Zero-Knowledge in the Applied Pi-calculus and Automated Verification of the Direct Anonymous Attestation Protocol

Michael Backes, Matteo Maffei, and Dominique Unruh
Saarland University, Saarbrücken, Germany
{backes,maffei,unruh}@cs.uni-sb.de

Abstract

We devise an abstraction of zero-knowledge protocols that is accessible to a fully mechanized analysis. The abstraction is formalized within the applied pi-calculus using a novel equational theory that abstractly characterizes the cryptographic semantics of zero-knowledge proofs. We present an encoding from the equational theory into a convergent rewriting system that is suitable for the automated protocol verifier ProVerif. The encoding is sound and fully automated. We successfully used ProVerif to obtain the first mechanized analysis of the Direct Anonymous Attestation (DAA) protocol. The analysis in particular required us to devise novel abstractions of sophisticated cryptographic security definitions based on interactive games.

1 Introduction

Proofs of security protocols are known to be error-prone and, owing to the distributed-system aspects of multiple interleaved protocol runs, awkward to make for humans. In fact, vulnerabilities have accompanied the design of such protocols ever since early authentication protocols like Needham-Schroeder [11, 24], over carefully designed de-facto standards like SSL and PKCS [28, 8], up to current widely deployed products like Microsoft Passport [13] and Kerberos [10]. Hence work towards the automation of such proofs has started soon after the first protocols were developed; some important examples of automated security proofs are [23, 22, 18, 21, 25, 27, 3, 5]. Language-based techniques are now widely considered a particularly salient approach for formally analyzing security protocols, dating back to Abadi's seminal work on secrecy by typing [1]. The ability to reason about security at the language level often allows for concisely clarifying why certain message components are included in a protocol, how their entirety suffices for establishing desired security guarantees, and for identifying ambiguities in protocol messages that could be exploited by an adversary to mount a successful attack on the protocol.

One of the central challenges in the analysis of complex and industrial-size protocols is the expressiveness of the formalism used in the formal analysis and its capability to model complex cryptographic operations. While such protocols traditionally relied only

on the basic cryptographic operations such as encryption and digital signatures, Modern Cryptography has invented more sophisticated primitives with unique security features that go far beyond the traditional understanding of cryptography to solely offer secrecy and authenticity of a communication. Zero-knowledge proofs constitute the most prominent and arguably most amazing such primitive. 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. On the other hand, a zero-knowledge proof does not reveal any information besides the bare fact that x constitutes a valid statement. In particular, a proof about some ciphertext would not leak the decryption key or the plaintext. Zero-knowledge proofs were introduced in [17] and were proven to exist for virtually all statements [16]. Zero-knowledge proofs have since shown to constitute very powerful building blocks for the construction of sophisticated cryptographic protocols to solve demanding protocol task: they allow for commonly evaluating a function on distributed inputs without revealing any inputs to the other protocol participants [15], they allow for developing encryption schemes that are secure under very strong active attacks [12], and many more.

Early general-purpose zero-knowledge proofs were mainly invented to show the mere existence of such proofs for the class of statements under consideration. These proofs were very inefficient and consequently of only limited use in practical applications. The recent advent of efficient zero-knowledge proofs for special classes of statements changed this. The unique security features that zero-knowledge proofs offer combined with the possibility to efficiently implement some of these proofs have paved these proofs the way into modern cryptographic protocols such as e-voting protocols and anonymity protocols. The best known representative of these protocols is the widely-deployed Direct Anonymous Attestation (DAA) protocol [9]. DAA constitutes a cryptographic protocol that enables the remote authentication of a Trusted Platform Module (TPM) while preserving the user's privacy. More precisely, if the user talks to the same verifier twice, the verifier is not able to tell if he communicates with the same user as before or with a different one. DAA achieves its anonymity properties by heavily relying on non-interactive zero-knowledge proofs. Intuitively, these allow the TPM to authenticate with the verifier without revealing the TPM's secret identifier.

1.1 Our Contributions

The contribution of the paper is threefold: First, we present an abstraction of non-interactive zero-knowledge proofs within the applied pi-calculus using a novel equational theory that abstractly characterizes the cryptographic semantics of these proofs. Second, we transform our abstraction into an equivalent formalization that is accessible to ProVerif [6], a well-established tool for the mechanized analysis of different security properties. Third, we apply our theory to the Direct Anonymous Attestation (DAA) protocol [9], the widely deployed authentication scheme for Trusted Platform Modules (TPMs), yielding its first mechanized security proof.

We express cryptographic protocols in the applied pi-calculus, an extension of the pi-calculus with functions, that has proven to constitute a salient foundation for the analysis of cryptographic protocols, see [2, 20, 6, 7, 14]. We devise a novel equational theory that concisely and elegantly characterizes the semantic properties of non-interactive zero-knowledge proofs, and that allows for abstractly reasoning about such proofs. The design of the theory in particular requires to carefully address the important principles that zero-knowledge proofs are based upon: the soundness and the completeness of the proof verification as well as the actual zero-knowledge property, i.e., a verifier must not be able to learn any new information from a zero-knowledge proof except for the validity of the proven statement. The only prior work on abstracting zero-knowledge proofs aims at formalizing in modal logic the informal prose used to describe the properties of these proofs [19]. In contrast to our abstraction, the abstraction in [19] has not been applied to any example protocols, and no mechanization of security proofs is considered there.

The mechanization of language-based security proofs has recently enjoyed substantial improvements that have further strengthened the position of language-based techniques as a promising approach for the analysis of complex and industrial-size cryptographic protocols. ProVerif [6] constitutes a well-established automated protocol verifier based on Horn clauses resolution that allows for the verification of observational equivalence and of different trace-based security properties such as authenticity. We present a mechanized encoding of our equational theory into a finite specification that is suitable for ProVerif. More precisely, the equational theory is compiled into a convergent rewriting system that ProVerif can efficiently cope with. We prove that the encoding preserves observational equivalence and a large class of trace-based security properties.

Finally, we exemplify the applicability of our theory to real-world protocols by analyzing the security properties of the Direct Anonymous Attestation (DAA) protocol [9]. DAA constitutes a cryptographic protocol that enables the remote authentication of a hardware module called the Trusted Platform Module (TPM), while preserving the anonymity of the user owning the module. Such TPMs are now widely included in end-user hardware such as desktop PCs and notebooks. The DAA protocol relies heavily on zero-knowledge proofs to achieve its anonymity guarantees. The occurrence of these proofs in particular prevented a previous analysis of the protocol using abstraction or any form of proof mechanization. Analyzing DAA first requires to devise novel abstractions of sophisticated cryptographic security definitions based on interactive games between honest participants and the adversary; comprehensive anonymity properties are of this form. We formulate the intended anonymity properties in terms of observational equivalence, we formulate authenticity as a trace-based property, and we prove these properties in the presence of external active adversaries as well as corrupted participants. The proofs are fully automated using ProVerif.

We are confident that the methodology presented in this paper is general and that the principles followed in the analysis of DAA can be successfully exploited for the verification of other cryptographic protocols based on non-interactive zero-knowledge proofs.

Table 1 Syntax of the applied pi-calculus

Terms

M, N, F, Z	::=	$s, k, \ldots, a, b, \ldots, n, m$	*names*
		x, y, z	*vars*
		$\mathsf{f}(M_1, \ldots, M_k)$	*function*

where $\mathsf{f} \in \Sigma$ and k is the arity of f.

Processes

P, Q	::=	$\mathbf{0}$	*nil*
		$\nu n.P$	*res*
		$if M = N \ then \ P \ else \ Q$	*cond*
		$a(x).P$	*input*
		$\overline{a}\langle N \rangle.P$	*output*
		$P \mid Q$	*par*
		$!P$	*repl*

Extended Processes

A	::=	P	*plain*
		$A_1 \mid A_2$	*par*
		$\nu n.A$	*name res*
		$\nu x.A$	*var res*
		$\{M/x\}$	*subst*

1.2 Outline of the Paper

We start by reviewing the applied pi-calculus in Section 2. Section 3 contains the equational theory for abstractly reasoning about non-interactive zero-knowledge proofs in the applied pi-calculus. This equational theory is rewritten into an equivalent finite theory in terms of a convergent rewriting system in Section 4. Section 5 and 6 elaborate on the analysis of DAA, the description of its security properties, and the use of ProVerif for mechanizing the analysis. Section 7 concludes and outlines future work.

2 Review of the Applied Pi-calculus

The syntax of the calculus is given in Table 1. Terms are defined by means of a *signature* Σ, which consists of a set of function symbols, each with an arity. The *set of terms* T_Σ is the free algebra built from names, variables, and function symbols in Σ applied to arguments. We partition each signature into *public* and *private* function symbols. The only difference is that private symbols are not available to the adversary. In the following, functions symbols are public unless stated otherwise. We presuppose a sort system for the set $\mathcal{N}$ of names: we let s, k (possibly with sub- and superscripts) range over names of base type (e.g., $\mathsf{Integer}$, Data, and so on), a, b over channel names, and n, m over names of any sort. We let u range over names and variables. Terms are equipped

with an *equational theory* E, i.e., an equivalence relation on terms that is closed under substitution of terms and under application of term contexts (terms with a hole). We write $E \vdash M = N$ and $E \nvdash M = N$ for an equality and an inequality, respectively, modulo E.

The grammar of processes (or *plain processes*) is defined as follows. The null process $\mathbf{0}$ does nothing; $\nu n.P$ generates a fresh name n and then behaves as P; *if* $M = N$ *then* P *else* Q behaves as P if $E \vdash M = N$, and as Q otherwise; $a(x).P$ receives a message N from the channel a and then behaves as $P\{N/x\}$; $\overline{a}\langle N\rangle.P$ outputs the message N on the channel a and then behaves as P; $P \mid Q$ executes P and Q in parallel; $!P$ generates an unbounded number of copies of P.

Extended processes are plain processes extended with *active substitutions*. An active substitution $\{M/x\}$ is a floating substitution that may apply to any process that it comes into contact with. To control the scope of active substitutions, we can restrict the variable x. Intuitively, $\nu x.(P \mid \{M/x\})$ constrains the scope of the substitution $\{M/x\}$ to process P. If the variable x is not restricted, as it is the case in the process $(P \mid \{M/x\})$, then the substitution is exported by the process and the environment has immediate access to M. As usual, the scope of names and variables is delimited by restrictions and by inputs. We write $fv(A)$ and $fn(A)$ (resp. $bv(A)$ and $bn(A)$) to denote the free (bound) variables and names in an extended process A, respectively. We let $\mathsf{free}(A) := fv(A) \cup fn(A)$ and $\mathsf{bound}(A) := bv(A) \cup bn(A)$. For sequences $\widetilde{M} = M_1, \ldots, M_k$ and $\widetilde{x} = x_1, \ldots, x_k$, we let $\{\widetilde{M}/\widetilde{x}\}$ denote $\{M_1/x_1\} \mid \ldots \mid \{M_k/x_k\}$. We always assume that substitutions are cycle-free, that extended processes contain at most one substitution for each variable, and that extended processes contain exactly one substitution for each restricted variable.

A *context* is a process or an extended process with a hole. An *evaluation context* is a context without private function symbols whose hole is not under a replication, a conditional, an input, or an output. A context $C[\_]$ closes A if $C[A]$ is closed. A *frame* is an extended process built up from $\mathbf{0}$ and active substitutions by parallel composition and restriction. We let ϕ and ψ range over frames. The domain $dom(\phi)$ of a frame ϕ is the set of variables that ϕ exports, i.e., those variables x for which ϕ contains a substitution $\{M/x\}$ not under a restriction on x. Every extended process A can be mapped to a frame $\phi(A)$ by replacing every plain process embedded in A with $\mathbf{0}$. The frame $\phi(A)$ can be viewed as an approximation of A that accounts for the static knowledge A exposes to its environment, but not for A's dynamic behavior.

The semantics is inherited from the applied pi-calculus and is defined in terms of *structural equivalence* ($\equiv$) and *internal reduction* ($\rightarrow$). Structural equivalence states which processes should be considered equivalent up to syntactic re-arrangement.

Definition 1 (Structural Equivalence) *Structural equivalence ($\equiv$) is the smallest equivalence relation on extended processes that satisfies the rules in Table 2 and that is closed under α-renaming, i.e., renaming of bound names and variables, and under application of evaluation contexts.*

Internal reduction defines the semantics for extended processes.

Definition 2 (Internal Reduction) *Internal reduction ($\rightarrow$) is the smallest relation on*

Table 2 Structural Equivalence

PAR-0	$A \equiv A \mid 0$	
PAR-A	$A_1 \mid (A_2 \mid A_3) \equiv (A_1 \mid A_2) \mid A_3$	
PAR-C	$A_1 \mid A_2 \equiv A_2 \mid A_1$	
REPL	$!P \equiv P \mid !P$	
RES-0	$\nu n.0 \equiv 0$	
RES-C	$\nu u.\nu u'.A \equiv \nu u'.\nu u.A$	
RES-PAR	$A_1 \mid \nu u.A_2 \equiv \nu u.(A_1 \mid A_2)$	if $u \notin \mathsf{free}(A_1)$
ALIAS	$\nu x.\{M/x\} \equiv 0$	
SUBST	$\{M/x\} \mid A \equiv \{M/x\} \mid A\{M/x\}$	
REWRITE	$\{M/x\} \equiv \{N/x\}$	if $\Sigma \vdash M = N$

Table 3 Internal reduction

COMM
$$\overline{a}\langle x\rangle.P \mid a(x).Q \rightarrow P \mid Q$$

THEN
$$\frac{M \text{ ground}}{\mathit{if}\, M = M\ \mathit{then}\ P\ \mathit{else}\ Q\ \rightarrow\ P}$$

ELSE
$$\frac{E \not\vdash M = N \qquad M, N \text{ ground}}{\mathit{if}\, M = N\ \mathit{then}\ P\ \mathit{else}\ Q\ \rightarrow\ Q}$$

extended processes that satisfies the rules in Table 3 and that is closed under structural equivalence and under application of evaluation contexts.

We write $A \Downarrow a$ to denote that A can send a message on a, i.e., $A \rightarrow^* C[\overline{a}\langle M\rangle.P]$ for some evaluation context $C[\_]$ that does not bind a. *Observational equivalence* constitutes an equivalence relation that captures the equivalence of processes with respect to their dynamic behavior.

Definition 3 (Observational Equivalence) *Observational equivalence ($\approx$) is the largest symmetric relation $\mathcal{R}$ between closed extended processes with the same domain such that $A\mathcal{R}B$ implies:*

1. *if $A \Downarrow a$, then $B \Downarrow a$;*
2. *If $A \rightarrow^* A'$, then $B \rightarrow^* B'$ and $A'\mathcal{R}B'$ for some B';*
3. *$C[A]\mathcal{R}C[B]$ for all closing evalution contexts $C[\_]$.*

3 An Equational Theory of Zero-Knowledge

In this section we define a signature and an equational theory for abstractly reasoning about non-interactive zero-knowledge proofs. Our equational theory is parametric in that it augments an arbitrary base equational theory.

3.1 An Underlying Cryptographic Base Theory

The base equational theory we consider in this paper is given in Table 4. (Note again though that any other base theory would work as well.) First, it consists of functions for constructing and destructing pairs, encrypting and decrypting messages by symmetric and asymmetric cryptography, signing messages and verifying signatures, modelling public and private keys, hashing, and constructing and verifying blind signatures. In blind signature schemes, the content of a message is disguised before it is signed while still ensuring public verifiability of the signature against the unmodified message. These functions have received prior investigation within the applied Pi-calculus, e.g., to analyze the JFK protocol [2] and the electronic voting protocol FOO 92 [20]. Second, the theory contains three binary functions eq, $\wedge$, and $\vee$ for modelling equality test, conjunction, and disjunction, respectively; these functions allow for modelling monotone Boolean formulas. In our example theory, we do not consider additional functions for, e.g., negation or specifying explicit inequalities. We shall often write $=$ instead of eq and use infix notation for the functions eq, $\wedge$, and $\vee$.

3.2 The Equational Theory for Zero-Knowledge

Our equational theory for abstractly reasoning about non-interactive zero-knowledge proofs is given in Table 5; its components are explained in the following. A non-interactive zero-knowledge proof is represented as a term of the form $\mathsf{ZK}_{i,j}(\widetilde{M}, \widetilde{N}, F)$, where $\widetilde{M}$ and $\widetilde{N}$ denote sequences $M_1, \ldots, M_i$ and $N_1, \ldots, N_j$ of terms, respectively, and where F constitutes a formula over those terms, see below. Hence $\mathsf{ZK}_{i,j}$ is a function of arity $i+j+1$. We shall often omit arities and write this statement as $\mathsf{ZK}(\widetilde{M}; \widetilde{N}; F)$, letting semicolons separate the respective components. The statement will keep secret the terms $\widetilde{M}$, called the statement's *private component*, while the terms $\widetilde{N}$, called the statement's *public component*, will be revealed to the verifier and to the adversary. The formula F constitutes a term without names and variables but additionally built upon distinguished nullary functions α_i and β_i with $i \in \mathbb{N}$.

Definition 4 ((i,j)-formulas) *We call a term an* (i,j)-formula *if the term contains neither names nor variables, and if for every α_k and β_l occurring therein, we have $k \in [1,i]$ and $l \in [1,j]$.*

The values α_i and β_j in F constitute placeholders for the terms M_i and N_j, respectively. For instance, the term

$$\mathsf{ZK}(\ k\ ;\ m, \mathsf{enc}_{\mathsf{sym}}(m,k)\ ;\ \beta_2 = \mathsf{enc}_{\mathsf{sym}}(\beta_1, \alpha_1))$$

denotes a zero-knowledge proof that the term $\mathsf{enc}_{\mathsf{sym}}(m,k)$ is an encryption of m with k. More precisely, the statement reads: "There exists a key such that the ciphertext $\mathsf{enc}_{\mathsf{sym}}(m,k)$ is an encryption of m with this key". As mentioned before, $\mathsf{enc}_{\mathsf{sym}}(m,k)$ and m are revealed by the proof while k is kept secret. This is formalized in general terms by the following infinite set of equational rules:

Table 4 A base equational theory containing basic cryptographic primitives and logical operators

$$\Sigma_{\mathsf{base}} = \left\{ \begin{array}{l} \mathsf{pair}, \mathsf{enc_{sym}}, \mathsf{dec_{sym}}, \mathsf{enc_{asym}}, \mathsf{dec_{asym}}, \\ \mathsf{sign}, \mathsf{ver}, \mathsf{msg}, \mathsf{pk}, \mathsf{sk}, \mathsf{hash}, \mathsf{blind}, \\ \mathsf{unblind}, \mathsf{blindsign}, \mathsf{blindver}, \mathsf{blindmsg}, \\ \wedge, \vee, \mathsf{eq}, \mathsf{first}, \mathsf{snd}, \mathsf{true}, \mathsf{false} \end{array} \right\}$$

ver and blindver of arity 3, pair, $\mathsf{enc_{sym}}$, $\mathsf{dec_{sym}}$, $\mathsf{enc_{asym}}$, $\mathsf{dec_{asym}}$, sign, blind, unblind, blindsign, $\wedge$, $\vee$ and eq of arity 2, msg, pk, sk, hash, blindmsg, first and snd of arity 1, true and false of arity 0.

