reconstructing protocol transformations. In the presence of zero-knowledge transformations, it can happen that the simulator parses (using τ) 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 β) 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: $\text{eval}(\hat{f}(\underline{s})) := f(\underline{\text{eval}(s)})$ and $\text{eval}(t) := t$ 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 [26], [30]. 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 IV-A) 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 τ and β. We only discuss the parts of the construction of τ and β that are different from previous work since the other cases are conducted as in previous work [18], [20]: Upon receiving a zero-knowledge proof bitstring z that is different from the ones that have been sent by the protocol, the parsing function τ first symbolically checks whether the proof only contains information that the attacker could have derived on his own. If the check succeeds, τ 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, τ 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, τ first applies all potentially necessary selective hiding transformations and then potentially a re-randomization transformation, where τ stores in the index of the symbolic attacker-randomness N^z the bitstring z. Let f denote this sequence of transformations. Then, τ 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 τ 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 β we are given a proof term t. For all honestly generated proofs t, we recursively evaluate β 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 Sim_f that fakes all honest encryptions, i.e., 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 β: 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 IV-B).

The indistinguishability of Sim. We then show that Sim and 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 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 τ and our construction function β potentially interpret transformed proofs as freshly generated proofs. Using derivation privacy [33], we show that a simulator Sim that uses β and τ 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.*

D. Computational soundness w.r.t. uniformity: self-monitoring

In this section, we discuss the distinguishing self-monitors for the symbolic model M. Let $b \in \{\mathrm{left}, \mathrm{right}\}$ throughout this section and let, for a CoSP bi-protocol Π, $b(\Pi)$ be for the sake of exposition the protocol that is currently executed and $\bar{b}(\Pi)$ 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*.

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 ν 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 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 τ of the computational soundness simulator (see Section IV-C).

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 ν in question. If this is not the case, the event bad-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 [21] applies verbatim.

Lemma 1 (Symbolic self-monitoring of the branching monitor)**.** *Let Π be a bi-protocol from the protocol class P, and M be the symbolic model from Section II. The branching monitor satisfies symbolic self-monitoring (see Definition 5).*

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 [21].

Lemma 2 (Computational self-monitoring of the branching monitor)**.** *The branching monitor satisfies computational self-monitoring (see Definition 5).*

2) The knowledge monitor: The distinguishing self-monitor $f_{\mathsf{bad\text{-}knowledge},\Pi}(b, tr)$, the knowledge monitor, for an output node ν starts like $f_{\mathsf{bad\text{-}branch},\Pi}(b, tr)$ by reconstructing a (symbolic) attacker strategy and simulating a symbolic execution of $\overline{b}(\Pi)$. However, instead of testing the branching behavior of $\overline{b}(\Pi)$, 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 ν in question, and then $f_{\mathsf{bad\text{-}knowledge},\Pi}(b, tr)$ compares m to the message that would be output in $\overline{b}(\Pi)$. 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 $\overline{b}(\Pi)$ 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 Π be a bi-protocol. Let $i \in \mathbb{N}$. Let Π'_i be the self-monitor for Π_i. For all $i \in \mathbb{N}$ the following holds. If there is an attacker strategy such that in Π'_i the event* bad-knowledge *occurs but in Π'_{i-1} the event* bad *does not occur and Π_{i-1} is symbolically indistinguishable, then Π_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 [21]. First, we show that the simulator-indistinguishability w.r.t. trace properties (see Section IV-C) 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 Sim_f (see Section IV-C) 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 Π is distinguishable.

Simulator-indistinguishability. The first step, follows from Lemma 12 in [37]. 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[7] 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 5).*

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 IV-B. Then, A is computationally sound w.r.t. uniformity of CoSP bi-protocols for the class of MZK-safe CoSP bi-protocols.*

V. Soundness of the verification

In section II-D, 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 Π calculus. Moreover, we have shown in section IV that the symbolic model that also contains re-randomization is computationally sound. Combined we can conclude the following result for an implementation:

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

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 II-D, 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. ■

VI. 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 the class of uniformity properties (the class of equivalence properties that ProVerif is able to verify). The abstraction and the computational soundness result are presented in CoSP, a framework for symbolic protocol analyses and conceptually modular computational soundness proofs.

References

[1] D. Dolev and A. C. Yao, "On the security of public key protocols," *IT'83*, vol. 29, no. 2, pp. 198–208, 1983.

[2] S. Even and O. Goldreich, "On the security of multi-party ping-pong protocols," in *FOCS'83*, 1983, pp. 34–39.

[3] M. Merritt, "Cryptographic protocols," Ph.D. dissertation, Georgia Institute of Technology, 1983.

[4] B. Schmidt, R. Sasse, C. Cremers, and D. Basin, "Automated Verification of Group Key Agreement Protocols," in *Proc. of S&P'14*. IEEE Computer Society Press, 2014, pp. 179–194.

[5] M. Backes, M. Maffei, and E. Mohammadi, "Computationally Sound Abstraction and Verification of Secure Multi-Party Computations," in *Proc. of FSTTCS'10*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010, pp. 352–363.

[6] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM Journal on Computing*, vol. 18, no. 1, pp. 186–207, 1989.

[7] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly Practical Verifiable Computation," in *Proc. of S&P'13*. IEEE Computer Society Press, 2013, pp. 238–252.

[8] M. Backes, M. Barbosa, D. Fiore, and R. M. Reischuk, "ADSNARK: Nearly Practical and Privacy-Preserving Proofs on Authenticated Data," in *Proc. of S&P'15*. IEEE Computer Society Press, 2015.

