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

Michael Backes
Max Planck Institute for Software Systems and Saarland University
backes@cs.uni-sb.de

Matteo Maffei Dominique Unruh
Saarland University
{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. This required us to devise novel abstractions of sophisticated cryptographic security definitions based on interactive games. The analysis reported a novel attack on DAA that was overlooked in its existing cryptographic security proof. We propose a revised variant of DAA that we successfully prove secure using ProVerif.

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, 26], over carefully designed de-facto standards like SSL and PKCS [30, 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 [25, 24, 20, 23, 27, 29, 5, 6]. 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 tasks: they allow for commonly evaluating a function on distributed inputs without reveal-

ing 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 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 Contributions

The contribution of this paper is threefold: First, we present an abstraction of non-interactive zero-knowledge proofs within the applied pi-calculus [4] 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 [7], 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 authentication scheme for Trusted Platform Modules (TPMs), yielding its first mechanized security proof. The analysis reported a novel attack on DAA that was overlooked in its existing cryptographic security proof. We propose a revised variant that we successfully prove secure.

We express cryptographic protocols in the applied pi-calculus, an extension of the pi-calculus with an arbitrary equational theory for terms, that has proven to constitute a salient foundation for the analysis of cryptographic protocols, see [3, 22, 7, 2, 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 in a general way zero-knowledge proofs aims at formalizing in modal logic the informal prose used to describe the properties of these proofs [21]. In contrast to our abstraction, the abstraction in [21] 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 [7] 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 notebooks. The DAA protocol relies heavily on zero-knowledge proofs to achieve its anonymity guarantees. 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 analysis confirmed a known attack on anonymity [28] and discovered a new attack on authenticity. We propose a revised variant and prove it secure.

The proofs are fully automated using ProVerif. We are confident that the methodology presented in this paper is general and 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*
		$u(x).P$	*input*
		$\overline{u}\langle N\rangle.P$	*output*
		$P \mid Q$	*par*
		$!P$	*repl*

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 elaborates on the analysis of DAA, the description of its security properties, and the use of ProVerif for mechanizing the analysis. Section 6 concludes and outlines future work.

2 Review of the Applied Pi-calculus

The syntax of the applied pi-calculus [4] 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 let u range over names and variables. We partition each signature into *public* and *private* function symbols. The only difference is that private symbols are not available to the adversary: For more detail on their semantics, we refer to the long version [19]. Private function symbols are supported by ProVerif and are used in the finite encoding of the equational theory for zero-knowledge proofs and in the DAA model. In the following, functions symbols are public unless stated otherwise. 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.

Table 2 Internal reduction

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

THEN
$$if\, M = M\ then\ P\ else\ Q \rightarrow P$$

ELSE
$$\frac{E \nvdash M = N \qquad M, N \text{ ground}}{if\, M = N\ then\ P\ else\ Q \rightarrow Q}$$

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; $u(x).P$ receives a message N from the channel u and then behaves as $P\{N/x\}$; $\overline{u}\langle N\rangle.P$ outputs the message N on the channel u and then behaves as P; $P \mid Q$ executes P and Q in parallel; $!P$ generates an unbounded number of copies of P.

As in the pi-calculus, the semantics 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. Internal reduction defines the semantics for extended processes.

Definition 1 (Internal Reduction) *Internal reduction ($\rightarrow$) is the smallest relation on extended processes that satisfies the rules in Table 2 and that is closed under structural equivalence and under application of evaluation contexts.*

Observational equivalence constitutes an equivalence relation that captures the equivalence of processes with respect to their dynamic behavior. Intuitively, two processes P and Q are observationally equivalent, written $P \approx Q$, if they are indistinguishable to an observer inspecting the messages sent and received on public channels. Due to space constraints, we postpone the formal definition of observational equivalence to Appendix A.