E_{base} is the smallest equational theory satisfying the following equations defined over all x, y, z:

$$\begin{array}{lcl} \mathsf{first}(\mathsf{pair}(x,y)) & = & x \\ \mathsf{snd}(\mathsf{pair}(x,y)) & = & y \\ \mathsf{dec_{sym}}(\mathsf{enc_{sym}}(x,y),y) & = & x \\ \mathsf{dec_{asym}}(\mathsf{enc_{asym}}(x,\mathsf{pk}(y)),\mathsf{sk}(y)) & = & x \\ \mathsf{msg}(\mathsf{sign}(x,y)) & = & x \\ \mathsf{ver}(\mathsf{sign}(x,\mathsf{sk}(y)),x,\mathsf{pk}(y)) & = & \mathsf{true} \\ \mathsf{blindver}(\mathsf{unblind}(\mathsf{blindsign}(\mathsf{blind}(x,z), & & \\ \qquad \mathsf{sk}(y)),z),x,\mathsf{pk}(y)) & = & \mathsf{true} \\ \mathsf{blindmsg}(\mathsf{unblind}(\mathsf{blindsign}(\mathsf{blind}(x,z),y),z)) & = & x \\ \mathsf{eq}(x,x) & = & \mathsf{true} \\ \wedge(\mathsf{true},\mathsf{true}) & = & \mathsf{true} \\ \vee(\mathsf{true},x) & = & \mathsf{true} \\ \vee(x,\mathsf{true}) & = & \mathsf{true} \end{array}$$

$$\begin{array}{lcll} \mathsf{Public}_p(\mathsf{ZK}_{i,j}(\widetilde{M},\widetilde{N},F)) & = & N_p & \text{with } p \in [1,j] \\ \mathsf{Formula}(\mathsf{ZK}_{i,j}(\widetilde{M},\widetilde{N},F)) & = & F & \end{array}$$

where Public_p and Formula constitute functions of arity 1. Since there is no destructor associated to the statement's private component, the terms $\widetilde{M}$ are kept secret. This models the *zero-knowledge* property discussed in the introduction. We define a statement $\mathsf{ZK}_{i,j}(\widetilde{M},\widetilde{N},F)$ to hold true if F is an (i,j)-formula and the formula obtained by substituting all α_k's and β_l's in F with the corresponding values M_k and N_l is valid. Verification of a statement $\mathsf{ZK}_{i,j}$ with respect to a formula is modelled as a function $\mathsf{Ver}_{i,j}$ of arity 2 that is defined by the following equational rule:

$$\begin{array}{ll} \mathsf{Ver}_{i,j}(F,\mathsf{ZK}_{i,j}(\widetilde{M},\widetilde{N},F)) = \mathsf{true} & \text{iff} \\ \quad 1) \quad E_{\mathsf{ZK}} \vdash F\{\widetilde{M}/\widetilde{\alpha}\}\{\widetilde{N}/\widetilde{\beta}\} = \mathsf{true} & \\ \quad 2) \quad F \text{ is an } (i,j)\text{-formula} & \end{array}$$

where $\{\widetilde{M}/\widetilde{\alpha}\}\{\widetilde{N}/\widetilde{\beta}\}$ denotes the substitution of each α_k with M_k and of each β_l with

Table 5 The equational theory for zero-knowledge, given a base theory $(\Sigma_{\mathsf{base}}, E_{\mathsf{base}})$

$$\Sigma_{\mathsf{ZK}} = \Sigma_{\mathsf{base}} \cup \left\{ \begin{array}{l} \mathsf{ZK}_{i,j}, \mathsf{Ver}_{i,j}, \mathsf{Public}_i, \mathsf{Formula}, \\ \alpha_i, \beta_i, \mathsf{true} \mid i,j \in \mathbb{N} \end{array} \right\}$$

$\mathsf{ZK}_{i,j}$ of arity $i+j+1$, $\mathsf{Ver}_{i,j}$ of arity 2, Public_i and $\mathsf{Formula}$ of arity 1, α_i, β_i and true of arity 0.

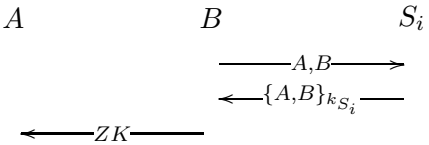
E_{ZK} is the smallest equational theory satisfying the equations of E_{base} and the following equations defined over all terms $\widetilde{M}, \widetilde{N}, F$:

$$\begin{array}{lcll} \mathsf{Public}_p(\mathsf{ZK}_{i,j}(\widetilde{M}, \widetilde{N}, F)) & = & N_p & \text{with } p \in [1,j] \\ \mathsf{Formula}(\mathsf{ZK}_{i,j}(\widetilde{M}, \widetilde{N}, F)) & = & F & \\ \mathsf{Ver}_{i,j}(F, \mathsf{ZK}_{i,j}(\widetilde{M}, \widetilde{N}, F)) & = & \mathsf{true} & \text{iff} \\ \quad 1) \quad E_{\mathsf{ZK}} \vdash F\{\widetilde{M}/\widetilde{\alpha}\}\{\widetilde{N}/\widetilde{\beta}\} = \mathsf{true} & & & \\ \quad 2) \quad F \text{ is an } (i,j)\text{-formula} & & & \end{array}$$

N_l. This rule guarantees in the abstract model the *soundness* and *correctness* of zero-knowledge protocols.

3.3 An Illustrating Example

We illustrate the zero-knowledge abstraction by means of the following example protocol. We keep the protocol simplistic in order to focus on the usage of zero-knowledge proofs; in particular, we ignore vulnerabilities due to replay attacks and corresponding countermeasures such as nonces and timestamps.



Party B receives a signed message $\{A, B\}$ from some server $S_i \in \{S_1, \ldots, S_n\}$. (This signed message might, e.g., serve as a certificate that allows B to prove that he has been authorized to contact A.) While B should be able to convince A that he owns a signature on this message issued by one of the possible n servers, the protocol should ensure that A does not learn which server S_i in fact issued the signature. This prevents B from simply forwarding the signed message to A. Instead, B proves knowledge of such a signature by a non-interactive zero-knowledge proof ZK.

We now carefully examine the proof of knowledge ZK. We aim at formalizing the following statement: "There exists α such that α is a signature of A and B, and this signature was created using one of the signature keys $k_{S_1}, \ldots, k_{S_n}$". Coming up with a formalization of this statement first requires us to tell the secret terms from the terms leaked to the verifier. The identifiers of A and B clearly have to be revealed since the proof intends to allow B to prove that he has been authorized to contact A. The signature

itself and the corresponding verification key $\mathsf{pk}(k_{S_i})$, however, have to be kept secret to preserve the anonymity of S_i. These requirements are cast in our zero-knowledge notation as follows:

$$ZK = \mathsf{ZK}_{2,n+1}\left(\begin{array}{l}\mathsf{sign}(\mathsf{pair}(A,B),\mathsf{sk}(k_{S_i})),\mathsf{pk}(k_{S_i});\\ \mathsf{pk}(k_{S_1}),\ldots,\mathsf{pk}(k_{S_n}),\mathsf{pair}(A,B);\\ \Big(\bigvee_{i=1,n}\alpha_2=\beta_i\Big)\wedge\mathsf{ver}(\alpha_1,\beta_{n+1},\alpha_2))\end{array}\right)$$

This statement captures that the signature $\mathsf{sign}(\mathsf{pair}(A,B),\mathsf{sk}(k_{S_i}))$ and the public key $\mathsf{pk}(k_{S_i})$ used in the verification are kept secret (i.e., the identity of S_i is not revealed) while the proof reveals the public keys of all servers (this includes $\mathsf{pk}(k_{S_i})$ but does not tell it from the remaining public keys) as well as the identifiers of A and B. The formula states that the verification key of the signature belongs to the set $\{\mathsf{pk}(k_{S_1}),\ldots,\mathsf{pk}(k_{S_n})\}$, and that the signed message consists of a pair composed of the identifiers of A and B. We obtain the following description of a single protocol run:

$$\begin{array}{lcl}A & \triangleq & a(y).\mathit{if}\ \mathit{Test}\ \mathit{then}\ \overline{b}\langle ok\rangle\ \mathit{else}\ \overline{b}\langle error_A\rangle\\ B & \triangleq & a(x).\mathit{if}\ (\mathsf{ver}(x,\mathsf{pair}(A,B),\mathsf{pk}(k_{S_i})))\\ & & \mathit{then}\ \overline{a}\langle ZK\rangle\ \mathit{else}\ \overline{b}\langle error_B\rangle\\ S_i & \triangleq & \overline{a}\langle\mathsf{sign}(\mathsf{pair}(A,B),\mathsf{sk}(k_{S_i}))\rangle\\ \mathit{Prot} & \triangleq & \nu k_A.\nu k_B.\nu k_{S_1}.\ldots.\nu k_{S_n}.\\ & & \overline{a}\langle\mathsf{pk}(k_{S_1})\rangle.\ldots.\overline{a}\langle\mathsf{pk}(k_{S_n})\rangle.(A\mid B\mid S_i)\end{array}$$

where *Test* constitutes the following condition:

$$\mathsf{Ver}_{2,n+1}\Big(\Big(\bigvee_{i=1,n}\alpha_2=\beta_i\Big)\wedge\mathsf{ver}(\alpha_1,\beta_{n+1},\alpha_2),y\Big)=\mathsf{true}$$
$$\bigwedge_{i=1,n}\mathsf{Public}_i(y)=\mathsf{pk}(k_{S_i})\wedge\mathsf{Public}_{n+1}(y)=\mathsf{pair}(A,B)$$

We wrote *Test* using conjunctions only to increase readability; *Test* can be straightforwardly encoded in the syntax of the calculus by a sequence of conditionals.

4 Towards a Mechanized Analysis of Zero-Knowledge

The equational theory Σ_{ZK} defined in the previous section is not suitable for existing tools for mechanized security protocol analysis. The reason is that the signature Σ_{ZK}, and consequently the number of equations in the specification, is infinite since, for instance, we assume a different $\mathsf{ZK}_{i,j}$ constructor for each possible arity. In this section, we specify an equivalent equational theory in terms of a convergent rewriting system. This theory turns out to be suitable for ProVerif [6], a well-established tool for mechanized verification of different security properties of cryptographic protocols specified in a variant of the applied pi-calculus.

4.1 A Finite Specification of Zero-Knowledge

The central idea of our equivalent finite theory is to focus on the zero-knowledge proofs used within the process specification and to abstract away from the additional ones that are possibly generated by the environment. This makes finite both the signature and the specification of the equational theory.

Pinning down this conceptually elegant and appealing idea requires to formally characterize the zero-knowledge proofs generated, verified, and read in the process specification. First, we track the zero-knowledge proofs generated or verified in the process specification by a set $\mathcal{F}$ of triples of the form (i, j, F), where i is the arity of the private component, j the arity of the public component, and F the formula. Second, we record the arity h (resp. l) of the largest private component (resp. public component) of zero-knowledge proofs used in the process specification. For terms M and processes P, we let $\mathit{terms}(M)$ denote the set of subterms of M and $\mathit{terms}(P)$ denote the set of terms in P. We can now formally define the notion of $(\mathcal{F}, h, l)$-validity of terms and processes.

Definition 5 (Process Validity) *A term Z is $(\mathcal{F}, h, l)$-valid if and only if the following conditions hold:*

1. *for every $\mathsf{ZK}_{i,j}(\widetilde{M}, \widetilde{N}, F) \in \mathit{terms}(Z)$ and $\mathsf{Ver}_{i,j}(F, M) \in \mathit{terms}(Z)$,*
   (a) *F is an (i, j)-formula and $(i, j, F) \in \mathcal{F}$,*
   (b) *$F \in T_{\Sigma_{\mathsf{base}} \cup \{\alpha_k, \beta_l \mid k \in [1,i], l \in [1,j]\}}$,*
   (c) *and for every $(i, j, F') \in \mathcal{F}$ such that $E_{\mathsf{ZK}} \vdash F = F'$, we have $F = F'$.*
2. *For every $k \in \mathbb{N}$, α_k and β_k occur in Z only inside of the last argument of some $\mathsf{ZK}_{i,j}$ or $\mathsf{Ver}_{i,j}$ function.*
3. *for every $(i, j, F) \in \mathcal{F}$, we have $i \in [0, h]$ and $j \in [0, l]$.*
4. *for every $\mathsf{Public}_p(M) \in \mathit{terms}(Z)$, we have $p \in [1, l]$.*

A process P is $(\mathcal{F}, h, l)$-valid if and only if M is $(\mathcal{F}, h, l)$-valid for every $M \in \mathit{terms}(P)$, the private arity of P is less or equal than h, and the public arity of P is less or equal than l.

We check that each zero-knowledge proof generation and verification is tracked in $\mathcal{F}$ (condition 1a). For the sake of simplicity, we prevent the occurrence of zero-knowledge operators within formulas in the process specification (condition 1b). Without loss of generality, we also require that equivalent formulas occurring in zero-knowledge proofs of the same arity are syntactically equal (condition 1c) and that the α_i's and β_j's only occur within formulas (condition 2). Finally, we check that the arity of private and public components of zero-knowledge proofs used in the process specification is less or equal than h and l, respectively (conditions 3 and 4).

Given an $(\mathcal{F}, h, l)$-valid process, we can easily define a finite equational theory $E_{\mathsf{FZK}}^{\mathcal{F},h,l}$ for $(\mathcal{F}, h, l)$-valid terms by a convergent rewriting system. For any $(i, j, F) \in \mathcal{F}$, we

Table 6 The finite equational theory for zero-knowledge with respect to an $(\mathcal{F}, h, l)$-valid process, given a base theory $(\Sigma_{\mathsf{base}}, E_{\mathsf{base}})$

$$\Sigma_{\mathsf{FZK}}^{\mathcal{F},h,l} = \Sigma_{\mathsf{base}} \cup \left\{ \begin{array}{l} \mathsf{ZK}_{i,j}^F, \mathsf{PZK}_{i,j}^F, \mathsf{Ver}_{i,j}^F, \mathsf{FakeZK}_k, \mathsf{Public}_p, \\ \mathsf{Formula}, \mathsf{FakeCollect}, \mathsf{FakePublic}, \mathsf{FakeVer}, \alpha_g, \beta_p \\ \mid (i,j,F) \in \mathcal{F}, g \in [1,h], \quad k \in [0,l], p \in [1,l] \end{array} \right\}$$

$\mathsf{PZK}_{i,j}^F$ of arity $i+j+1$, $\mathsf{ZK}_{i,j}^F$ of arity $i+j$, FakeZK_k of arity $k+2$, $\mathsf{FakeVer}$ of arity 4, $\mathsf{FakePublic}$ and $\mathsf{FakeCollect}$ of arity 2, $\mathsf{Ver}_{i,j}^F$, Public_p, and $\mathsf{Formula}$ of arity 1, α_g and β_p of arity 0. $\mathsf{PZK}_{i,j}^F$ is private.

$E_{\mathsf{FZK}}^{\mathcal{F},h,l}$ is the smallest equational theory satisfying the equations of E_{base} and the following equations for every $(i,j,F) \in \mathcal{F}$:

$$\begin{array}{lcll} \mathsf{ZK}_{i,j}^F(\widetilde{x},\widetilde{y}) & = & \mathsf{PZK}_{i,j}^F(\widetilde{x},\widetilde{y},F\{\widetilde{x}/\widetilde{\alpha}\}\{\widetilde{y}/\widetilde{\beta}\}) & \\ \mathsf{Ver}_{i,j}^F(\mathsf{PZK}_{i,j}^F(\widetilde{x},\widetilde{y},\mathsf{true})) & = & \mathsf{true} & \\ \mathsf{Public}_p(\mathsf{PZK}_{i,j}^F(\widetilde{x},\widetilde{y},z)) & = & y_p & p \in [1,j] \\ \mathsf{Formula}(\mathsf{PZK}_{i,j}^F(\widetilde{x},\widetilde{y},z)) & = & F & \\ \mathsf{Public}_p(\mathsf{FakeZK}_k(x,\widetilde{y},z)) & = & y_k & p \in [1,k], k \in [0,l] \\ \mathsf{Formula}(\mathsf{FakeZK}_k(x,\widetilde{y},z)) & = & z & k \in [0,l] \end{array}$$

include in the signature $\Sigma_{\mathsf{FZK}}^{\mathcal{F},h,l}$ the function symbols $\mathsf{ZK}_{i,j}^F$ and $\mathsf{Ver}_{i,j}^F$ of arity $i+j$ and 1, respectively. We then replace every term $\mathsf{ZK}_{i,j}(\widetilde{M},\widetilde{N},F)$ and $\mathsf{Ver}_{i,j}(F,M)$ in the process specification by $\mathsf{ZK}_{i,j}^F(\widetilde{M},\widetilde{N})$ and $\mathsf{Ver}_{i,j}^F(M)$, respectively. Since formulas are uniquely determined by the $\mathsf{ZK}_{i,j}^F$ function symbol, they can be omitted from the protocol specification. Furthermore, we need in the equational theory only those functions α_i and β_j that satisfy $i \in [1,h]$ and $j \in [1,l]$; the remaining ones can be safely omitted since they do not offer the adversary any additional capabilities. For finitely modelling the verification of zero-knowledge proofs, we include in $\Sigma_{\mathsf{FZK}}^{\mathcal{F},h,l}$ the function symbols $\mathsf{PZK}_{i,j}^F$ of arity $i+j+1$. A term $\mathsf{ZK}_{i,j}^F(\widetilde{M},\widetilde{N})$ is equivalent to $\mathsf{PZK}_{i,j}^F(\widetilde{M},\widetilde{N},F\{\widetilde{M}/\widetilde{\alpha}\}\{\widetilde{N}/\widetilde{\beta}\})$. This can be captured using a finite description, since the number of formulas in the process specification is finite:

$$\mathsf{ZK}_{i,j}^F(\widetilde{x},\widetilde{y}) \quad = \quad \mathsf{PZK}_{i,j}^F(\widetilde{x},\widetilde{y},F\{\widetilde{x}/\widetilde{\alpha}\}\{\widetilde{y}/\widetilde{\beta}\})$$

For verifying a zero-knowledge proof, it thus suffices to check whether the last argument of the $\mathsf{PZK}_{i,j}^F$ is true or not:

$$\mathsf{Ver}_{i,j}^F(\mathsf{PZK}_{i,j}^F(\widetilde{x},\widetilde{y},\mathsf{true})) \quad = \quad \mathsf{true}$$

The rule for extracting the public component is defined in the expected manner. Extracting the formula from a zero-knowledge proof $\mathsf{PZK}_{i,j}^F(\widetilde{M},\widetilde{N},F\{\widetilde{M}/\widetilde{\alpha}\}\{\widetilde{N}/\widetilde{\beta}\})$ requires an additional thought: for preserving the secrecy of private components, the function

$\mathsf{Formula}$ yields the formula F (without the substitution $\{\widetilde{M}/\widetilde{\alpha}\}\{\widetilde{N}/\widetilde{\beta}\}$) in order to prevent the adversary from deriving the formula instantiated with private terms.

$$\begin{array}{lcll} \mathsf{Public}_p(\mathsf{PZK}^F_{i,j}(\widetilde{x},\widetilde{y},z)) & = & y_p & p \in [1,j] \\ \mathsf{Formula}(\mathsf{PZK}^F_{i,j}(\widetilde{x},\widetilde{y},z)) & = & F & \end{array}$$

We obtain a finite set of rules since the number of $\mathsf{ZK}^F_{i,j}$ and $\mathsf{Ver}^F_{i,j}$ constructors corresponds to the (finite) number of formulas occurring in the process specification. The $\mathsf{PZK}^F_{i,j}$ functions are private; hence they cannot be used by the adversary to derive terms of the form $\mathsf{ZK}^F_{i,j}(\widetilde{M},\widetilde{N},\mathsf{true})$, which would be successfully verified by trusted participants regardless of the value of $F\{\widetilde{M}/\widetilde{\alpha}\}\{\widetilde{N}/\widetilde{\beta}\}$. The possibility to construct such terms would break the soundness property of zero-knowledge proofs.