[9] M. R. Clarkson, S. Chong, and A. C. Myers, "Civitas: Toward a Secure Voting System," in *Proc. of S&P'08*. IEEE Computer Society Press, 2008, pp. 354–368.

[10] F. Baldimtsi and A. Lysyanskaya, "Anonymous Credentials Light," in *Proc. of CCS'13*. ACM Press, 2013, pp. 1087–1098.

[11] J. Groth and A. Sahai, "Efficient non-interactive proof systems for bilinear groups," in *EUROCRYPT'08*. Springer-Verlag, 2008, pp. 415–432.

[12] M. Maffei and K. Pecina, "Position paper: Privacy-aware proof-carrying authorization," in *PLAS 2011*, 2011.

[13] M. Backes, M. Maffei, and K. Pecina, "Automated Synthesis of Privacy-Preserving Distributed Applications," in *NDSS'12*. Internet Society, 2012.

[14] M. Maffei, K. Pecina, and M. Reinert, "Security and privacy by declarative design," in *Proc. of CSF'13*. IEEE Computer Society, 2013.

[15] M. Backes, M. Maffei, and D. Unruh, "Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol," in *S&P'08*, 2008, pp. 158–169.

[16] M. Backes, C. Hriţcu, and M. Maffei, "Type-checking zero-knowledge," in *Proc. 15th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM Press, 2008, pp. 357–370.

[17] ——, "Union and intersection types for secure protocol implementations," in *Proc. 2011 International Conference on Theory of Security and Applications*, ser. TOSCA'11. Springer-Verlag, 2012, pp. 1–28.

[18] M. Backes, F. Bendun, and D. Unruh, "Computational soundness of symbolic zero-knowledge proofs: Weaker assumptions and mechanized verification," in *POST'13*, 2013, pp. 206–225.

[19] M. Backes and D. Unruh, "Computational soundness of symbolic zero-knowledge proofs against active attackers," in *Proc. 21th IEEE Computer Security Foundations Symposium (CSF)*, 2008.

[20] M. Backes, D. Hofheinz, and D. Unruh, "Cosp: A general framework for computational soundness proofs," in *Proc. 16th ACM SIGSAC Conference on Computer and Communications Security (CCS)*, November 2009, pp. 66–78.

[21] M. Backes, E. Mohammadi, and T. Ruffing, "Computational Soundness Results for ProVerif," in *Proc. of POST'14*. Springer, 2014, pp. 42–62.

[22] M. Backes, M. Maffei, and D. Unruh, "Computationally sound verification of source code," in *Proc. 17th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM Press, October 2010, pp. 387–398.

[23] "Full version: Symbolic Malleable Zero-knowledge Proofs," http://www.infsec.cs.uni-saarland.de/~mohammadi/paper/malleable_zk.pdf.

[24] B. Blanchet, M. Abadi, and C. Fournet, "Automated Verification of Selected Equivalences for Security Protocols," *Journal of Logic and Algebraic Programming*, vol. 75, pp. 3–51, 2008.

[25] M. Backes, S. Lorenz, M. Maffei, and K. Pecina, "Anonymous webs of trust," in *Proceedings of the 10th international conference on Privacy enhancing technologies*, ser. PETS'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 130–148. [Online]. Available: http://dl.acm.org/citation.cfm?id=1881151.1881159

[26] D. Unruh, "The Impossibility of Computationally Sound XOR," http://eprint.iacr.org/2010/389, 2010.

[27] S. Meier, B. Schmidt, C. Cremers, C. Staub, R. Sasse, and D. Basin, Accessed on 19 April 2015 at http://www.infsec.ethz.ch/research/software/tamarin.html.

[28] "Case Study: Verifying AWoT in ProVerif," http://e.mohammadi.eu/paper/awot.zip.

[29] D. Unruh, "Termination-Insensitive Computational Indistinguishability (and Applications to Computational Soundness)," in *Proc. 24th IEEE Computer Security Foundations Symposium (CSF)*. IEEE Computer Society Press, 2011, pp. 251–265.

[30] M. Backes and B. Pfitzmann, "Limits of the Cryptographic Realization of Dolev-Yao-Style XOR," in *Proc. of ESORICS'05*, ser. LNCS, vol. 3679. Springer Berlin Heidelberg, 2005, pp. 178–196.

[31] S. Mödersheim and G. Katsoris, "A Sound Abstraction of the Parsing Problem," in *Proc. 27th IEEE Computer Security Foundations Symposium (CSF)*. IEEE Computer Society Press, 2014, pp. 259–273.

[32] G. Fuchsbauer, "Automorphic signatures and applications," Ph.D. dissertation, École normale supérieure, Paris, 2010.

[33] M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn, "Malleable proof systems and applications," in *EUROCRYPT'12*, ser. LNCS, vol. 7237. Springer, 2012, pp. 281–300.

[34] Chase, Melissa and Kohlweiss, Markulf and Lysyanskaya, Anna and Meiklejohn, Sarah, "Succinct malleable nizks and an application to compact shuffles," in *Proc. of TCC'13*, ser. LNCS, vol. 7785. Springer-Verlag, 2013, pp. 100–119.

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

[36] J. Camenisch, K. Haralambiev, M. Kohlweiss, J. Lapon, and V. Naessens, "Structure preserving cca secure encryption and applications," in *Proc. of ASIACRYPT'11*. Springer-Verlag, 2011, pp. 89–106.

[37] M. Backes, E. Mohammadi, and T. Ruffing, "Full Version: Computational Soundness Results for ProVerif," http://e.mohammadi.eu/paper/bridge.pdf, 2014.