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 3. (Note again though that any other base the-

Table 3 A base equational theory containing basic cryptographic primitives and logical operators

$$\Sigma_{\mathsf{base}} = \left\{ \begin{array}{l} \mathsf{pair}, \mathsf{enc_{sym}}, \mathsf{dec_{sym}}, \mathsf{enc_{asym}}, \mathsf{dec_{asym}}, \\ \mathsf{sign}, \mathsf{ver}, \mathsf{msg}, \mathsf{pk}, \mathsf{sk}, \mathsf{h}, \mathsf{bl}, \\ \mathsf{unbl}, \mathsf{blsign}, \mathsf{blver}, \mathsf{blmsg}, \\ \wedge, \vee, \mathsf{eq}, \mathsf{first}, \mathsf{snd}, \mathsf{true}, \mathsf{false} \end{array} \right\}$$

ver and blver of arity 3, pair, $\mathsf{enc_{sym}}$, $\mathsf{dec_{sym}}$, $\mathsf{enc_{asym}}$, $\mathsf{dec_{asym}}$, sign, bl, unbl, blsign, $\wedge$, $\vee$ and eq of arity 2, msg, pk, sk, h, blmsg, 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:

$$\begin{array}{lcl} \mathsf{first}(\mathsf{pair}(x,y)) & = & x \\ \mathsf{snd}(\mathsf{pair}(x,y)) & = & y \\ \mathsf{dec_{sym}}(\mathsf{enc_{sym}}(x,y),y) & = & x \\ \mathsf{dec_{asym}}(\mathsf{enc_{asym}}(x,\mathsf{pk}(y)),\mathsf{sk}(y)) & = & x \\ \mathsf{msg}(\mathsf{sign}(x,y)) & = & x \\ \mathsf{blver}(\mathsf{unbl}(\mathsf{blsign}(\mathsf{bl}(x,z),\mathsf{sk}(y)),z),x,\mathsf{pk}(y)) & = & \mathsf{true} \\ \mathsf{blmsg}(\mathsf{unbl}(\mathsf{blsign}(\mathsf{bl}(x,z),y),z)) & = & x \\ \mathsf{ver}(\mathsf{sign}(x,\mathsf{sk}(y)),x,\mathsf{pk}(y)) & = & \mathsf{true} \\ \mathsf{eq}(x,x) & = & \mathsf{true} \\ \wedge(\mathsf{true},\mathsf{true}) & = & \mathsf{true} \\ \vee(\mathsf{true},x) & = & \mathsf{true} \\ \vee(x,\mathsf{true}) & = & \mathsf{true} \end{array}$$

ory 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 [3] and the electronic voting protocol FOO 92 [22]. 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 4; its components are explained in the following. A non-interactive zero-knowledge proof is represented as a term

Table 4 Equational theory for zero-knowledge

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

$\mathsf{ZK}_{i,j}$ of arity $i+j+1$, $\mathsf{Ver}_{i,j}$ of arity 2, Public_i and 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 $\widetilde{M}, \widetilde{N}, F$:

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

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 constant without names and variables, which is built upon distinguished nullary functions α_i and β_i with $i \in \mathbb{N}$.

Definition 2 ((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,

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

denotes a zero-knowledge proof that the term $\mathsf{enc_{sym}}(m,k)$ is an encryption of m with k. More precisely, the statement reads: "There exists a key such that the ciphertext $\mathsf{enc_{sym}}(m,k)$ is an encryption of m with this key". As mentioned before, $\mathsf{enc_{sym}}(m,k)$ 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:

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

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

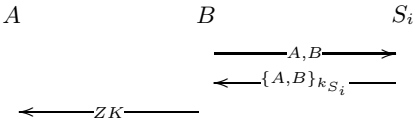
the *zero-knowledge* property discussed in the introduction. We define a statement $\mathsf{ZK}_{i,j}(\widetilde{M}, \widetilde{N}, F)$ to hold true if F is an (i,j)-formula and the formula obtained by substituting all α_k's and β_l's in F with the corresponding values M_k and N_l is valid. Verification 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 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:

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

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:

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

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. 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 [7], 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 *terms*(M) denote the set of subterms of M and *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 3 (Process Validity) *A term Z is $(\mathcal{F}, h, l)$-valid if and only if the following conditions hold:*

1. *for every $\mathsf{ZK}_{i,j}(\widetilde{M}, \widetilde{N}, F) \in$ terms(Z) and $\mathsf{Ver}_{i,j}(F, M) \in$ 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 $\mathsf{Public}_p(M) \in$ 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 $M \in$ terms(P).

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 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.

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

We obtain a finite set of rules since the number of $\mathsf{ZK}_{i,j}^F$ and $\mathsf{Ver}_{i,j}^F$ constructors corresponds to the (finite) number of formulas occurring in the process specification. The $\mathsf{PZK}_{i,j}^F$ functions are private; hence they cannot be used by the adversary to derive terms of the form $\mathsf{PZK}_{i,j}^F(\widetilde{M}, \widetilde{N}, \mathsf{true})$, 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.