It now remains to encode the zero-knowledge proofs generated by the environment. These proofs possibly contain formulas or have arities different from the ones specified in the process. We include in $\Sigma^{\mathcal{F},h,l}_{\mathsf{FZK}}$ a finite set of symbols FakeZK_k of arity $k+2$, where $k \in [0,l]$. The term $\mathsf{FakeZK}_k(M,\widetilde{N},F)$ never occurs in process specifications and represents zero-knowledge statements forged by the adversary; here M constitutes a distinguished term that uniquely refers to the zero-knowledge proof and that plays a role only in the proof of soundness, $\widetilde{N}$ denotes the first k elements of the public component, and F is the formula. The equational rules for extracting the public components and the formula from FakeZK_k terms are specified as follows:

$$\begin{array}{lcl} \mathsf{Public}_p(\mathsf{FakeZK}_k(x,\widetilde{y},z)) & = & y_k \\ \mathsf{Formula}(\mathsf{FakeZK}_k(x,\widetilde{y},z)) & = & z \end{array}$$

for any $p \in [1,k]$ and $k \in [0,l]$. We additionally include in $\Sigma^{\mathcal{F},h,l}_{\mathsf{FZK}}$ functions $\mathsf{FakeCollect}$, $\mathsf{FakePublic}$, and $\mathsf{FakeVer}$. These functions are only used for proving the finite theory equivalent to the infinite one in the next section; the functions are free in that they do not occur in any equations.

4.2 Compilation into Finite Form

We now define the static compilation of terms and processes.

Definition 6 (Static Compilation) *The* $(\mathcal{F},h,l)$-static compilation *is the partial function* $\sigma : T_{\Sigma_{\mathsf{ZK}}} \to T_{\Sigma^{\mathcal{F},h,l}_{\mathsf{FZK}}}$ *recursively defined as follows:*

$$\begin{array}{lcll} \mathsf{ZK}_{i,j}(\widetilde{M},\widetilde{N},F)\sigma & = & \mathsf{ZK}^F_{i,j}(\widetilde{M}\sigma,\widetilde{N}\sigma) & \forall (i,j,F) \in \mathcal{F} \\ \mathsf{Ver}_{i,j}(F,M)\sigma & = & \mathsf{Ver}^F_{i,j}(M\sigma) & \forall (i,j,F) \in \mathcal{F} \\ \mathsf{Public}_p(M)\sigma & = & \mathsf{Public}_p(M\sigma) & \forall p \in [1,l] \\ \mathsf{Formula}(M)\sigma & = & \mathsf{Formula}(M\sigma) & \\ \mathsf{f}(M_1,\ldots,M_i)\sigma & = & \mathsf{f}(M_1\sigma,\ldots,M_i\sigma) & \forall \mathsf{f} \in \Sigma_{\mathsf{base}} \\ x\sigma & = & x & \forall x \\ n\sigma & = & n & \forall n \end{array}$$

Table 7 Labelled transition system

$$\textsc{In}\ \frac{M \in T_{\Sigma^+}}{a(x).P \overset{a(M)}{\to} P\{M/x\}} \qquad \textsc{Out-Atom}\ \overline{a}\langle u\rangle.P \overset{\overline{a}\langle u\rangle}{\to} P \qquad \textsc{Open-Atom}\ \frac{A \overset{\overline{a}\langle u\rangle}{\to} A' \qquad u \neq a}{\nu u.A \overset{\nu u.\overline{a}\langle u\rangle}{\to} A'}$$

$$\textsc{Scope}\ \frac{A \overset{\mu}{\to} A' \qquad u \text{ does not occur in } \mu}{\nu u.A \overset{\mu}{\to} \nu u.A'} \qquad \textsc{Par}\ \frac{A \overset{\mu}{\to} A' \qquad \mathsf{bound}(\mu) \cap \mathsf{free}(B) = \emptyset}{A \mid B \overset{\mu}{\to} A' \mid B}$$

$$\textsc{Struct}\ \frac{A \equiv B \qquad B \overset{\mu}{\to} B' \qquad B' \equiv A'}{A \overset{\mu}{\to} A'}$$

Notation: Σ^+ contains the public function symbols in Σ. In Out-Atom, u is either a channel name or a variable.

The $(\mathcal{F}, h, l)$- static compilation constitutes a total function when restricted to $(\mathcal{F}, h, l)$-valid terms. The first equations deal with the compilation of zero-knowledge proofs and operators acting on them. The static compilation acts component-wise on the remaining terms and behaves as the identity function on names and variables. The compilation of a process P, written $P\sigma$, is defined by the compilation of the terms occurring therein.

The following theorem finally states that observational equivalence is preserved under static compilation and hence asserts the soundness of the encoding from the infinite specification into the finite specification. Its proof is given in the next section.

Theorem 1 (Preservation of Observational Equivalence) *Let P and Q be $(\mathcal{F}, h, l)$-valid processes and σ be the $(\mathcal{F}, h, l)$-static compilation. If $P \approx_{E_{\mathsf{ZK}}} Q$, then $P\sigma \approx_{E_{\mathsf{FZK}}^{\mathcal{F},h,l}} Q\sigma$.*

We additionally prove that a comprehensive class of trace-based properties is preserved under static compilation. We first define the notion of an execution trace. This requires to review the *labelled* operational semantics that extends the semantics given in Table 3 by allowing us to reason about processes that interact with their environment. The labelled transition system is given in Table 7.

Definition 7 (Execution Traces) *The set of execution traces of an extended process A, written traces(A), is defined as follows:*

$$\mathit{traces}(A) = \{\mu_1\phi(A_1), \ldots, \mu_n\phi(A_n) \mid A \to^* \overset{\mu_1}{\to} A_1 \ldots \to^* \overset{\mu_n}{\to} A_n\}$$

In the following, we let s range over execution traces. We now introduce the notion of trace-based security property. We assume the existence of a special channel c that is never restricted by the process. In the following, we let $B(M_1, \ldots, M_n)$ denote a

boolean formula over the terms $M_1, \ldots, M_n$: such terms are meant to express trace-based security properties. For instance, the notion of authenticity can be formalized as $\mathsf{pair}(\mathsf{end}, x) \Rightarrow \mathsf{pair}(\mathsf{begin}, x)$, where end and begin are special nullary functions.

Definition 8 (Trace-based Security Property) *A trace s satisfies the event M with substitution ξ, written $s \vdash_\xi M$ if and only if there exist s_1, s_2, N, ξ such that $s \vdash s_1 :: \overline{c}\langle N \rangle :: s_2$ and $E_{\mathsf{ZK}} \vdash N = M\xi$.*

A trace s satisfies the property $B(M_1, \ldots, M_n)$ with substitution ξ, written $s \vdash_\xi B(M_1, \ldots, M_n)$, if and only if $B(s \vdash_\xi M_1, \ldots, s \vdash_\xi M_n)$.

A process satisfies the property $B(M_1, \ldots, M_n)$, written $P \vdash_\xi B(M_1, \ldots, M_n)$, if and only if for every trace $s \in traces(P)$, there exists ξ such that $s \vdash_\xi B(M_1, \ldots, M_n)$

Finally, we can state the theorem of preservation for trace-based security properties.

Theorem 2 (Trace-based Security Property Preservation) *Let P be a $(\mathcal{F}, h, l)$-valid process, σ be the $(\mathcal{F}, h, l)$-static compilation, and $M_1, \ldots, M_n$ be $(\mathcal{F}, h, l)$-valid terms. If $P\sigma \vdash B(M_1\sigma, \ldots, M_n\sigma)$, then $P \vdash B(M_1, \ldots, M_n)$.*

4.3 Preservation of Observational Equivalence and Trace-based Security Properties

Instead of proving that observational equivalence is preserved under static compilation, we show preservation of an equivalent formulation of observational equivalence based on static equivalence and labelled bisimilarity. We first review these notions.

Definition 9 (Term Equality in Frames) *Two terms M and N are equal in a frame ϕ, written $(M = N)\phi$, if and only if $\phi \equiv \nu\widetilde{n}.\sigma$, $M\sigma = N\sigma$, and $\{\widetilde{n}\} \cap (fn(M) \cup fn(N)) = \emptyset$ for some names $\widetilde{n}$ and substitution σ.*

Definition 10 (Static Equivalence) *Two closed frames ϕ and ψ are statically equivalent, written $\phi \approx^s \psi$ if and only if $dom(\phi) = dom(\psi)$ and for all terms M and N, it holds that $(M = N)\phi$ if and only if $(M = N)\psi$.*

We say that two closed extended processes are statically equivalent, written $A \approx^s B$ if and only if their frames are statically equivalent.

We now define the notion of *labelled bisimilarity*, which constitutes an equivalent notion of observational equivalence. Labelled bisimilarity does not rely on the universal quantification over evalution contexts used in the definition of observational equivalence.

Definition 11 (Labelled Bisimilarity) *Labelled bisimilarity ($\approx^l$) is the largest symmetric relation $\mathcal{R}$ on closed extended processes such that $A\mathcal{R}B$ implies:*

1. *$A \approx^s B$;*
2. *if $A \rightarrow A'$, then $B \rightarrow^* B'$ and $A'\mathcal{R}B'$ for some B';*
3. *if $A \xrightarrow{\mu} A'$ and $fv(\mu) \subseteq dom(A)$ and $bn(\mu) \cap fn(B) = \emptyset$, then $B \rightarrow^* \xrightarrow{\mu} \rightarrow^* B'$ and $A'\mathcal{R}B'$ for some B'.*

We finally state the well-known equivalence between observational equivalence and labelled bisimilarity.

Theorem 3 (Observational Equivalence and Labelled Bisimilarity) *Observational equivalence coincides with labelled bisimilarity:* $\approx = \approx^l$.

It hence remains to be shown that labelled bisimilarity is preserved under static compilation. In the following, we write $P \equiv_E Q$ to emphasize that P and Q are structurally equivalent with respect to an equational theory E. Furthermore, we write $M\phi$ for the ground term obtained by repeated application of the substitution in ϕ to M, where we assume that $fv(M) \subseteq fv(\phi) \cup bv(\phi)$. This notation is well-defined since frames do not contain substitutions with cyclic dependencies. The next definition introduces a normal form for terms. Intuitively, a term is in $(\mathcal{F}, h, l)$-normal form if the subterms generated by the environment cannot be further simplified (conditions 1 and 2) and, in the case of zero-knowledge proofs, they either comply with the process specification or belong to a different equivalence class (condition 3).

Definition 12 (Normal Form) *A term $M \in T_{\Sigma_{\mathsf{ZK}}}$ is in $(\mathcal{F}, h, l)$-normal form with respect to a frame ϕ if and only if the following conditions hold:*

1. *for every $\mathsf{Public}_j(Z) \in terms(M)$, $i, j', \widetilde{M}, \widetilde{N}, F$ such that $j > l$ and $E_{\mathsf{ZK}} \vdash Z\phi = \mathsf{ZK}_{i,j'}(\widetilde{M}, \widetilde{N}, F)$, we have $j' < j$.*
2. *for every $\mathsf{Ver}_{i,j}(F, Z) \in terms(M)$, $\widetilde{M}, \widetilde{N}$ such that $E_{\mathsf{ZK}} \vdash Z\phi = \mathsf{ZK}_{i,j}(\widetilde{M}, \widetilde{N}, F)$, we have that $(i, j, F) \in \mathcal{F}$.*
3. *for every $\mathsf{ZK}_{i,j}(\widetilde{M}, \widetilde{N}, F) \in terms(M), F'$ such that $(i, j, F') \in \mathcal{F}$ and $\Sigma_{\mathsf{ZK}} \vdash F = F'$, we have $F = F'$.*

For any term there exists an equivalent term in normal form.

Proposition 1 (Normal Form) *For any term $M \in T_{\Sigma_{\mathsf{ZK}}}$ and frame ϕ, there exists a term $N \in T_{\Sigma_{\mathsf{ZK}}}$ in $(\mathcal{F}, h, l)$-normal form with respect to ϕ such that $E_{\mathsf{ZK}} \vdash (M = N)\phi$.*

Proof. By an inspection of the equational rules in Table 5 and Definition 12. □

We now characterize the notion of validity of extended processes. Intuitively, an extended process is $(\mathcal{F}, h, l)$-valid if it can be separated into an $(\mathcal{F}, h, l)$-valid process and a frame where free variables, referring to output messages, are associated to $(\mathcal{F}, h, l)$-valid terms, and bound variables, referring to input messages, are associated to terms in $(\mathcal{F}, h, l)$-normal form that only contain free names and free variables.

Definition 13 (Extended Process Validity) *A frame ϕ is $(\mathcal{F}, h, l)$-valid if and only if there exist $\widetilde{n}, \widetilde{y}, \{\widetilde{Z}/\widetilde{x}\}$, with $\widetilde{y} \subseteq \widetilde{x}$, such that the following conditions hold:*

1. *$\phi = \nu\widetilde{n}.\nu\widetilde{y}.\{\widetilde{Z}/\widetilde{x}\}$.*
2. *for every $x_k \in fv(\phi)$, we have that Z_k is $(\mathcal{F}, h, l)$-valid.*

3. *for every* $x_k \in bv(\phi)$*, we have that* Z_k *is in* $(\mathcal{F},h,l)$*-normal form with respect to* ϕ *and* $\mathsf{free}(Z_k) \cap \mathsf{bound}(\phi) = \emptyset$*.*

An extended process A *is* $(\mathcal{F},h,l)$*-valid if and only if there exist* $\widetilde{n}, \widetilde{y}, \{\widetilde{M}/\widetilde{x}\}$*, with* $\widetilde{y} \subseteq \widetilde{x}$*, such that the following conditions hold:*

1. $A = \nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{Z}/\widetilde{x}\}|P)$*.*
2. $\nu\widetilde{n}.\nu\widetilde{y}.\{\widetilde{Z}/\widetilde{x}\}$ *is* $(\mathcal{F},h,l)$*-valid.*
3. P *is* $(\mathcal{F},h,l)$*-valid.*

In the following, we use $\mathsf{FakeCollect}(\widetilde{M})$ for $\widetilde{M} = M_1,\ldots,M_n$ as an abbreviation for the term $\mathsf{FakeCollect}(M_1, \mathsf{FakeCollect}(M_2,\ldots,\mathsf{FakeCollect}(M_{n-1},M_n)))$. We further consider a countable set of names that are meant to represent natural numbers, denoted i,j, and nullary functions α_i and β_j with $i > h$ and $j > l$, denoted f_{α_i} and f_{β_j}, respectively. Without loss of generality, we discipline α-renaming to guarantee that such names are never restricted in the process.

We now introduce the dynamic compilation of terms at run-time.

Definition 14 (Dynamic Compilation) *The* $(\mathcal{F},h,l)$-dynamic compilation *is the function* $\rho : T_{\Sigma_{\mathsf{ZK}}} \to T_{\Sigma_{\mathsf{FZK}}^{\mathcal{F},h,l}}$ *recursively defined as follows:*

$$
\begin{array}{lll}
\mathsf{Public}_j(M)\rho & = \mathsf{Public}_j(M\rho) & \text{if } j \in [1,l] \\
 & \quad \mathsf{FakePublic}(j, M\rho) & \text{otherwise} \\
\mathsf{ZK}_{i,j}(\widetilde{M},\widetilde{N},F)\rho & = \mathsf{ZK}^F_{i,j}(\widetilde{M}\rho,\widetilde{N}\rho) & \text{if } (i,j,F) \in \mathcal{F} \\
 & \quad \mathsf{FakeZK}_k(g,\widetilde{N}_k\rho,F\rho) & \text{otherwise} \\
 & & (k = \mathit{min}(j,l), \\
 & & \ g = \mathsf{FakeCollect}(i,j,\widetilde{M}\rho,\widetilde{N}\rho)) \\
\mathsf{Formula}(M)\rho & = \mathsf{Formula}(M\rho) & \\
\alpha_i\rho & = \alpha_i & \text{if } i \in [1,h] \\
 & \quad f_{\alpha_i} & \text{otherwise} \\
\beta_j\rho & = \beta_j & \text{if } j \in [1,l] \\
 & \quad f_{\beta_j} & \text{otherwise} \\
\mathsf{Ver}_{i,j}(F,M)\rho & = \mathsf{Ver}^F_{i,j}(F\rho,M\rho) & \text{if } (i,j,F) \in \mathcal{F} \\
 & \quad \mathsf{FakeVer}(i,j,F\rho,M\rho) & \text{otherwise} \\
\mathsf{f}(M_1,\ldots,M_i)\rho & = \mathsf{f}(M_1\rho,\ldots,M_i\rho) & \forall \mathsf{f} \in \Sigma_{\mathsf{base}} \\
x\rho & = x & \forall x \\
n\rho & = n & \forall n
\end{array}
$$

The next proposition states that ρ is closed under variable substitution.

Proposition 2 (Closure of Dynamic Compilation) *Let ρ be the $(\mathcal{F},h,l)$-dynamic compilation. For every frame ϕ and every term M in $(\mathcal{F},h,l)$-normal form with respect to ϕ, we have $(M\rho)\phi\rho = (M\phi)\rho$*

Proof. By an inspection of Definition 14 and Definition 12. □

We next lemma states that term equality is preserved by dynamic compilation.

Lemma 1 (Preservation of Term Equality) *Let ϕ be an $(\mathcal{F},h,l)$-valid frame and ρ be the $(\mathcal{F},h,l)$-dynamic compilation. Then for any ground terms $M_1, M_2 \in T_{\Sigma_{\mathsf{ZK}}}$ in $(\mathcal{F},h,l)$-normal form with respect to ϕ, we have $E_{\mathsf{ZK}} \vdash M_1 = M_2 \Leftrightarrow E_{\mathsf{FZK}}^{\mathcal{F},h,l} \vdash M_1\rho = M_2\rho$.*

Proof. We prove the $\Rightarrow$ implication by induction on the length of the derivation of M_1. We first discuss the interesting base cases:

$M_1 = \mathsf{Public}_k(\mathsf{ZK}_{i,j}(\widetilde{M},\widetilde{N},F))$, $M_2 = N_k$ We have two cases:

1. $(i,j,F) \in \mathcal{F}$: By definition of ρ (cf. Definition 14), we get $M_1\rho = \mathsf{Public}_k(\mathsf{ZK}_{i,j}^F(\widetilde{M}\rho,\widetilde{N}\rho))$. By definition of $E_{\mathsf{FZK}}^{\mathcal{F},h,l}$ (cf. Table 6), we get $E_{\mathsf{FZK}}^{\mathcal{F},h,l} \vdash \mathsf{Public}_k(\mathsf{ZK}_{i,j}^F(\widetilde{M}\rho,\widetilde{N}\rho)) = N_k\rho$, as desired.
2. $(i,j,F) \notin \mathcal{F}$: By definition of ρ, we can derive that $M_1\rho = \mathsf{Public}_k(\mathsf{FakeZK}_{\mathsf{min}(j,l)}(\mathsf{FakeCollect}(i,j,\widetilde{M}\rho,\widetilde{N}\rho),\widetilde{N}_{1\mathsf{min}(j,l)},F\rho)$. By definition of $E_{\mathsf{FZK}}^{\mathcal{F},h,l}$, $E_{\mathsf{FZK}}^{\mathcal{F},h,l} \vdash M_1\rho = N_k\rho$, as desired.

$M_1 = \mathsf{Formula}(\mathsf{ZK}_{i,j}(\widetilde{M},\widetilde{N},F))$, $M_2 = F$ We have two cases:

1. $(i,j,F) \in \mathcal{F}$: By definition of ρ, we get $M_1\rho = \mathsf{Formula}(\ \mathsf{ZK}_{i,j}^F(\widetilde{M}\rho,\widetilde{N}\rho))$. By definition of $E_{\mathsf{FZK}}^{\mathcal{F},h,l}$, we get $E_{\mathsf{FZK}}^{\mathcal{F},h,l} \vdash \mathsf{Formula}(\mathsf{ZK}_{i,j}^F(\widetilde{M}\rho,\widetilde{N}\rho)) = F$ and, since M_1 is in $(\mathcal{F},h,l)$-normal form with respect to ϕ, by definition of ρ we have that $F\rho = F$, as desired.
2. $(i,j,F) \notin \mathcal{F}$: By Definition 14, we obtain $M_1\rho = \mathsf{Formula}(\ \mathsf{FakeZK}_{\mathsf{min}(j,l)}(\mathsf{FakeCollect}(i,j,\widetilde{M}\rho,\widetilde{N}\rho),\widetilde{N}_{1\mathsf{min}(j,l)},F\rho)$. By an inspection of Table 6, we have $E_{\mathsf{FZK}}^{\mathcal{F},h,l} \vdash M_1\rho = F\rho$, as desired.

$M_1 = \mathsf{Ver}(F,\mathsf{ZK}_{i,j}(\widetilde{M},\widetilde{N},F))$ and $M_2 = \mathsf{true}$ It must be the case that $(i,j,F) \in \mathcal{F}$, otherwise M_1 is not in $(\mathcal{F},h,l)$-normal form with respect to ϕ. By definition of ρ, $M_1\rho = \mathsf{Ver}_{i,j}^F(\mathsf{ZK}_{i,j}^F(\widetilde{M}\rho,\widetilde{N}\rho))$. By the equational theory of Table 6, $E_{\mathsf{FZK}}^{\mathcal{F},h,l} \vdash M_1\rho = \mathsf{true}$, as desired.

We prove the induction step by cases:

Symmetry We have that $E_{\mathsf{ZK}} \vdash M_1 = M_2$ is proved by symmetry from $E_{\mathsf{ZK}} \vdash M_2 = M_1$. By induction hypothesis, $E_{\mathsf{FZK}}^{\mathcal{F},h,l} \vdash M_2\rho = M_1\rho$. The result follows by symmetry of $E_{\mathsf{FZK}}^{\mathcal{F},h,l}$.

Transitivity The result follows directly from the induction hypothesis.

The proof of the $\Leftarrow$ implication is similar and relies on the fact that ρ is injective when applied to terms in $(\mathcal{F},h,l)$-normal form with respect to ϕ. $\square$

Exploiting that term equality is preserved under dynamic compilation, we proceed by showing the preservation of process reduction. The following lemma also proves that the validity of extended processes is preserved by internal reduction and labelled transition, up to structural equivalence. In addition, Theorem 2 constitutes a direct consequence of this lemma.

Lemma 2 (Preservation of Process Reduction) *Let A be an extended process such that $A \equiv_{E_{\mathsf{ZK}}} \nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\}|P)$, for some $(\mathcal{F},h,l)$-valid extended process $\nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\}|P)$, let σ be the $(\mathcal{F},h,l)$ static compilation, and let ρ be the $(\mathcal{F},h,l)$-dynamic compilation. Then the following statements hold:*

1. *For every B, $A \rightarrow_{E_{\mathsf{ZK}}} B$ if and only if there exists an $(\mathcal{F},h,l)$-valid extended process $\nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M'}/\widetilde{x'}\}|P') \equiv_{E_{\mathsf{ZK}}} B$ such that $\nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\}\rho|P\sigma) \rightarrow_{E_{\mathsf{FZK}}^{\mathcal{F},h,l}} \nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\}\rho|P'\sigma)$.*
2. *For every μ containing only terms in $(\mathcal{F},h,l)$-normal form with respect to $\nu\widetilde{n}.\nu\widetilde{y}.\{\widetilde{M}/\widetilde{x}\}$ and every B, $A \xrightarrow{\mu}_{E_{\mathsf{ZK}}} B$ if and only if there exists an $(\mathcal{F},h,l)$-valid extended process $\nu\widetilde{n'}.\nu\widetilde{y'}.(\{\widetilde{M'}/\widetilde{x'}\}|P') \equiv_{E_{\mathsf{ZK}}} B$ such that $\nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\}\rho|P\sigma) \xrightarrow{\mu\rho}_{E_{\mathsf{FZK}}^{\mathcal{F},h,l}} \nu\widetilde{n'}.\nu\widetilde{y'}.(\{\widetilde{M'}/\widetilde{x'}\}\rho|P'\sigma)$ where*
   - *if $\mu = a(M)$, then $\widetilde{n} = \widetilde{n'}, \widetilde{y'} = \widetilde{y}, x$, for some $x \notin \{\widetilde{x}\}$, and $\{\widetilde{M'}/\widetilde{x'}\} = \{\widetilde{M}/\widetilde{x}\} \mid \{M/x\}$.*
   - *if $\mu = \overline{a}\langle b\rangle$, then $\widetilde{n} = \widetilde{n'}, \widetilde{y'} = \widetilde{y}$, and $\{\widetilde{M'}/\widetilde{x'}\} = \{\widetilde{M}/\widetilde{x}\}$.*
   - *if $\mu = \nu b.\overline{a}\langle b\rangle$, then $\widetilde{n} = (\widetilde{n'}, b), \widetilde{y'} = \widetilde{y}$, and $\{\widetilde{M'}/\widetilde{x'}\} = \{\widetilde{M}/\widetilde{x}\}$.*
   - *if $\mu = \nu x.\overline{a}\langle x\rangle$, then $\widetilde{n} = \widetilde{n'}, \widetilde{y'} = \widetilde{y}$, and $\{\widetilde{M'}/\widetilde{x'}\} = \{\widetilde{M}/\widetilde{x}\} \mid \{M/x\}$, for some $(\mathcal{F},h,l)$-valid M.*

Proof. We prove statement 1 by cases on the internal reduction rule. Let us first deal with the "only if" implication.

Comm By an inspection of Table 2, there exist M, x, Q, P_1, P_2 such that $\nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\} \mid P) \equiv_{E_{\mathsf{ZK}}} \nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\}| \ Q \mid \overline{a}\langle M\rangle.P_1 \mid a(x).P_2)$ and $Q \mid \overline{a}\langle M\rangle.P_1 \mid a(x).P_2$ is $(\mathcal{F},h,l)$-valid. By α-renaming, we can assume that $x \notin fv(P_1)$. We also have that $B \equiv_{E_{\mathsf{ZK}}} \nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\}| \ Q \mid P_1| \ P_2\{M/x\})$.

By Alias, Res-Par, and Subst, we get $\nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\} \mid P) \equiv_{E_{\mathsf{ZK}}} \nu\widetilde{n}.\nu\widetilde{y}.\nu x.(\{\widetilde{M}/\widetilde{x}\} \mid \{M/x\} \mid Q \mid \overline{a}\langle x\rangle.P_1| \ a(x).P_2)$. Since σ behaves as the identity function on variables and names and it is defined on $(\mathcal{F},h,l)$-valid terms and

processes, we get $\nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\}\ \rho \mid P\sigma) \equiv_{E_{\mathsf{FZK}}^{\mathcal{F},h,l}} \nu\widetilde{n}.\nu\widetilde{y}.\nu x.(\{\widetilde{M}/\widetilde{x}\}\ \rho \mid \{M\sigma/x\} \mid Q\sigma \mid \overline{a}\langle x\rangle.P_1\sigma \mid a(x).P_2\sigma)$. By Comm, Subst, Res-Par, and Alias, we have that $\nu\widetilde{n}.\nu\widetilde{y}.\nu x.(\{\widetilde{M}/\widetilde{x}\}\ \rho \mid \{M\sigma/x\} \mid Q\sigma \mid \overline{a}\langle x\rangle.P_1\sigma \mid a(x).P_2\sigma) \rightarrow \nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\} \mid P_1\sigma \mid P_2\{M/x\}\sigma)$, as desired. Notice that internal reduction is closed by structural equivalence. It is easy to see that $P_2\{M/x\}$ is $(\mathcal{F},h,l)$-valid since M occurs in the $(\mathcal{F},h,l)$-valid process $\overline{a}\langle M\rangle.P_1$ and it is thus $(\mathcal{F},h,l)$-valid as well.

Then By an inspection of Table 2, there exist M,N,Q,P_1,P_2 such that $\nu n.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\} \mid P) \equiv_{E_{\mathsf{ZK}}} \nu n.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\} \mid Q \mid \mathit{if}\,(M = N)\ \mathit{then}\ P_1\ \mathit{else}\ P_2$, for some M,N,P_1,P_2,Q such that the process $Q \mid \mathit{if}\, M = N\ \mathit{then}\ P_1\ \mathit{else}\ P_2$ is $(\mathcal{F},h,l)$-valid and $\Sigma_{\mathsf{ZK}} \vdash M\{\widetilde{M}/\widetilde{x}\} = N\{\widetilde{M}/\widetilde{x}\}$. We also have that $B \equiv_{E_{\mathsf{ZK}}} P_2$. Similarly, by applying Subst, we get $\nu n.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\}\rho \mid P\sigma) \equiv_{E_{\mathsf{ZK}}} \nu n.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\}\rho \mid Q\{\widetilde{M}/\widetilde{x}\}\rho \mid (\mathit{if}\,(M = N)\ \mathit{then}\ P_1\ \mathit{else}\ P_2)\sigma\{\widetilde{M}/\widetilde{x}\}\rho)$. Since $\widetilde{M}$ is in $(\mathcal{F},h,l)$-normal form with respect to $\nu n.\nu\widetilde{y}.\{\widetilde{M}/\widetilde{x}\}$ and M is $(\mathcal{F},h,l)$-valid, it is easy to see that $E_{\mathsf{FZK}}^{\mathcal{F},h,l} \vdash (M\sigma)\{\widetilde{M}/\widetilde{x}\}\rho = (M\{\widetilde{M}/\widetilde{x}\})\rho$ and $(M\{\widetilde{M}/\widetilde{x}\})\rho$ is in $(\mathcal{F},h,l)$-normal form with respect to $\nu n.\nu\widetilde{y}.\{\widetilde{M}/\widetilde{x}\}$. The reasoning is the same for N. By Lemma 1, we get $E_{\mathsf{FZK}}^{\mathcal{F},h,l} \vdash M\{\widetilde{M}/\widetilde{x}\}\rho = N\{\widetilde{M}/\widetilde{x}\}\rho$. The result follows from Then and structural equivalence.

Else The reasoning is similar to the one in the previous item.

Notice that the previous cases cover both the application of evaluation contexts and the closure by structural equivalence. The proof for the "if" implication is similar and relies on the fact that ρ is injective when applied to terms in $(\mathcal{F},h,l)$-normal form.

We now prove that process reduction, as defined by the labelled transition systems, is preserved as well. We proceed by cases on the label μ:

$\mu = a(M)$ By an inspection of Table 7, there exist x,P',Q such that $\nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\} \mid P) \equiv_{E_{\mathsf{ZK}}} \nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\} \mid a(x).P' \mid Q)$ and $B \equiv_{E_{\mathsf{ZK}}} \nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\} \mid P'\{M/x\} \mid Q)$. Similarly, we have that $\nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\}\rho \mid P\sigma) \equiv_{E_{\mathsf{ZK}}} \nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\}\rho \mid a(x).P'\sigma \mid Q\sigma)$. By α-renaming, we can assume $x \notin \widetilde{x}$ and, by Scope, we derive $\mathsf{free}(M) \cap \{\widetilde{n},\widetilde{y}\} = \emptyset$. By In, Alias, Subst, and Res-Par, we get $\nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\}\rho \mid a(x).P'\sigma \mid Q\sigma) \xrightarrow{\mu} \nu\widetilde{n}.\nu\widetilde{y}.\nu x.(\{M/x\}\rho \mid \{\widetilde{M}/\widetilde{x}\}\rho \mid P'\sigma \mid Q\sigma)$.

$\mu = \overline{a}\langle b\rangle$ The output term is a free channel. We have that $\nu\widetilde{n}.\nu\widetilde{y}.\ (\{\widetilde{M}/\widetilde{x}\} \mid P) \equiv_{E_{\mathsf{ZK}}} \nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\} \mid \overline{a}\langle b\rangle.P' \mid Q)$ and $B \equiv_{E_{\mathsf{ZK}}} \nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\} \mid P' \mid Q)$. By Scope, $a \notin \widetilde{n}$, and $b \notin \widetilde{n}$. Similarly, we have that $\nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\}\rho \mid P\sigma) \equiv_{E_{\mathsf{ZK}}} \nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\}\rho \mid \overline{a}\langle b\rangle.P'\sigma \mid Q\sigma)$. The result follows from Out-Atom and Scope.

$\mu = \nu b.\overline{a}\langle b\rangle$ The output term is a private channel. We have that $\nu\widetilde{n},b.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\} \mid P) \equiv_{E_{\mathsf{ZK}}} \nu\widetilde{n},b.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\} \mid \overline{a}\langle b\rangle.P' \mid Q)$ and $B \equiv_{E_{\mathsf{ZK}}} \nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\} \mid P' \mid Q)$. Similarly, $\nu\widetilde{n},b.\nu\widetilde{y}.\ (\{\widetilde{M}/\widetilde{x}\}\rho \mid P\sigma) \equiv_{E_{\mathsf{ZK}}} \nu\widetilde{n},b.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\}\rho \mid \overline{a}\langle b\rangle.P'\sigma \mid Q\sigma)$. The result follows from Out-Atom and Open-Atom.

$\mu = \nu x.\overline{a}\langle x\rangle$ We have that $\nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\} \mid P) \equiv_{E_{\mathsf{ZK}}} \nu\widetilde{n}.\nu\widetilde{y}.\ (\{\widetilde{M}/\widetilde{x}\} \mid \overline{a}\langle M\rangle.P' \mid Q)$, and, by Alias, Res-Par, Out-Atom, and Open-Atom, $B \equiv_{E_{\mathsf{ZK}}} \nu\widetilde{n}.\nu\widetilde{y}.(\{M/x\} \mid \{\widetilde{M}/\widetilde{x}\} \mid P' \mid Q)$, for some $x \notin \widetilde{x}$ and with $fv(M) \subseteq \widetilde{x}$. Similarly, we have that $\nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\}\rho \mid P\sigma) \equiv_{E_{\mathsf{ZK}}} \nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\}\rho \mid \overline{a}\langle M\rangle.P'\sigma \mid Q\sigma)$. The result follows from Alias, Res-Par, Out-Atom, and Open-Atom.

In all cases, it is easy to see that the resulting extended process is $(\mathcal{F}, h, l)$-valid. The proof for the "if" implication is similar and relies on the fact that ρ is injective when applied to terms in $(\mathcal{F}, h, l)$-normal form.

□

We are finally ready to prove that the dynamic compilation preserves static equivalence. We first characterize a notion of similarity for frames. The crucial ingredient of this definition is that the two frames coincide when restricted to bound variables, i.e., if the terms received as input by the corresponding extended processes coincide. This property is naturally fulfilled by the frames associated to labelled bisimilar extended processes.

The next lemma says that a test succeeds if and only if its compilation does.

Lemma 3 (Test Preservation) *Let ϕ be an $(\mathcal{F}, h, l)$-valid frame and ρ be the $(\mathcal{F}, h, l)$-dynamic compilation. For every M, N in $(\mathcal{F}, h, l)$-normal form with respect to ϕ such that $(\mathsf{free}(M) \cup \mathsf{free}(N)) \cap \mathsf{bound}(\phi) = \emptyset$, we have that $(M = N)\phi \Leftrightarrow (M\rho = N\rho)\phi\rho$.*

Proof. The proof follows from Lemma 1 and Proposition 2. □

The next definition introduces the notion of similarity for frames.

Definition 15 (Frame Similarity) *Two frames ϕ and ψ are similar, written $\phi \sim \psi$, if and only if the following conditions hold:*

1. *There exist $\mathcal{F}$, h, and l such that ϕ and ψ are $(\mathcal{F}, h, l)$-valid frames.*
2. *$\phi = \nu\widetilde{n}.\nu\widetilde{y}.\{\widetilde{M}/\widetilde{x}\}$ and $\phi = \nu\widetilde{m}.\nu\widetilde{y}.\{\widetilde{N}/\widetilde{x}\}$.*
3. *For every $x_i \in bv(\phi)$, we have $M_i = N_i$.*

The next lemma says that for testing similar frames, it suffices to only consider terms in $(\mathcal{F}, h, l)$-normal form.

Lemma 4 (Valid Tests) *Let ϕ and ψ be two $(\mathcal{F}, h, l)$-valid and similar frames. For every M, N such that $(M = N)\phi$ holds and $(M = N)\psi$ does not hold, there exist M', N' in $(\mathcal{F}, h, l)$-normal form with respect to ϕ and ψ such that $(M' = N')\phi$ holds and $(M' = N')\psi$ does not hold.*

Proof. We first show that it is possible to replace M by a term M' such that every subterm of the form $\mathsf{Public}_j(Z)$ is in $(\mathcal{F}, h, l)$-normal form with respect to ϕ.

$M = T[\mathsf{Public}_j(Z)]$ and $E_{\mathsf{ZK}} \vdash Z\phi = \mathsf{ZK}i, j'(\widetilde{M}, \widetilde{N}, F)$, with $j' \geq j > l$. By an inspection of the equational theory, we have two cases:

- There exists $Z' = \mathsf{ZK}i, j'(\widetilde{M'}, \widetilde{N'}, F') \in \mathit{terms}(Z)$ such that $E_{\mathsf{ZK}} \vdash Z'\phi = \mathsf{ZK}i, j'(\widetilde{M}, \widetilde{N}, F)$. Therefore, $E_{\mathsf{ZK}} \vdash \mathsf{Public}_j(Z)\phi = N'_j\phi$. If $E_{\mathsf{ZK}} \vdash \mathsf{Public}_j(Z)\psi = N'_j\psi$, then we can replace $\mathsf{Public}_j(Z)$ by N'_j. Otherwise, we have that $(Z = Z')\phi$ holds and $(Z = Z')\psi$ does not hold, as desired.
- There exists a variable $x \in \mathit{terms}(Z)$ such that $\mathsf{ZK}i, j'(\widetilde{M'}, \widetilde{N'}, F') \in \mathit{terms}(x\phi)$ and $E_{\mathsf{ZK}} \vdash \mathsf{ZK}i, j'(\widetilde{M'}, \widetilde{N'}, F') = \mathsf{ZK}i, j'(\widetilde{M}, \widetilde{N}, F)$. By definition 13 and definition 15, there exists a variable $y \in \mathsf{bound}(\phi) \cap \mathsf{bound}(\psi)$ and a term Z bound to y in ϕ and ψ such that $\mathsf{ZK}i, j'(\widetilde{M''}, \widetilde{N''}, F'') \in \mathit{terms}(Z)$, $\mathsf{free}(Z) \cap \mathsf{bound}(\phi) = \emptyset$, and $N''_j\phi = N_j$. If $E_{\mathsf{ZK}} \vdash \mathsf{Public}_j(Z)\psi = N''_j\psi$, then we can replace $\mathsf{Public}_j(Z)$ by N''_j. Otherwise, $(Z = \mathsf{ZK}i, j'(\widetilde{M''}, \widetilde{N''}, F''))\phi$ holds and $(Z = \mathsf{ZK}i, j'(\widetilde{M''}, \widetilde{N''}, F''))\psi$ does not hold, as desired. By Definition 13, $\mathsf{ZK}i, j'(\widetilde{M''}, \widetilde{N''}, F'')$ and N''_j are in normal form with respect to ϕ and ψ.

We can similarly prove that it is possible to remove every subterm of the form $\mathsf{Ver}_{i,j}(F, Z)$ that is not in $(\mathcal{F}, h, l)$-normal form. At the end of such a process, possibly applied to N, we get two terms M' and N' in $(\mathcal{F}, h, l)$-normal form with respect to ϕ and ψ such that $(M' = N')\phi$ holds and $(M' = N')\psi$ does not hold, as desired. □

We can now formulate the theorem stating that verifying static equivalence on frames obtained by the encoding suffices to prove static equivalence on the original frames.

Lemma 5 (Preservation of Static Equivalence) *Let ϕ and ψ be similar and $(\mathcal{F}, h, l)$-valid frames such that $dom(\phi) = dom(\psi)$. Let ρ be the $(\mathcal{F}, h, l)$-dynamic compilation. If $\phi\rho \approx^s_{E^{\mathcal{F},h,l}_{\mathsf{FZK}}} \psi\rho$ then $\phi \approx^s_{E_{\mathsf{ZK}}} \psi$.*

Proof. By Definition 10, we have to prove that $E_{\mathsf{ZK}} \vdash (M = N)\phi \Leftrightarrow E_{\mathsf{ZK}} \vdash (M = N)\psi$, for every $M, N \in T_{\Sigma_{\mathsf{ZK}}}$, only if $E^{\mathcal{F},h,l}_{\mathsf{FZK}} \vdash (M' = N')\phi\rho \Leftrightarrow E^{\mathcal{F},h,l}_{\mathsf{FZK}} \vdash (M' = N')\psi\rho$, for every $M', N' \in T_{\Sigma^{\mathcal{F},h,l}_{\mathsf{FZK}}}$. Suppose that there exist M, N such that $(M = N)\phi$ holds and $(M = N)\psi$ does not hold. By Lemma 4, we can assume that M and N are in $(\mathcal{F}, h, l)$-normal form with respect to ϕ and ψ. By Lemma 3, $(M\rho = N\rho)\phi\rho$ holds and $(M\rho = N\rho)\psi\rho$ does not hold. Therefore, $\phi\rho$ and $\psi\rho$ are not statically equivalent, yielding a contradiction.

□

The following lemma asserts that the equivalence of the terms occurring in input labels does not affect labelled bisimilarity.

Lemma 6 (Equivalent Labels) *Let A and B be extended processes such that $A \stackrel{a(M)}{\rightarrow} A'$, $B \stackrel{a(M)}{\rightarrow} B'$, and $\phi(A) \approx^s_{E_{\mathsf{ZK}}} \phi(B)$. Then for every N such that $E_{\mathsf{ZK}} \vdash M\phi(A) = N\phi(A)$ and $A \stackrel{a(N)}{\rightarrow} A'$, we have that $B \stackrel{a(N)}{\rightarrow} B'$.*

Proof. Since the frames of the two extended processes are statically equivalent, we have that $E_{\mathsf{ZK}} \vdash M\phi(B) = N\phi(B)$ and $dom(\phi(A)) = dom(\phi(B))$. Possibly after applying α-renaming on bound names, we get the result by applying In, Scope, and Struct. $\square$

We can finally show that verifying labelled bisimilarity on extended processes obtained by the compilation suffices to prove labelled bisimilarity on the original extended processes. With Theorem 3, this proves Theorem 1 as desired. In the following, for every $(\mathcal{F}, h, l)$-valid $A = \nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\}|P)$, we write $A\rho\sigma$ to denote $\nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\}\rho|P\sigma)$.

Lemma 7 (Preservation of Labelled Bisimilarity) *Let A, B be extended processes such that $A = \nu\widetilde{n}.\nu\widetilde{y}.(\{\widetilde{M}/\widetilde{x}\}|P)$, $B = \nu\widetilde{n'}.\nu\widetilde{y}.(\{\widetilde{M'}/\widetilde{x}\}|\ P')$, for some $(\mathcal{F}, h, l)$-valid P and P' and $\nu\widetilde{n}.\nu\widetilde{y}.\{\widetilde{M}/\widetilde{x}\} \sim \nu\widetilde{n'}.\nu\widetilde{y}.\{\widetilde{M'}/\widetilde{x}\}$. Let σ be the $(\mathcal{F}, h, l)$ static compilation and ρ be the $(\mathcal{F}, h, l)$-dynamic compilation. If $A\rho\sigma \approx^l_{E^{\mathcal{F},h,l}_{\mathsf{FZK}}} B\rho\sigma$, then $A \approx^l_{E_{\mathsf{ZK}}} B$.*

Proof. Since $A\rho\sigma \approx^l_{E^{\mathcal{F},h,l}_{\mathsf{FZK}}} B\rho\sigma$, we can consider the smallest symmetric relation $\mathcal{R}' \subseteq \approx^l_{E^{\mathcal{F},h,l}_{\mathsf{FZK}}}$ satisfying the conditions 1, 2, and 3 of Definition 11 and such that $A\rho\sigma\mathcal{R}'B\rho\sigma$. Given ρ and σ, let us define the relation $\mathcal{R}$ as the smallest symmetric relation satisfying the following conditions:

1. for every $(\mathcal{F}, h, l)$-valid A, B such that $A\rho\sigma\mathcal{R}'B\rho\sigma$ and $\phi(A\rho\sigma) \sim \phi(B\rho\sigma)$, we have that $A\mathcal{R}B$.
2. for every A, B, A', B' such that $A\mathcal{R}B$, $A \equiv_{E_{\mathsf{ZK}}} A'$ and $B \equiv_{E_{\mathsf{ZK}}} B'$, we have that $A'\mathcal{R}B'$.

We want to prove that $\mathcal{R}$ satisfies the conditions 1, 2, and 3 of Definition 11.

Condition 1 We want to prove that for every A, B such that $A\mathcal{R}B$, we have that $\phi(A) \approx^s_{E_{\mathsf{ZK}}} \phi(B)$. If $A\mathcal{R}B$, then there exist $(\mathcal{F}, h, l)$-valid A' and B' such that $A \equiv_{E_{\mathsf{ZK}}} A'$, $B \equiv_{E_{\mathsf{ZK}}} B'$, and $A'\rho\sigma\mathcal{R}'B'\rho\sigma$. By definition of $\mathcal{R}'$, $\phi(A'\rho\sigma) \sim \phi(B'\rho\sigma)$ and $\phi(A'\rho\sigma) \approx^s_{E^{\mathcal{F},h,l}_{\mathsf{FZK}}} \phi(B'\rho\sigma)$. It is easy to see that $\phi(A') \sim \phi(B')$. By Lemma 5, $\phi(A') \approx^s_{E_{\mathsf{ZK}}} \phi(B')$. Since structural equivalence preserves static equivalence, $\phi(A) \approx^s_{E_{\mathsf{ZK}}} \phi(B)$, as desired.

Condition 2 We want to prove that for every A, B such that $A\mathcal{R}B$, we have that (if $A \to A_1$, then $B \to^* B_1$ and $A_1\mathcal{R}B_1$ for some B'). If $A\mathcal{R}B$, then there exist $(\mathcal{F}, h, l)$-valid A' and B' such that $A \equiv_{E_{\mathsf{ZK}}} A'$, $B \equiv_{E_{\mathsf{ZK}}} B'$, and $A'\rho\sigma\mathcal{R}'B'\rho\sigma$. By Lemma 2, for every A_1 such that $A \to A_1$, there exists a $(\mathcal{F}, h, l)$-valid A'_1 such that $A' \to A'_1$, $A'_1 \equiv_{E_{\mathsf{ZK}}} A_1$, and $A'\rho\sigma \to A'_1\rho\sigma$; we can find similar B_1 and B'_1 for B and B', respectively. By Lemma 2 and Definition 15, it is easy to see that $\phi(A'_1\rho\sigma) \sim \phi(B'_1\rho\sigma)$. By definition of $\mathcal{R}$, $A'_1\mathcal{R}B'_1$ and, since ρ is closed by structural equivalence, $A_1\mathcal{R}B_1$, as desired.

Condition 3 We want to prove that for every A, B such that $A\mathcal{R}B$, we have that (if $A \xrightarrow{\mu} A_1$ and $fv(\mu) \subseteq dom(A)$ and $bn(\mu) \cap fn(B) = \emptyset$, then $B \rightarrow^* \xrightarrow{\mu} \rightarrow^* B_1$ and $A_1\mathcal{R}B_1$ for some B_1). If $A\mathcal{R}B$, then there exist $(\mathcal{F}, h, l)$-valid A' and B' such that $A \equiv_{E_{\mathsf{ZK}}} A'$, $B \equiv_{E_{\mathsf{ZK}}} B'$, and $A'\rho\sigma\mathcal{R}'B'\rho\sigma$. By Lemma 2 and Lemma 6, for every A_1 such that $A \xrightarrow{\mu} A_1$, there exists a $(\mathcal{F}, h, l)$-valid A'_1 such that $A' \xrightarrow{\mu} A'_1$, $A'_1 \equiv_{E_{\mathsf{ZK}}} A_1$, and $A'\rho\sigma \rightarrow A'_1\rho\sigma$; we can find similar B_1 and B'_1 for B and B', respectively. By Lemma 2 and Definition 15, it is easy to see that $\phi(A'_1\rho\sigma) \sim \phi(B'_1\rho\sigma)$. By definition of $\mathcal{R}$, $A'_1\mathcal{R}B'_1$ and, since ρ is closed by structural equivalence, $A_1\mathcal{R}B_1$, as desired.

Therefore $A \approx^l_{E_{\mathsf{ZK}}} B$, as desired.

□

Theorem 1 then follows directly from Lemma 7 since $\approx_l$ and $\approx$ coincide in the applied pi-calculus.

In some cases, the analysis of observational equivalence using the tool ProVerif [6] does not terminate due to the presence of the constructors $\wedge$ and $\vee$ and their equations. In these cases, it is useful to remove $\wedge$ and $\vee$ from the equational theory if they are not used in the protocol. Protocols often contain these constructors only in the formulas of zero-knowledge proofs. Then, after compilation, the protocol does not contain $\wedge$ and $\vee$ any more, but the equational theory produced by the compiler does. In these cases, the following theorem often allows to modify the equational theory produced by the compiler in such a way that $\wedge$ and $\vee$ do not occur any more:

Theorem 4 (Unfolding) *Given $\mathcal{F}, h, l$, let $E^{\mathcal{F},h,l}_{\mathsf{FZK}}$ be the equational theory defined in Table 6. Let $\mathcal{F}' \subseteq \mathcal{F}$ and $\widetilde{\tau}^1_{i,j,F}, \ldots, \widetilde{\tau}^n_{i,j,F}$ be tuples of arity $i + j$ associated to each $(i, j, F) \in \mathcal{F}'$, where $n = n_{i,j,F}$. Assume that for every (i, j, F) and tuples $\widetilde{M} = M_1, \ldots, M_{i+j}$ of arity $i + j$, we have that ($\exists k, \sigma$ such that $\widetilde{M} = \widetilde{\tau}^k_{i,j,F}\sigma \Leftrightarrow E^{\mathcal{F},h,l}_{\mathsf{FZK}} \vdash F\{\widetilde{M}_{1,i}/\alpha_{1,i}\}\{\widetilde{M}_{i+1,i+j}/\beta_{1,j}\} = \mathsf{true}$), where $\widetilde{M}_{1,i} = M_1, \ldots, M_i$ and $\widetilde{M}_{i+1,i+j} = M_{i+1}, \ldots, M_{i+j}$. Let E be obtained by replacing all the rules containing $\mathsf{PZK}^F_{i,j}$, for every $(i, j, F) \in \mathcal{F}'$, by the following set of rules:*

$$\mathsf{Ver}^F_{i,j}(\mathsf{ZK}^F_{i,j}(\widetilde{\tau}^1_{i,j,F})) = \mathsf{true}, \ldots, \mathsf{Ver}^F_{i,j}(\mathsf{ZK}^F_{i,j}(\widetilde{\tau}^n_{i,j,F})) = \mathsf{true}$$
$$\mathsf{Public}_k(\mathsf{ZK}^F_{i,j}(\widetilde{x}, \widetilde{y})) = y_k$$
$$\mathsf{Formula}(\mathsf{ZK}^F_{i,j}(\widetilde{x}, \widetilde{y})) = F$$

Then $E^{\mathcal{F},h,l}_{\mathsf{FZK}} \backslash \{(M, N) \mid \mathsf{PZK}^F_{i,j}$ occurs in M or $N \wedge (i, j, F) \in \mathcal{F}'\} = E$.

Note that $E^{\mathcal{F},h,l}_{\mathsf{FZK}} \backslash \{(M, N) \mid \mathsf{PZK}^F_{i,j}$ occurs in M or $N \wedge (i, j, F) \in \mathcal{F}'\} = E$ trivially implies the preservation of observational equivalence, since $\mathsf{PZK}^F_{i,j}$ is a private constructor not used in the protocol and thus never appears in terms produced by the protocol or the adversary.

Proof. All the equations defining $E^{\mathcal{F},h,l}_{\mathsf{FZK}}$ and depending on $\mathsf{PZK}^F_{i,j}$ have a direct counterpart in the definition of E. The only subtlety concerns the equations for the verification of zero-knowledge proofs: for every $(i, j, F) \in \mathcal{F}'$, $\widetilde{M}, \widetilde{N}$ such that $F\{\widetilde{M}_{1i}/\alpha_{1i}$,

$\widetilde{M}_{i+1,j}/\beta_{1j}\} = \mathsf{true}$, $E^{\mathcal{F},h,l}_{\mathsf{FZK}} \vdash \mathsf{Ver}^F_{i,j}(\mathsf{PZK}^F_{i,j}(\widetilde{M},\widetilde{N},\mathsf{true})) = \mathsf{true}$ can be exploited to prove $\mathsf{Ver}^F_{i,j}(\mathsf{ZK}^F_{i,j}(\widetilde{M},\widetilde{N})) = \mathsf{true}$. However, for every $(i,j,F) \in \mathcal{F}'$, $\widetilde{M},\widetilde{N}$ there exist $\tau^k_{i,j,F},\sigma$ such that $\widetilde{M},\widetilde{N} = \tau^k\sigma$ if and only if $E^{\mathcal{F},h,l}_{\mathsf{FZK}} \vdash F\{\widetilde{M}_{1i}/\alpha_{1i},\widetilde{M}_{i+1,j}/\beta_{1j}\} = \mathsf{true}$. Therefore $E^{\mathcal{F},h,l}_{\mathsf{FZK}} \vdash \mathsf{Ver}^F_{i,j}(\mathsf{ZK}^F_{i,j}(\widetilde{M},\widetilde{N})) = \mathsf{true}$ if and only if $E \vdash \mathsf{Ver}^F_{i,j}(\mathsf{ZK}^F_{i,j}(\widetilde{M},\widetilde{N})) = \mathsf{true}$. □

Additionally, after having removed all occurrences of $\wedge$ and $\vee$, we need to be able to remove their equational rules. The soundness of this transformation is shown by the following simple lemma:

Lemma 8 (Removal of $\wedge$ and $\vee$) *Let E_0 be an equational theory with signature Σ_0 and $\wedge,\vee \notin \Sigma_0$ and $\mathsf{true} \in \Sigma_0$. Let $\Sigma_1 := \Sigma_0 \cup \{\wedge,\vee\}$ and let E_1 be the smallest equational theory over Σ_1 containing E_0 and the equations $\{\wedge(\mathsf{true},\mathsf{true}) = \mathsf{true}, \vee(\mathsf{true},x) = \mathsf{true}, \vee(x,\mathsf{true}) = \mathsf{true}\}$. Let $\Sigma_2 := \Sigma_1$ and let E_2 be the smallest equational theory over Σ_2 containing E_0. Then for all processes P and Q not containing $\wedge$ or $\vee$, we have that $P \approx_{E_1} Q$ if and only if $P \approx_{E_2} Q$.*

Proof. The proof has the same structure as the proof of Theorem 1. We first define a notion of normal form for T_{Σ_1} terms with respect to ϕ, requiring that for any term of the form $\wedge(M_1,M_2)$ (resp. $\vee(M_1,M_2)$) occurring therein, $E_{\mathsf{ZK}} \not\vdash M_1\phi = \mathsf{true}$ or $E_{\mathsf{ZK}} \not\vdash M_2\phi = \mathsf{true}$ (resp. $E_{\mathsf{ZK}} \not\vdash M_1\phi = \mathsf{true}$ and $E_{\mathsf{ZK}} \not\vdash M_2\phi = \mathsf{true}$). We then define a notion of validity for terms and plain processes, which requires that $\wedge$ and $\vee$ do not occur therein. The definition of validity for frames and extended processes is similar to Definition 13, where the new definition of normal form and validity for terms and plain processes is taken into account. Finally, the compilation from T_{Σ_1} to T_{Σ_2} is simply defined as the identity function. It is easy to see that Proposition 1, Proposition 2, and Lemma 1 still hold. Since the identity function is bijective, we have the double implication in Lemma 5 and Lemma 7, as desired. □

The previous proof shows that the framework proposed in this section provides a methodology for proving the soundness of any transformation of equational theories for which Proposition 1, Proposition 2, and Lemma 1 hold.

5 Case Study: Direct Anonymous Attestation

To exemplify the applicability of our theory to real-world protocols, we analyze the security properties of the Direct Anonymous Attestation (DAA) scheme [9]. DAA constitutes a cryptographic protocol that enables the remote authentication of a hardware module called the Trusted Platform Module (TPM), while preserving the privacy of the user owning the module. Such TPMs are now widely included in end-user hardware such as desktop PCs and notebooks.

The goal of the DAA protocol is to enable the TPM to sign arbitrary messages and to send them to an entity called the verifier in such a way that the verifier will only learn that a valid TPM signed that message, but without revealing the TPM's identity. The DAA protocol relies heavily on zero-knowledge proofs to achieve anonymity. The

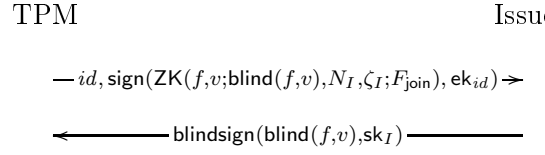
occurrence of these proofs in particular prevented a previous analysis of the protocol using abstract verification.

The DAA protocol is composed of two subprotocols: the *join protocol* and the *DAA-sign protocol*. The join protocol allows a TPM to obtain a certificate from an entity called the issuer. The protocol ensures that even the issuer cannot link the TPM to its subsequently produced signatures. The DAA-sign protocol enables a TPM to sign a message. This signed message is then verified by the verifier.

We assume that every TPM has a unique id as well as a secret signature key called the *endorsement key* (EK). The issuer is assumed to know the public keys corresponding to the secret EKs. We assume further a publicly known string bsn_I called the basename of the issuer, as well as a publicly known unique string bsn_V for each verifier V. Every TPM has a secret seed $daaseed_{id}$ that allows for deriving secret values $f_{cnt} := H(daaseed_{id}, cnt)$ where H is some hash function. We will call f_{cnt} the f-value for counter cnt. Each such f-value represents a virtual identity with respect to which the TPM can execute the join and the DAA-sign protocol.

5.1 Join protocol

In the join protocol, the TPM can receive a certificate for one of its f-values f from the issuer. Such a certificate is basically just a signature on f of the TPM. However, since we do not want the issuer to learn f, we have to use blind signatures, i.e., the request from the TPM to the issuer contains $\mathsf{blind}(f, v)$, for some random v, instead of just f. Furthermore, for reasons that will become clear in the description of rogue-tagging below, the TPM is required to also send the hash value $N_I := \mathsf{exp}(\zeta_I, f)$ along with its request where ζ_I is a value derived from the issuer's basename bsn_I. The function exp constitutes an exponentation in the original specification of DAA; we model it as a hash function with two arguments. Since we do not want the TPM to use different f-values in the computation of N_I and of $\mathsf{blind}(f, v)$, we have to attach a ZK proof that the same f-value has been used in both cases. After checking the proof, the issuer signs the blinded f-value $\mathsf{blind}(f, v)$ and returns this signature $x := \mathsf{blindsign}(\mathsf{blind}(f, v), sk_I)$. Then $cert := \mathsf{unblind}(x, v)$ is a valid blind signature on f. This certificate $cert$ will be used for the DAA-sign protocol. Since we want to guarantee that only valid TPMs can receive certificates, the TPM authenticates all its communication to the issuer using its endorsement key ek_{id}. The join protocol has the following overall shape:



with $F_{\mathsf{join}} := (\beta_1 = \mathsf{blind}(\alpha_1, \alpha_2) \wedge \beta_2 = \mathsf{exp}(\beta_3, \alpha_1))$. In our calculus, we can model the behavior of the TPM in the join protocol as follows:

$$
\begin{aligned}
\mathsf{tpmjoin} := \; & \mathit{let}\ f = \mathsf{hash}(\mathsf{pair}(\mathsf{daaseed}(id), cnt))\ \mathit{in} \\
& \nu v. \\
& \mathit{let}\ U = \mathsf{blind}(f, v)\ \mathit{in} \\
& \mathit{let}\ \zeta_I = \mathsf{hash}(\mathsf{pair}(n_1, bsn_I))\ \mathit{in} \\
& \mathit{let}\ N_I = \mathsf{exp}(\zeta_I, f)\ \mathit{in} \\
& \mathit{let}\ zkp = \mathsf{ZK}(f, v; U, N_I, \zeta_I; F_{\mathsf{join}})\ \mathit{in} \\
& \overline{pub}\langle \mathsf{pair}(id), \mathsf{sign}(zkp, \mathsf{sk}(\mathsf{ek}(id)))\rangle. \\
& pub(x). \\
& \mathit{let}\ cert = \mathsf{unblind}(x, v)\ \mathit{in} \\
& \mathit{if}\ \mathsf{blindver}(cert, f, \mathsf{pk}(\mathsf{issuerK})) = \mathsf{true}\ \mathit{then} \\
& \mathsf{event}\ \mathsf{JOINED}(id, cnt, cert). \\
& \overline{och}\langle cert\rangle
\end{aligned}
$$

Here we use $\mathit{let}\ x = M\ \mathit{in}\ P$ as syntactic sugar for $P\{M/x\}$. The occurence of an event M is modeled as $\overline{c}\langle M\rangle$ where c is a distinguished channel used only for events. Given the explanations above, most steps in this process should be self-explanatory, however, a few points merit further explanation: The secret seed $daaseed_{id}$ is modelled by the private constructor $\mathsf{daaseed}$ taking as input id. In the computation of $\zeta_I := \mathsf{hash}(\mathsf{pair}(n_1, bsn_I))$, n_1 is a free name. In the original DAA protocol [9], the integer 1 is used here. For communication with the issuer, we use the channel pub. The secret key ek_{id} and the public key sk_I are modeled as $\mathsf{sk}(\mathsf{ek}(id))$ and $\mathsf{pk}(\mathsf{issuerK})$ where ek and $\mathsf{issuerK}$ are private constructors. That is, by $\mathsf{ek}(id)$ we model a secret function mapping a TPM's identity to the endorsement secret/public key pair. We then use the operators sk and pk to access the secret and the public key. The function $\mathsf{issuerK}$ is nullary since, for the sake of simplicity, we model a single issuer. The private channel och will later be modeled as a secret channel to pass the received certificate to the DAA-sign process.

Accordingly, we model the issuer's part in the join protocol as follows:

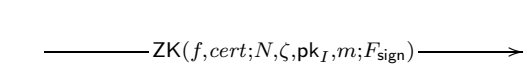
$$
\begin{aligned}
\mathsf{issuer} := \; & !\ pub(msg). \\
& \mathit{let}\ id = \mathsf{first}(msg)\ \mathit{in} \\
& \mathit{let}\ sig = \mathsf{snd}(msg)\ \mathit{in} \\
& \mathit{let}\ zkp = \mathsf{msg}(sig)\ \mathit{in} \\
& \mathit{if}\ \mathsf{ver}(\mathsf{sig}, zkp, \mathsf{pk}(\mathsf{ek}(id)) = \mathsf{true}\ \mathit{then} \\
& \mathit{if}\ \mathsf{Ver}_{2,3}(F_{\mathsf{join}}; zkp) = \mathsf{true}\ \mathit{then} \\
& \mathit{let}\ U = \mathsf{Public}_1(zkp)\ \mathit{in} \\
& \mathit{let}\ N = \mathsf{Public}_2(zkp)\ \mathit{in} \\
& \mathit{let}\ \zeta = \mathsf{Public}_3(zkp)\ \mathit{in} \\
& \mathit{if}\ rogue = \mathsf{true}\ \mathit{then}\ 0\ \mathit{else} \\
& \mathit{if}\ rogueid = \mathsf{true}\ \mathit{then}\ 0\ \mathit{else} \\
& \mathit{if}\ \zeta = \mathsf{hash}(\mathsf{pair}(n_1, bsn_I))\ \mathit{then} \\
& \mathit{let}\ cert = \mathsf{blindsign}(U, \mathsf{sk}(\mathsf{issuerK}))\ \mathit{in} \\
& \overline{pub}\langle cert\rangle
\end{aligned}
$$

In this process, *rogue* and *rogueid* represent predicates depending on N, ζ and id. These

are used for the detection of rogue TPMs. We will specify *rogue* and *rogueid* in more detail below when we discuss rogue detection.

5.2 DAA-sign protocol

After successfully executing the join protocol, the TPM has a valid certificate *cert* for its f-value f signed by the issuer. Since we only want valid TPMs to be able to DAA-sign a message m, the TPM will have to convince a verifier V that it possesses a valid certificate *cert*. Of course, the TPM cannot directly send *cert* to the verifier V, since this would reveal f. Instead, the TPM produces a zero-knowledge proof *zkp* that it knows a valid certificate. If the TPM, however, would just send (zkp, m) to the verifier, the protocol would be subject to a trivial message substitution attack. We instead combine m with the proof such that one can only replace m if one redoes the proof (and this again can only be done by knowing a valid certificate). Fortunately, this can easily be done in our formalism by including m in the public parameters of the zero-knowledge proof *zkp* (there is no condition that a parameter included in the proof actually has to be used by the formula). In this fashion we produce a kind of zero-knowledge signature that can only be forged if the attacker is able to produce a valid proof. Furthermore, we again include a value $N := \mathsf{exp}(\zeta, f)$ whose importance will become clear below. The overall shape of the DAA-sign protocol is hence as follows:



with

$$F_{\mathsf{sign}} \;:=\; \beta_1 = \mathsf{exp}(\beta_2, \alpha_1) \;\wedge\; \mathsf{blindver}(\alpha_2, \alpha_1, \beta_3).$$

An interesting point here is the choice of ζ. By prescribing different derivations of ζ, we get different modes of DAA-signing: an anonymous and a pseudonymous one. In case of anonymous DAA-signing, ζ is a fresh name chosen by the verifier. In this case, two signatures by the same TPM will contain values $N = \mathsf{exp}(\zeta, f)$ and $N' = \mathsf{exp}(\zeta', f)$ for different ζ, ζ', so the attacker will not be able to link these signatures. In the case of pseudonymous DAA-signatures, however, we derive ζ in a deterministic fashion from the basename bsn_V of the verifier. Then any two signatures for the same verifier using the same f-value will have the same value of N; hence these signatures can be linked. It will not be possible, however, to link these signatures to the execution of the join-protocol or to signatures for other verifiers. N takes the role of a verifier-specific pseudonym.

We now discuss how to write this protocol in our calculus. We start with the anony-

mous variant where ζ is a fresh name:

$$
\begin{aligned}
\mathsf{daasigna} := {} & \nu\zeta. \\
& \mathit{let}\ f = \mathsf{hash}(\mathsf{pair}(\mathsf{daaseed}(id), cnt))\ \mathit{in} \\
& \mathit{let}\ N = \mathsf{exp}(\zeta, f)\ \mathit{in} \\
& \mathit{let}\ zkp = \mathsf{ZK}(f, cert; N, \zeta, \mathsf{pk}(\mathsf{issuerK}), m; F_{\mathsf{sign}})\ \mathit{in} \\
& \mathsf{event}\ \mathsf{DAASIGNEDA}(id, cnt, m). \\
& \overline{pub}\langle zkp \rangle
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{daavera} := {} & pub(zkp). \\
& \mathit{if}\ \mathsf{Ver}_{2,4}(F_{\mathsf{sign}}; zkp) = \mathsf{true}\ \mathit{then} \\
& \mathit{let}\ N = \mathsf{Public}_1(zkp)\ \mathit{in} \\
& \mathit{let}\ \zeta = \mathsf{Public}_2(zkp)\ \mathit{in} \\
& \mathit{if}\ \mathsf{Public}_3(zkp) = \mathsf{pk}(\mathsf{issuerK})\ \mathit{then} \\
& \mathit{let}\ m = \mathsf{Public}_4(zkp)\ \mathit{in} \\
& \mathit{if}\ \mathit{rogue} = \mathsf{true}\ \mathit{then}\ 0\ \mathit{else} \\
& \mathsf{event}\ \mathsf{DAAVERIFIEDA}(m)
\end{aligned}
$$

As in the case of the issuer process, *rogue* is a predicate depending on ζ and N that we will elaborate upon further when we discuss rogue detection below.

The pseudonymous variants of these processes are similarly defined: The pseudonymous DAA-signing process daasignp is defined like daasigna, except that $\nu\zeta$ is replaced by $\mathit{let}\ \zeta = \mathsf{hash}(\mathsf{pair}(n_1, bsn_V))\ \mathit{in}$. The corresponding verification process daaverp is defined like daavera, except that after $\mathit{let}\ \zeta = \mathsf{Public}_2(zkp)\ \mathit{in}$ we insert $\mathit{if}\ \zeta = H(\mathsf{pair}(n_1, bsn_V))\ \mathit{then}$. Furthermore, to be able to formulate a more fine-grained authenticity property below, we output the more informative events $\mathsf{DAASIGNEDP}(id, cnt, bsn_V, m)$ and $\mathsf{DAAVERIFIEDP}(m, bsn_V, N)$ instead of $\mathsf{DAASIGNEDA}(id, cnt, m)$ and $\mathsf{DAAVERIFIEDA}(m)$, respectively. These changes yield the fol-

lowing two processes:

$$
\begin{aligned}
\mathsf{daasignp} := {} & \mathit{let}\ \zeta = \mathsf{hash}(\mathsf{pair}(n_V, bsn_V))\ \mathit{in} && (*)\\
& \mathit{let}\ f = \mathsf{hash}(\mathsf{pair}(\mathsf{daaseed}(id), cnt))\ \mathit{in}\\
& \mathit{let}\ N = \mathsf{exp}(\zeta, f)\ \mathit{in}\\
& \mathit{let}\ zkp = \mathsf{ZK}(f, cert; N, \zeta, \mathsf{pk}(\mathsf{issuerK}), m; F_{\mathsf{sign}})\ \mathit{in}\\
& \mathsf{event}\ \mathsf{DAASIGNEDP}(id, cnt, bsn_V, m).\\
& \overline{pub}\langle zkp\rangle
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{daaverp} := {} & pub(zkp).\\
& \mathit{if}\ \mathsf{Ver}_{2,4}(F_{\mathsf{sign}}; zkp) = \mathsf{true}\ \mathit{then}\\
& \mathit{let}\ N = \mathsf{Public}_1(zkp)\ \mathit{in}\\
& \mathit{let}\ \zeta = \mathsf{Public}_2(zkp)\ \mathit{in}\\
& \mathit{if}\ \zeta = \mathsf{hash}(\mathsf{pair}(n_V, bsn_V))\ \mathit{then} && (*)\\
& \mathit{if}\ \mathsf{Public}_3(zkp) = \mathsf{pk}(\mathsf{issuerK})\ \mathit{then}\\
& \mathit{let}\ m = \mathsf{Public}_4(zkp)\ \mathit{in}\\
& \mathit{if}\ \mathit{rogue} = \mathsf{true}\ \mathit{then}\ 0\ \mathit{else}\\
& \mathsf{event}\ \mathsf{DAAVERIFIEDP}(m, bsn_V, N)
\end{aligned}
$$

with $n_V := n_1$. The most important changes with respect to the anonymous DAA-sign protocol are marked with $(*)$. Note that we parametrized these processes with respect to the value $n_V := n_1$ used in the computation of ζ. This is to be able to express the changes needed for circumventing the attack described in [26], see below.

5.3 Rogue-tagging

So far, we presented the DAA protocol under the assumption that no TPM is compromised. A TPM is a single chip so that it is very difficult to extract private information from a TPM. Extracting such private information is however not impossible, so we have to expect that a few TPMs can get compromised. But as soon as a single TPM is compromised, the attacker can sign arbitrary messages, and these signatures even cannot be traced to this specific TPM. Even worse, the attacker could release the f-value and a corresponding certificate on the Internet; this would allow everyone to fake DAA-signatures. To capture this last case, a so-called rogue list is introduced that contains all f-values that have been published on the Internet. Furthermore, the issuer maintains a list of revoked TPM ids. Since the communication with the issuer is authenticated, the issuer can refuse to issue certificates to a revoked TPM. Already issued certificates stay valid. To address this problem – note that in every protocol execution (join or DAA-sign) based on some f-value f – the TPM sends a pair (ζ, N) with $N = \mathsf{exp}(\zeta, f)$. So given a list of rogue f-values $F := (f_1, \ldots, f_n)$, we can check whether $f \in F$ by checking whether $N = \mathsf{exp}(\zeta, f_i)$ for some $i \in [1, n]$. Thus the attacker cannot use a certificate relative to an f-value that has been marked rogue.

To model this mechanism in our calculus, we introduce two predicates *rogueid* and *rogue* in the issuer and verifier processes above. The predicate *rogueid* (used only be the issuer) is defined to evaluate to true iff the TPM id is marked rogue. So if, e.g., the ids id_1, id_2, id_3 are marked rogue, we would set $rogueid := (id = id_1 \vee id = id_2 \vee id = id_3)$. The predicate *rogue* checks whether $N = \mathsf{exp}(\zeta, f')$ for some f' on the rogue list, so if, e.g., the f-values f_1, f_2, f_3 were rogue-listed, we would define $rogue := (N = \mathsf{exp}(\zeta, f_1) \vee N = \mathsf{exp}(\zeta, f_2) \vee N = \mathsf{exp}(\zeta, f_3)$.

5.4 Security properties of DAA

We will now discuss the main security properties of DAA and how to model them in our calculus.

5.4.1 Authenticity

The first property we would like to model is authenticity: If the verifier accepts a message m, then *some* TPM has DAA-signed this message m. To model this, we consider the following process:

$$\mathsf{issuer}|\overline{pub}\langle\mathsf{pk}(\mathsf{issuerK})\rangle|!pub(id).\mathsf{TPMs}|!\mathsf{daavera}|!\mathsf{daaverp}$$

The output $\overline{pub}\langle\mathsf{pk}(\mathsf{issuerK})\rangle$ reflects that the public key $\mathsf{pk}(\mathsf{issuerK})$ is publicly known. if we omitted this output, the adversary could not generate this term, since $\mathsf{issuerK}$ is a private name (otherwise the adversary would know $\mathsf{sk}(\mathsf{issuerK})$). The subprocess TPMs reflects that we require authenticity to hold even if the adversary controls an arbitrary number of TPMs in an arbitrary fashion (except for learning their secrets). We model this process as follows:

$$\begin{aligned}\mathsf{TPMs} :=& !pub(cnt).\nu och.(\mathsf{tpmjoin}|\\ &(och(cert).!pub(m).(\mathsf{daasigna}|pub(bsn_V).\mathsf{daasignp}))).\end{aligned}$$

Thus for any pair of id, cnt received from the adversary, this process performs a join, and with the certificate *cert* received from the issuer, it DAA-signs any message m anonymously or pseudonymously with respect to arbitrary basenames bsn_V. Note how we used inputs to bind the free variables id, cnt, och, m, bsn_V in $\mathsf{tpmjoin}$, $\mathsf{daasigna}$, and $\mathsf{daasignp}$.

Given this process, authenticity is defined as the fulfillment of the following two trace properties:

$$\begin{aligned}&\mathsf{DAAVERIFIEDP}(m, bsn, N) \Rightarrow \mathsf{DAASIGNEDP}(id, cnt, bsn, m)\\ &\mathsf{DAAVERIFIEDA}(m) \Rightarrow \big(\mathsf{DAASIGNEDA}(id, cnt, m) \vee\\ &\qquad\qquad \mathsf{DAASIGNEDP}(id, cnt, bsn, m)\big).\end{aligned}$$

Intuitively, the first property means that if an event $\mathsf{DAAVERIFIEDP}(m, bsn, N)$ occurs, then also $\mathsf{DAASIGNEDP}(id, cnt, bsn, m)$ occurs in that trace *with the same values of bsn and*

m, i.e., when a verifier accepts a pseudonymously signed message m, then a valid TPM actually sent that message m for that verifier. Similarly, the second property guarantees that if a verifier accepts a message as anonymously signed, that message has been signed anonymously or pseudonymously by some valid TPM. (An inspection of the protocol reveals that we cannot expect pseudonymously signed messages not to be accepted by anonymous verification.) We refer to Definition 8 for a formal definition of these trace properties.

Trace properties such as the above authenticity properties can be verified with the mechanized prover ProVerif [6]. We applied the compilation described in Section 4 and feed the output – now a process in a finitely generated equational theory – to ProVerif. ProVerif successfully verifies the authenticity properties. The running time of this proof is 3 seconds on a Pentium 4, 3 GHz. A more detailed description of the necessary steps is given in Section 6.2. The tool implementing the compiler from Section 4.1 can be found at [4]. So far, we have not investigated the case that some TPMs are rogue-listed (i.e., $\mathit{rogue} = \mathit{rogueid} = \mathsf{false}$). An analysis of this case can be found in Section 6.2.3.

5.4.2 Anonymity

The second property we would like to examine is the anonymity of the anonymous DAA-sign operation. In other words, if two TPMs T_1, T_2 might have signed a given message, the attacker should not be able to distinguish which TPM has signed the message. Obviously, this can be formalized as observational equivalence between two processes P_1, P_2, where in P_i the TPM T_i signed the concerned message. E.g., a natural formulation would be to define P_1 and P_2 as follows:

$$\begin{aligned} P_i := \mathsf{leak} \mid & \\ & (\mathit{let}\ (\mathit{id}, \mathit{cnt}, \mathit{och}) = (\mathit{id}_1, n_1, \mathit{int}_1)\ \mathit{in}\ \mathsf{tpmjoin}) \mid \\ & (\mathit{let}\ (\mathit{id}, \mathit{cnt}, \mathit{och}) = (\mathit{id}_2, n_1, \mathit{int}_1)\ \mathit{in}\ \mathsf{tpmjoin}) \mid \\ & (\mathit{int}_1(\mathit{cert}_1).\mathit{int}_2(\mathit{cert}_2). \\ & \quad \mathit{let}\ (\mathit{id}, \mathit{cnt}, \mathit{cert}) = (\mathit{id}_i, n_1, \mathit{cert}_i)\ \mathit{in}\ \mathsf{daasigna}) \end{aligned}$$

with $\mathsf{leak} := \left(!\mathit{pub}(\mathit{id}).\overline{\mathit{pub}}\langle\mathsf{pk}(\mathsf{ek}(\mathit{id}))\rangle\right) \mid \overline{\mathit{pub}}\langle\mathsf{pk}(\mathsf{issuerK})\rangle \mid \overline{\mathit{pub}}\langle\mathsf{sk}(\mathsf{issuerK})\rangle$, where $\mathit{id}_1, \mathit{id}_2, n_1$ are free names and $\mathit{int}_1, \mathit{int}_2$ are private channels for transmitting the certificate from the tpmjoin process to the daasigna process. The leak process leaks all public information and all secrets of the issuer. This models the case that the issuer is corrupted, thus making the security property stronger since anonymity holds even when the issuer colludes with the attacker. The two invocations of tpmjoin request certificates for different ids id_1 and id_2. These certificates are then assigned to the variables cert_1 and cert_2. Then a message m (m is a free name in daasigna) is signed with respect to either id_1 and cert_1 or id_2 and cert_2, depending on whether we consider the process P_1 or P_2. Anonymity is then defined as the statement that P_1 and P_2 are observationally equivalent.

Although we can successfully prove this fact using our compiler and ProVerif, closer inspection reveals that this property is not very general. For example, it does not cover

Table 8 The processes P_1 and P_2 in the definition of anonymity. The numbers in square brackets refer to the steps in the description of the security property.

[1] $P_i := \mathsf{leak} \mid (pub(x).\ let\ id = \mathsf{corrupt}(x)\ in\ \overline{pub}\langle \mathsf{daaseed}(id)\rangle.\overline{pub}\langle \mathsf{sk}(\mathsf{ek}(id))\rangle) \mid$ (1)

[3] $(let\ (id, cnt, och) = (id_1, cnt_1, int_1)\ in\ \mathsf{tpmjoin}) \mid$ (2)

[3] $(let\ (id, cnt, och) = (id_2, cnt_2, int_2)\ in\ \mathsf{tpmjoin}) \mid$ (3)

[4] $(let\ id = id_1\ in\ \mathsf{TPMs}) \mid (let\ id = id_2\ in\ \mathsf{TPMs}) \mid$ (4)

[3] $\big(int_1(cert_1).int_2(cert_2).$ (5)

[5] $\big((!pub(m).\ let\ (id, cnt, cert) = (id_i, cnt_i, cert_i)\ in\ \mathsf{daasigna}) \mid$ (6)

[4] $(!pub(m).\ let\ (id, cnt, cert) = (id_1, cnt_1, cert_1)\ in\ \mathsf{daasigna}) \mid$ (7)

[4] $(!pub(m).\ let\ (id, cnt, cert) = (id_2, cnt_2, cert_2)\ in\ \mathsf{daasigna}) \mid$ (8)

[4] $(!pub(m).\ pub(bsn_V).\ let\ (id, cnt, cert) = (id_1, cnt_1, cert_1)\ in\ \mathsf{daasignp}) \mid$ (9)

[4] $(!pub(m).\ pub(bsn_V).\ let\ (id, cnt, cert) = (id_2, cnt_2, cert_2)\ in\ \mathsf{daasignp})\big)$ (10)

the case that the TPM T_1 first signs a few messages, and then either T_1 or T_2 sends another message (so that the adversary can try to link messages). Further it does not take into account that the adversary might influence (i.e., choose) the messages to be signed, or that the T_i signs several messages, or that additionally pseudonymous signatures are produced. To capture all these cases, we need a much more complex security definition which is captured by the following game:

1. The issuer and an arbitrary number of TPMs are corrupted (i.e., their secrets leak).
2. Two challenge TPM ids id_1, id_2 are chosen. Two cnt-value cnt_1, cnt_2 are chosen.
3. The TPMs id_1, id_2 join with respect to cnt-value cnt_1, cnt_2, respectively.
4. The adversary may ask both challenge TPMs to execute the join protocol and to sign messages chosen by the adversary anonymously or pseudonymously with respect to either the certificates obtained in Step 3 or the certificates obtained in this step. This may happen arbitrarily often.
5. The adversary may ask the challenge TPM id_i to sign a message chosen by the adversary with respect to the certificate $cert_i$. Here $i \in \{1, 2\}$ depending on whether we are running the process P_1 or P_2 (and the adversary has to distinguish whether $i = 1$ or $i = 2$). This may happen arbitrarily often.

We model this by the processes P_1, P_2 given in Table 8. These processes constitute a formalization of the game depicted above. Note that although the adversary's possibilities in lines (7–10) seem to be subsumed by the invocations of the subprocess TPMs in line (4), there is a slight difference: The process TPMs does not allow the attacker to sign messages *with the certificates obtained in Step 3.* The constructor $\mathsf{corrupt}$ in (1) is used to generate

Table 9 The processes P_1 and P_2 in the definition of pseudonymity. Compare also with Table 8.

$$
\begin{aligned}
P_i := {} & \mathsf{leak} \mid (pub(x).\ let\ id = \mathsf{corrupt}(x)\ in\ \overline{pub}\langle \mathsf{daaseed}(id)\rangle.\overline{pub}\langle \mathsf{sk}(\mathsf{ek}(id))\rangle) \mid && (11)\\
& (let\ (id, cnt, och) = (id_1, cnt_1, int_1)\ in\ \mathsf{tpmjoin}) \mid && (12)\\
& (let\ (id, cnt, och) = (id_2, cnt_2, int_2)\ in\ \mathsf{tpmjoin}) \mid && (13)\\
& (let\ id = id_1\ in\ \widetilde{\mathsf{TPMs}}) \mid (let\ id = id_2\ in\ \widetilde{\mathsf{TPMs}}) \mid && (14)\\
& \big(int_1(cert_1).int_2(cert_2). && (15)\\
& \ \big((!pub(m).\ let\ (id, cnt, cert) = (id_i, cnt_i, cert_i)\ in\ \mathsf{daasigna}) \mid && (16)\\
& \ \big((!pub(m).pub(x).\ let\ (id, cnt, cert, bsn_V) = \\
& \qquad\qquad (id_i, cnt_i, cert_i, \mathsf{bsnVch}(x))\ in\ \mathsf{daasignp}) \mid && (17)\\
& \quad (!pub(m).\ let\ (id, cnt, cert) = (id_1, cnt_1, cert_1)\ in\ \mathsf{daasigna}) \mid && (18)\\
& \quad (!pub(m).\ let\ (id, cnt, cert) = (id_2, cnt_2, cert_2)\ in\ \mathsf{daasigna}) \mid && (19)\\
& \quad (!pub(m).pub(x).\ let\ (id, cnt, cert, bsn_V) = \\
& \qquad\qquad (id_1, cnt_1, cert_1, \mathsf{bsnVoth}(x))\ in\ \mathsf{daasignp}) \mid && (20)\\
& \quad (!pub(m).pub(x).\ let\ (id, cnt, cert, bsn_V) = \\
& \qquad\qquad (id_2, cnt_2, cert_2, \mathsf{bsnVoth}(x))\ in\ \mathsf{daasignp})\big) && (21)
\end{aligned}
$$

with

$$
\begin{aligned}
\mathsf{TPMs} := {} & !pub(cnt).\nu och.(\mathsf{tpmjoin} | (och(cert).!pub(m).\\
& \qquad (\mathsf{daasigna} | pub(x).let\ bsn_V = \mathsf{bsnVoth}(x)\ in\ \mathsf{daasignp}))).
\end{aligned}
$$

an infinite supply of ids of corrupted TPMs.

The property of anonymity is then formalized as the statement that P_1 and P_2 are observationally equivalent, which is a statement accessible to ProVerif. When directly applying ProVerif to the output of the compiler described in Section 4.1, however, ProVerif does not terminate. Instead, we additionally have to rewrite the resulting theory using the technique given by Theorem 4 and Lemma 8. After this additional step, ProVerif successfully verifies that P_1 and P_2 are observationally equivalent. The running time is 149 seconds on a Pentium 4, 3 GHz. More details can be found in Section 6.2.3. Note that in the case of anonymity, we do not need to consider the case of rogue-listing, since neither the issuer nor the verifier appear in corrupted form.

5.4.3 Pseudonymity

Modeling the pseudonymity requirements is similar to anonymity; however, there are a few additional subtleties to be considered. A naive approach of modeling the security

of the pseudonymous signatures would be to take the process described in Table 8, but replace daasigna by daasignp in line (6) and let the adversary choose the value of bsn_V in that line. This denotes the fact that the adversary can now ask the challenge TPM id_i to perform a pseudonymous signature. The resulting security property, however, cannot be expected to hold since the adversary could request a pseudonymous DAA-signature from the challenge TPM id_i via line (6) and a pseudonymous from the TPM id_1 via line (9). Then the adversary could compare whether both signatures carry the same pseudonym N, if so we have $i = 1$, otherwise we have $i = 2$. Instead, we must require that the signatures produced in lines (9,10) to use a different basenames than those in line (6). We do this using two different sets of basenames. The basenames allowed for requesting a signature from the challenge TPM id_i are of the form $\mathsf{bsnVch}(x)$ and those allowed for all other DAA-sign requests are of the form $\mathsf{bsnVoth}(x)$ where bsnVch and bsnVoth are constructors of arity 1. The resulting processes P_1, P_2 are depicted in Table 9. Note that we allow the adversary to request both anonymous and pseudonymous DAA-signatures from the challenge TPM id_i.

Using our compiler and ProVerif, we can show that P_1 and P_2 are observationally equivalent.[1] The verification of this fact takes 56 seconds on a Pentium 4, 3 GHz.

Our modelling in Table 9 implicitly assumes that it is guaranteed that the basename bsn_I of the issuer does not equal any of the basenames of the verifiers. (The basename bsn_I of the issuer is modelled as a free name.) It is known that if the basename of the issuer may coincide with one of the verifiers basenames there is an attack on the pseudonymity of the system [26]: If $bsn_I = bsn_V$, the values ζ computed by the tpmjoin and the daasignp processes are equal. If further both processes use the same f-value f, the resulting pseudonym $N = \exp(\zeta, f)$ will also be equal. This allows to link signatures and joins. To model this, we model bsn_I as a term of the form $\mathsf{bsnVch}(x)$. More exactly, we set

$$\tilde{P}_i := \mathit{let}\ bsn_I := \mathsf{bsnVch}(n_0)\ \mathit{in}\ P_i$$

and ask whether $\tilde{P}_1$ and $\tilde{P}_2$ are observationally equivalent. As expected, the combination of our compiler and ProVerif successfully detects the attack and outputs that $\tilde{P}_1$ and $\tilde{P}_2$ are not observationally equivalent. The verification takes 40 seconds on a Pentium 4, 3 GHz.

In [26] it is proposed to fix the protocol by using different integers in the computation of ζ in the processes tpmjoin and daasignp. We do this by defining $n_V := n_0$ instead of $n_V := n_1$ in the definition of daasignp. Using this change, our compiler together with ProVerif successfully determines that $\tilde{P}_1$ and $\tilde{P}_2$ are observationally equivalent. The verification takes 62 seconds on a Pentium 4, 3 GHz.

6 Mechanized Security Proofs for DAA

In this section, we will examine the practical applicability of the results of the previous sections to mechanized security proofs. Instead of designing a new tool from scratch,

[1] As with the proof of anonymity, we have to apply Theorem 4 and Lemma 8 to ensure termination.

we implemented a compiler that generates input for the automated prover ProVerif [6] according to the description given in Section 4.1. This compiler together with example inputs can be found at [4]. To show how our theory is applied, we analyze two protocols, namely the simple example protocol from Section 3.3 and the DAA protocol [9]. We will describe how to prove different security properties of these protocols and also what pitfalls occurred in our investigation and how to avoid these.

6.1 Example Protocol

We first examine the example protocol from 3.3. Many of the techniques described here will also be used in the more complex example of DAA below. We model the example protocol as follows (omitting the specification of the base theory here):[2]

```
free pub,A,B.
private free priv,s1,s2,s3.

define zkproof =
  land(
    or(or(eq(alpha2,beta1),eq(alpha2,beta2)),eq(alpha2,beta3)),
    sigver(alpha1,beta4,alpha2)).

let server = event GAVEAUTHFOR(s,A,B); out(priv,sign(pair(A,B),sk(s))).
let B = in(priv,sig); if sigver(sig,pair(A,B),pk(s))=true then
        out(pub,zk(sig,pk(s);pk(s1),pk(s2),pk(s3),pair(A,B);zkproof)).
let A = in(pub,zkp); if zkver(2;4;zkproof;zkp)=true then
        if public1(zkp)=pk(s1) then
        if public2(zkp)=pk(s2) then
        if public3(zkp)=pk(s3) then
        if fst(public4(zkp))=A then
        event GOTAUTHFOR(snd(public4(zkp))).
let leakpublic = out(pub,pk(s1)) | out(pub,pk(s2)) | out(pub,pk(s3)).
```

The syntax of this protocol should be mostly self-explanatory. It is the syntax of ProVerif with a few additions particular to our tool. The `define` statement defines an abbreviation `zkproof` for the formula we use in all ZK proofs and verifications (we use `land` instead of `and` since `and` is a reserved keyword in proVerif). The process `server` produces a signature on $\mathsf{pair}(A, B)$ using the secret key $\mathsf{sk}(\mathtt{s})$ and sends it to B over a secret channel. All server processes S_i are modelled using this single process `server` by instantiating `s` with different identities.

The process `B` then waits for a message from a server, checks whether this message constitutes a valid signature of $\mathsf{pair}(A, B)$ and then sends a ZK proof to A that it knows a signature `sig` that is valid with respect to one of the keys $\mathsf{pk}(\mathtt{s1}), \mathsf{pk}(\mathtt{s2}), \mathsf{pk}(\mathtt{s3})$ (without revealing which one). Note the syntax of the ZK constructor: It takes arguments

[2] [4], file `simple.pvi`.

$(\alpha_1, \ldots, \alpha_i; \beta_1, \ldots, \beta_j; F)$; the placement of the semicolons indicate which arguments are private (α_μ), which are public (β_μ) and which is the formula to be proven (F).

The process `A` waits for the proof sent by `B` and assigns it to the variable `zkp`. It first verifies whether `zkp` is actually a valid proof of the correct arity (`2;4`) for the right formula `zkproof`. Further it verifies that the public keys given in the ZK proof are the right ones and that the message m of which `B` claims to know a signature is indeed a pair having `A` as its first component. If so, `A` claims to have received authorization to communicate with the process whose identity is given in the second component of m.

Finally, we need to model a fourth process. This is due to the fact that we had to declare the server ids `s1, s2, s3` as private free names since otherwise the adversary would know the secret keys `sk(s1), sk(s2), sk(s3)`. Since the adversary should however know `pk(s1), pk(s2), pk(s3)`, we define a process `leakpublic` that outputs these values on a public channel.

So far these processes stand by themselves and are not executed in a common context. How these processes are actually executed depends on the property we want to prove.

We will now model the first security property. We require that A will not accept to communicate with B unless some server has signed an authorization. Or in the parlance of the events defined in the protocol description above, we want that if the event `GOTAUTHFOR`(*sender*) occurs, then the event `GAVEAUTHFOR`(*server*, *recipient*, *sender*) occurred earlier with the same value of *sender*. This is modelled by the following code fragment:[3]

```
compiler ZK.
passthrough query ev:GOTAUTHFOR(sender)
               ==> ev:GAVEAUTHFOR(server,A,sender).

process
        leakpublic |
        (let s=s1 in server) |
        (let s=s2 in server) |
        (let s=s3 in server) |
        (let s=s1 in B) |
        A
```

Here we see how we instantiate the value `s` to different server ids, so that we can use the single definition of `server` for all occurrences of the server: We runs several instances of `server`, and in each of them we substitute `s` with a different server id using the `let` statement. Similarly, we instantiate the process `B` so that it expects a message from server `s1`. The first line of the code fragment indicates which property we would like to prove. The keyword `passthrough` simply indicates that this command should be passed through directly to ProVerif and not be parsed by our compiler.

Finally, we have to tell our compiler what to do with our code. This is done by the statement `compiler ZK` which instructs our tool to implement the compiler as described

[3] [4], file `simple-auth.pvz`.

in Section 4.1.

If we compile and execute this code (see the `README` file in [4] for instructions) ProVerif successfully determines that the required property is indeed fulfilled (the running time is less than one second on a Pentium 4, 3 GHz). This property intuitively depends both on the soundness of the ZK proof (i.e., we cannot prove a wrong statement) and on the unforgeability of the signatures.

We will now investigate a more complex property: We require that given the public communication between `A` and `B`, we cannot determine which server authorized the communication. In other words, we want observational equivalence between a process where server `s1` authorizes `B` and a process where server `s2` authorizes `B`. This can be modelled as follows:[4]

```
process
        leakpublic |
        (let s=choice[s1,s2] in server) |
        (let s=choice[s1,s2] in B) |
        A
```

In the language of ProVerif, the `choice` operator is used to check for observational equivalence. The code given here specifies two processes P_1, P_2, where P_i results from replacing every occurrence of $\mathtt{choice}[\mathtt{t}_1, \mathtt{t}_2]$ by t_i, and ProVerif tries to prove that P_1 and P_2 are observationally equivalent. In the present case, ProVerif tries to prove observational equivalence between processes P_1 and P_2 where P_1 is an execution of our example protocol where `B` gets its authorization from server `s1`, and P_2 is an execution where `B` gets its authorization from server `s2`. Unfortunately, however, on the input described above, ProVerif does not seem to terminate. Experiments show that we need to get rid of the constructors `land` and `or` to allow for termination. Unfortunately, we cannot just remove them from our equational theory, since our protocol actually uses them (in `zkproof`). Even after applying our compiler, which removes all occurrences of the formula $F := \mathtt{zkproof}$ from the process itself, `land` and `or` are still contained in the equational theory generated by the compiler since this theory contains the following rule:

$$\begin{aligned}\mathsf{ZK}^F_{2,4}(x_1,x_2,y_1,y_2,y_3,y_4) = \mathsf{PZK}^F_{2,4}(x_1,x_2,y_1,y_2,y_3,y_4,\\ \mathtt{land}(\mathtt{or}(\mathtt{or}(\mathtt{eq}(x_2,y_1),\mathtt{eq}(x_2,y_2)),\mathtt{eq}(x_2,y_3)),\mathtt{sigver}(x_1,y_4,x_2))).\end{aligned} \quad (22)$$

Theorem 4 allows us to remove this rule. It is easy to see that in our equational theory Theorem 4 applies with $n_{2,4,F} = 3$ and

$$\begin{aligned}\tilde{\tau}^1_{2,4,F} &= (\mathtt{sign}(x,\mathtt{sk}(y)),\mathtt{pk}(y),\mathtt{pk}(y),p_2,p_3,x),\\ \tilde{\tau}^2_{2,4,F} &= (\mathtt{sign}(x,\mathtt{sk}(y)),\mathtt{pk}(y),p_1,\mathtt{pk}(y),p_3,x),\\ \tilde{\tau}^3_{2,4,F} &= (\mathtt{sign}(x,\mathtt{sk}(y)),\mathtt{pk}(y),p_1,p_2,\mathtt{pk}(y),x).\end{aligned}$$

[4] [4], file `simple-obseq-nonterm.pvz`.

Application of Lemma 8 then removes the rule (22). Instead, the rules

$$\mathsf{Ver}^F_{2,4}(\mathsf{ZK}^F_{2,4}(\mathtt{sign}(x,\mathtt{sk}(y)),\mathtt{pk}(y),\mathtt{pk}(y),p_2,p_3,x) = \mathtt{true},$$
$$\mathsf{Ver}^F_{2,4}(\mathsf{ZK}^F_{2,4}(\mathtt{sign}(x,\mathtt{sk}(y)),\mathtt{pk}(y),p_1,\mathtt{pk}(y),p_3,x) = \mathtt{true},$$
$$\mathsf{Ver}^F_{2,4}(\mathsf{ZK}^F_{2,4}(\mathtt{sign}(x,\mathtt{sk}(y)),\mathtt{pk}(y),p_1,p_2,\mathtt{pk}(y),x) = \mathtt{true}$$

are introduced (besides the obvious rules concerning the $\mathtt{public}_p$ constructor). Since now neither the process nor the equational theory contains `land` or `or`, by Lemma 8 we can remove the corresponding equational rules.

These additional transformations can also be performed using our tool. For this, we have to add the following additional commands to the input file:[5]

```
compiler AlternativeZKVer(
        zkver(2;4;zkproof;zk(sign(x,sk(y)),pk(y);pk(y),p2,p3,x;zkproof)),
        zkver(2;4;zkproof;zk(sign(x,sk(y)),pk(y);p1,pk(y),p3,x;zkproof)),
        zkver(2;4;zkproof;zk(sign(x,sk(y)),pk(y);p1,p2,pk(y),x;zkproof))).

compiler RemoveEquations(or).
compiler RemoveEquations(land).
```

(after the `compiler ZK` command). The first command corresponds to an application of Theorem 4. The tuples $t_1, \ldots, t_n$ are implicitly given by supplying terms of the form $\mathsf{Ver}_{i,j}(F, \mathsf{ZK}_{i,j}(t_i, F))$. Finally, `compiler RemoveEquations(`$c$`)` for a constructor c removes all equations of the form $c(\ldots) = \ldots$

Using the resulting modified but equivalent (see Theorem 4 and Lemma 8) equational theory, ProVerif terminates and successfully proves observational equivalence, i.e., the adversary cannot distinguish which server authorizes B to communicate with A. The verification takes 16 seconds on a Pentium 4, 3 GHz.

6.2 Direct Anonymous Attestation

We will now describe the mechanized analysis of the DAA protocol [9] described in Section 5.

6.2.1 Join

First, we describe the basic definition of the various components of the DAA protocol (join and DAA-sign). The join protocol is described by the following two processes tpmjoin and issuer.[6]

```
define joinproof = land(eq(beta1,blind(alpha1,alpha2)),
                        eq(beta2,exp(beta3,alpha1))).
```

[5] [4], file `simple-obseq.pvz`.
[6] [4], file `daa.pvi`.

```
let tpmjoin =
  let f = hash(pair(daaseed(id),cnt)) in
  new v;
  let U = blind(f,v) in
  let zetaI = hash(pair(n1,bsnI)) in
  let NI = exp(zetaI,f) in
  let zkp = zk(f,v;U,NI,zetaI;joinproof) in
  out(comm,pair(id,sign(zkp,sk(ek(id))))));
  in(comm,A);
  let cert = unblind(A,v) in
  if blindver(cert,f,pk(issuerK))=true then
  event JOINED(id,cnt,cert);
  out(och,cert).

let issuer =
  ! in(comm,msg);
  let id = fst(msg) in
  let sig = snd(msg) in
  let zkp = message(sig) in
  if sigver(sig,zkp,pk(ek(id)))=true then
  if zkver(2;3;joinproof;zkp)=true then
  let U = public1(zkp) in
  let N = public2(zkp) in
  let zeta = public3(zkp) in
  if rogue=true then event ROGUEI(id) else
  if rogueid=true then event ROGUEID(id) else
  if zeta=hash(pair(n1,bsnI)) then
  let cert = blindsign(U,sk(issuerK)) in
  event CERTIFIED(id,N);
  out(comm,cert).
```

These processes are formalizations of the corresponding processes in Section 5.1. We added the additional events `JOINED` and `CERTIFIED` to have to possibility of formulating additional properties. In contrast to Section 5.1, the communication channel is represented by the variable `comm`, which we then instantiate with the public channel `pub` or a private channel, depending on the property we model. The predicates `rogue` and `rogueid` can be defined using `define rogue = ...` and `define rogue = ...` depending on the situation. In most cases we will set `rogue` = `rogueid` = `false` to model that no rogue checking occurs. The term `joinproof` corresponds to F_{join} in Section 5.1.

For the analysis of protocols with rogue TPMs we will need an additional process. The process `issuer` will never issue a certificate to a TPM that is detected to be rogue, but we might want to model the case that some TPMs have already received a certificate before they were marked rogue. In order to be able to model this situation, we introduce

the following process `rogueissuer` that issues a certificate for a given f-value.[7]

```
let rogueissuer =
  ! in(pub,v); out(pub,unblind(blindsign(blind(f,v),sk(issuerK)),v)).
```

The variable `f` will be assigned the correct value in our security properties using a `let` directive. The nonce `v` used for blinding the signature is chosen by the adversary to model that a rogue TPM is assumed to be completely under the control of the adversary.

Finally, we also need to model the fact that the issuer is corrupted. This is achieved by giving all the issuer's knowledge to the adversary:[8]

```
let leakissuer =
  (!in(pub,id); out(pub,pk(ek(id)))) |
  out(pub,pk(issuerK)) | out(pub,sk(issuerK)).
```

Similarly, we model that a given TPM is corrupted:[9]

```
let leaktpm =
  out(pub,daaseed(id)) | out(pub,sk(ek(id))) | out(pub,pk(ek(id))).
```

Finally, besides leaking private information of corrupted principals, the adversary should get all public information:

```
let leakpublic = out(pub,pk(issuerK)) | !in(pub,id); out(pub,pk(ek(id))).
```

6.2.2 DAA-Sign

We will now describe the modelling of the second part of the DAA protocol, namely the DAA-sign protocol. The processes for performing an anonymous DAA-sign are defined as follows:[10]

```
define signproof = land(eq(beta1,exp(beta2,alpha1)),
                        blindver(alpha2,alpha1,beta3)).

let daasigna =
  new zeta;
  let f = hash(pair(daaseed(id),cnt)) in
  let N = exp(zeta,f) in
  let zkproof = zk(f,cert;N,zeta,pk(issuerK),m;signproof) in
  event DAASIGNEDA(id,cnt,m);
  out(comm,zkproof).

let daavera =
```

[7] [4], file `daa.pvi`.
[8] [4], file `daa.pvi`.
[9] [4], file `daa.pvi`.
[10] [4], file `daa.pvi`.

```
  in(comm,zkproof);
  if zkver(2;4;signproof;zkproof)=true then
  let N = public1(zkproof) in
  let zeta = public2(zkproof) in
  if public3(zkproof)=pk(issuerK) then
  let m = public4(zkproof) in
  if rogue=true then event ROGUEAV(m) else
  event DAAVERIFIEDA(m).
```

These are again direct encodings of the corresponding processes presented in Section 5.2, except that we have added a few more events and use `comm` for communication. The term `signproof` corresponds to F_{sign} in Section 5.2.

Similarly, we define the pseudonymous DAA-sign protocol:

```
define numberZetaV = n1.

let daasignp =
  let zeta = hash(pair(numberZetaV,bsnV)) in
  let f = hash(pair(daaseed(id),cnt)) in
  let N = exp(zeta,f) in
  let zkproof = zk(f,cert;N,zeta,pk(issuerK),m;signproof) in
  event DAASIGNEDP(id,cnt,bsnV,m);
  out(comm,zkproof).

let daaverp =
  in(comm,zkproof);
  if zkver(2;4;signproof;zkproof)=true then
  let N = public1(zkproof) in
  let zeta = public2(zkproof) in
  if zeta=hash(pair(numberZetaV,bsnV)) then
  if public3(zkproof)=pk(issuerK) then
  let m = public4(zkproof) in
  if rogue=true then event ROGUEPV(m,bsnV,N) else
  event DAAVERIFIEDP(m,bsnV,N).
```

This formalizes the corresponding processes from Section 5.2 with the addition of events and the change of the communication channel.

For convenience, we further implement the following processes:

```
let daaverifier = (! daavera) | (! daaverp).
```

This process represents a verification server that waits for anonymous and pseudonymous signatures, checks them, and outputs the corresponding events.

```
let tpmcontrolled =
  let comm=pub in ! in(pub,cnt); new och; (tpmjoin | (in(och,cert);
```

```
    ! in(pub,m); (daasigna | in(pub,bsnV); daasignp))).

let tpmcontrolledall = ! in(pub,id); tpmcontrolled.
```

The process `tpmcontrolled` corresponds to the process TPMs introduced on page 31 and models a TPM that is under the control of the adversary without revealing any of its secrets, i.e., the TPM joins and signs at the adversary's discretion. The process `tpmcontrolledall` represents the fact that *all* TPMs are under the control of the adversary.

6.2.3 Security Properties

The authenticity property from Section 5.4.1 is encoded as follows:[11]

```
passthrough query ev:DAAVERIFIEDP(xm,xbsn,xN)
                  ==> ev:DAASIGNEDP(xid,xcnt,xbsn,xm).
passthrough query ev:DAAVERIFIEDA(xm)
   ==> (ev:DAASIGNEDA(xid,xcnt,xm) | ev:DAASIGNEDP(xid,xcnt,xbsn,xm)).

process
  (let comm=pub in issuer) | leakpublic |
  tpmcontrolledall |
  (let comm=pub in daaverifier).
```

The `passthrough` directive is only necessary since our compiler is not able to parse the full syntax of `query`. It does not have any semantic meaning. Applying the compiler and ProVerif, the result is that the queried properties hold. The running time is 3 seconds on a Pentium 4, 3 GHz.

To implement this security property in the presence of rogue TPMs, we have to choose some rogue f-value and rogue TPM ids and define the predicates `rogue` and `rogueid` to returns `true` if a rogue f-value or id is being used. We choose to implement a test with a fixed number of rogue ids `rogueid1`,`rogueid2`,`rogueid3` and three fixed f-values `rogueF1`,`rogueF2`,`rogueF3`. These are defined using the following code:[12]

```
free rogueid1,rogueid2,rogueid3.

define rogueF1 = hash(pair(daaseed(rogueid1),n1)).
define rogueF2 = hash(pair(daaseed(rogueid2),n1)).
define rogueF3 = hash(pair(daaseed(rogueid2),n2)).

fun roguetest/5.
equation roguetest(zeta,exp(zeta,x),x,y,z) = true.
equation roguetest(zeta,exp(zeta,y),x,y,z) = true.
```

[11] [4], file `daa-verify-tpmcontrolled.pvz`.
[12] [4], file `daa-verify-tpmcontrolled-rogue3.pvz`.

```
equation roguetest(zeta,exp(zeta,z),x,y,z) = true.

define rogue = roguetest(zeta,N,rogueF1,rogueF2,rogueF3).
define rogueid = or(or(eq(id,rogueid1),eq(id,rogueid2)),eq(id,rogueid3)).
```

Note that we did not use the somewhat more natural definition

```
define rogue = or(or(eq(exp(zeta,rogueF1)),eq(exp(zeta,rogueF2))),
                  eq(exp(zeta,rogueF3))).
```

but instead used a definition using a special constructor `roguetest`. Using the more natural definition ProVerif fails to prove security; it seems that the rogue test is simply ignored. A minimal ProVerif example that reproduces this behaviour is given in [4, file `artifacts/or.pv`].

Furthermore, we have to model the fact that the rogue TPMs may already have joined before they were rogue-listed, and that the adversary may know the secret information of the rogue TPMs. This is done by adding the following processes (additionally to those given in the authenticity property without rogue listing):[13]

```
  (let id=rogueid1 in leaktpm) |
  (let id=rogueid2 in leaktpm) |
  (let id=rogueid3 in leaktpm) |
  (let f=rogueF1 in rogueissuer) |
  (let f=rogueF2 in rogueissuer) |
  (let f=rogueF3 in rogueissuer)
```

The authenticity property in this setting is proven in 63 seconds on a Pentium 4, 3 GHz.

The anonymity property from Section 5.4.2 is given by the following code (cf. Table 8):[14]

```
free challengeid1,challengeid2.
free challengecnt1,challengecnt2.
define challengecnt = choice[challengecnt1,challengecnt2].
define challengeid = choice[challengeid1,challengeid2].
define challengecert = choice[challengecert1,challengecert2].
private free int1,int2.

fun corruptid/1.

process
  leakpublic |

  leakissuer |
```

[13] [4], file `daa-verify-tpmcontrolled-rogue3.pvz`.
[14] [4], file `daa-obseq-anonymity4.pvz`.

```
(in(pub,x); let id=corruptid(x) in leaktpm) |

(let (id,cnt,comm,och) = (challengeid1,challengecnt1,pub,int1)
                                                  in tpmjoin) |
(let (id,cnt,comm,och) = (challengeid2,challengecnt2,pub,int2)
                                                  in tpmjoin) |

(let id=challengeid1 in tpmcontrolled) |
(let id=challengeid2 in tpmcontrolled) |

(in(int1,challengecert1); in(int2,challengecert2);
  ((!in(pub,m); let (id,cnt,comm,cert) =
      (challengeid,challengecnt,pub,challengecert) in daasigna) |
   (!in(pub,m); let (id,cnt,comm,cert) =
      (challengeid1,challengecnt1,pub,challengecert1) in daasigna) |
   (!in(pub,m); let (id,cnt,comm,cert) =
      (challengeid2,challengecnt2,pub,challengecert2) in daasigna) |
   (!in(pub,m); in(pub,bsnV); let (id,cnt,comm,cert) =
      (challengeid1,challengecnt1,pub,challengecert1) in daasignp)
   (!in(pub,m); in(pub,bsnV); let (id,cnt,comm,cert) =
      (challengeid2,challengecnt2,pub,challengecert2) in daasignp)
  )
)
```

The two processes to compare are implicitly given by the `choice` operator. The semantics is that the process P_1 is the one resulting from replacing `choice[`x`,`y`]` by x, and P_2 is the one resulting from replacing `choice[`x`,`y`]` by y. Running our compiler and ProVerif one this process directly does not lead to termination. The technique for removing the `land` and `or` constructors that was already described in Section 6.1 helps to ensure termination. In the case of DAA we apply Theorem 4 with $n_{2,3,F_{\mathsf{join}}} = n_{2,4,F_{\mathsf{sign}}} = 1$ and

$$\tilde{\tau}^1_{2,3,F_{\mathsf{join}}} = (f, v; \mathsf{blind}(f, v), \exp(zeta, f), zeta)$$
$$\tilde{\tau}^1_{2,4,F_{\mathsf{sign}}} = (\mathsf{unblind}(\mathsf{blindsign}(\mathsf{blind}(x, z), \mathsf{sk}(y)), z); \exp(\zeta, f), \zeta, \mathsf{pk}(y), m)$$

Then we can remove the equations for `land` and `or` (by Lemma 8). These modification of the equational theory are encoded as follows:[15]

```
compiler AlternativeZKVer(zkver(2;3;joinproof;
  zk(f,v;blind(f,v),exp(zeta,f),zeta; joinproof))).

compiler AlternativeZKVer(zkver(2;4;signproof;
  zk(x,unblind(blindsign(blind(x,z),sk(y)),z);exp(zeta,f),zeta,pk(y),m;
```

[15] [4], file `alternative-zk.pvi`.

```
        signproof))).

compiler RemoveEquations(land).
compiler RemoveEquations(or).
```

After these changes, the proof terminates and we get the result that the two processes are observationally equivalent, i.e., that we have anonymity, after 149 seconds on a Pentium 4, 3 GHz.

The remaining property is that of pseudonymity. Since the encoding of the processes given in Section 5.4.3 for modeling this property does not give any new insights, we refer the reader to the files `daa-obseq-pseudonymity6.pvz` (for the processes P_1, P_2 given in Table 9), `daa-obseq-pseudonymity-attack.pvz` (for the processes $\tilde{P}_1, \tilde{P}_2$ capturing the attack of [26]) and `daa-obseq-pseudonymity-fix.pvz` (modeling $\tilde{P}_1, \tilde{P}_2$ in the fixed version of the protocol with $n_V := n_0$) in [4].

7 Conclusion and Future Work

We have designed an abstraction of non-interactive zero-knowledge protocols in the applied-pi calculus. A novel equational theory for terms characterizes the semantic properties of non-interactive zero-knowledge proofs. Additionally, we propose an encoding into a finite specification in terms of a convergent rewriting system that is accessible to a fully mechanized analysis. The encoding is sound and fully automated. We successfully used the automated protocol verifier ProVerif to obtain the first mechanized analysis of the Direct Anonymous Attestation (DAA) protocol. The analysis in particular required us to come up with suitable abstractions of sophisticated cryptographic security definitions that are based on interactive games; we consider these definitions of independent interest.

Future work on this topic comprises the investigation of computational soundness results, the analysis of other commonly employed protocols based on zero-knowledge, as well as the investigation of interactive zero-knowledge proofs which have additional properties like the impossibility to reproduce a proof after the protocols ends. Furthermore, other, more direct techniques for mechanizing the analysis directly in the original, infinite equational theory might be worth investigating.

References

[1] M. Abadi. Secrecy by typing in security protocols. *Journal of the ACM*, 46(5):749–786, 1999.

[2] M. Abadi, B. Blanchet, and C. Fournet. Just fast keying in the pi calculus. *ACM Transactions on Information and System Security*, 10(3):9, 2007.

[3] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 1999.

[4] M. Backes, M. Maffei, and D. Unruh, 2007. Implementation of the compiler from zero-knowledge protocol descriptions into ProVerif-accepted specifications. Available at `http://www.infsec.cs.uni-sb.de/zk.zip`.

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

[6] B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW)*, pages 82–96. IEEE Computer Society Press, 2001.

[7] B. Blanchet and C. Fournet. Automated verification of selected equivalences for security protocols. In *Proc. 20th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 331–340. IEEE Computer Society Press, 2005.

[8] D. Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS. In *Advances in Cryptology: CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 1–12. Springer-Verlag, 1998.

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

[10] F. Butler, I. Cervesato, A. D. Jaggard, A. Scedrov, and C. Walstad. Formal analysis of kerberos 5. *Theoretical Computer Science*, 367(1):57–87, 2006.

[11] D. E. Denning and G. M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, 1981.

[12] D. Dolev, C. Dwork, and M. Naor. Nonmalleable cryptography. *SIAM Journal on Computing*, 30(2):391–437, 2000.

[13] D. Fisher. Millions of .Net Passport accounts put at risk. *eWeek*, May 2003. (Flaw detected by Muhammad Faisal Rauf Danka).

[14] C. Fournet, A. D. Gordon, and S. Maffeis. A type discipline for authorization in distributed systems. In *Proc. 20th IEEE Symposium on Computer Security Foundations (CSF)*, pages 31–45. IEEE Computer Society Press, 2007.

[15] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game – or – a completeness theorem for protocols with honest majority. In *Proc. 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.

[16] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):690–728, 1991. Online available at `http://www.wisdom.weizmann.ac.il/~oded/X/gmw1j.pdf`.

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

[18] R. Kemmerer. Analyzing encryption protocols using formal verification techniques. *IEEE Journal on Selected Areas in Communications*, 7(4):448–457, 1989.

[19] S. Kramer. *Logical Concepts in Cryptography.* PhD thesis, École Polytechnique Fédérale de Lausanne, 2007.

[20] S. Kremer and M. Ryan. Analysis of an electronic voting protocol in the applied pi-calculus. In *Proc. 14th European Symposium on Programming (ESOP)*, Lecture Notes in Computer Science, pages 186–200. Springer-Verlag, 2005.

[21] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. 2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 1996.

[22] C. Meadows. Using narrowing in the analysis of key management protocols. In *Proc. 10th IEEE Symposium on Security & Privacy*, pages 138–147, 1989.

[23] J. K. Millen. The interrogator: A tool for cryptographic protocol security. In *Proc. 5th IEEE Symposium on Security & Privacy*, pages 134–141, 1984.

[24] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 12(21):993–999, 1978.

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

[26] B. Smyth, L. Chen, and M. D. Ryan. Direct anonymous attestation: ensuring privacy with corrupt administrators. In *Proceedings of the Fourth European Workshop on Security and Privacy in Ad hoc and Sensor Networks*, number 4572 in Lecture Notes in Computer Science, pages 218–Â231. Springer-Verlag, 2007.

[27] F. J. Thayer Fabrega, J. C. Herzog, and J. D. Guttman. Strand spaces: Why is a security protocol correct? In *Proc. 19th IEEE Symposium on Security & Privacy*, pages 160–171, 1998.

[28] D. Wagner and B. Schneier. Analysis of the SSL 3.0 protocol. In *Proc. 2nd USENIX Workshop on Electronic Commerce*, pages 29–40, 1996.