Table 5 Finite equational theory for zero-knowledge with respect to an $(\mathcal{F}, h, l)$-valid process.

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

$\mathsf{PZK}_{i,j}^F$ of arity $i+j+1$, $\mathsf{ZK}_{i,j}^F$ of arity $i+j$, FakeZK_k of arity $k+2$, FakeVer of arity 4, FakePublic and FakeCollect of arity 2, $\mathsf{Ver}_{i,j}^F$, Public_p, and 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 $(i,j,F) \in \mathcal{F}$:

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

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 $\Sigma_{\mathsf{FZK}}^{\mathcal{F},h,l}$ 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 $\Sigma_{\mathsf{FZK}}^{\mathcal{F},h,l}$ functions FakeCollect, FakePublic, and FakeVer. These functions are only used in the proofs and do not occur in any equations.

4.2 Compilation into Finite Form

We now define the static compilation of terms and processes.

Definition 4 (Static Compilation) *The* $(\mathcal{F},h,l)$-static compilation *is the partial function* $\sigma : T_{\Sigma_{\mathsf{ZK}}} \to T_{\Sigma_{\mathsf{FZK}}^{\mathcal{F},h,l}}$ *recursively defined as follows:*

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

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. Due to space constraints, we refer the interested reader to the long version [19] for the proof of this theorem.

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* $P\sigma \approx_{E_{\mathsf{FZK}}^{\mathcal{F},h,l}} Q\sigma$*, then* $P \approx_{E_{\mathsf{ZK}}} Q$*.*

In the long version [19], we additionally prove that a comprehensive class of trace-based properties that includes secrecy and authenticity carries over to the original process specifications.

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 notebooks.

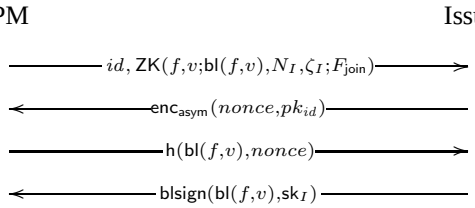
The goal of the DAA protocol is to enable the TPM to sign arbitrary messages and to send them to an entity called the verifier in such a way that the verifier will only learn that a valid TPM signed that message, but without revealing the TPM's identity. The DAA protocol relies heavily on zero-knowledge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

The 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. The DAA protocol includes also a rogue-tagging procedure preventing corrupted TPMs from getting certificates and authenticating messages. Due to space constraints, we omit the specification and the analysis of the rogue-tagging procedure. We refer the interested reader to the long version [19].

Every TPM has a unique id as well as a key-pair called *endorsement key* (EK). The issuer is assumed to know the public component of each EK. 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 $f_{cnt} := \mathsf{h}(daaseed_{id}, cnt)$ 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 $\mathsf{bl}(f, v)$, for some random v, instead of just f. Furthermore, 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. This message is used in the rogue-tagging procedure mentioned before. 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 $\mathsf{bl}(f, v)$, 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 $\mathsf{bl}(f, v)$ and returns this signature $x := \mathsf{blsign}(\mathsf{bl}(f, v), sk_I)$. Then $cert := \mathsf{unbl}(x, v)$ 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 by a challenge-response nonce handshake: the issuer outputs a nonce encrypted by the TPM's public endorsement key, and the TPM proves its identity by hashing the nonce together with the blinded f-value $\mathsf{bl}(f, v)$. The join protocol has the following overall shape:



with $F_{\mathsf{join}} := (\beta_1 = \mathsf{bl}(\alpha_1, \alpha_2) \wedge \beta_2 = \mathsf{exp}(\beta_3, \alpha_1))$. In our calculus, we can model the behavior of the TPM in the join protocol as reported in Table 6. Here we use $let\ x = M\ in\ P$ 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 seed taking as input *id*. In the computation of $\zeta_I := \mathsf{h}(\mathsf{pair}(n_1, bsn_I))$, 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 public keys pk_{id} and pk_I are modeled as $\mathsf{pk}(\mathsf{ek}(id))$ and $\mathsf{pk}(\mathsf{issuerK})$ where ek and issuerK are private constructors. That is, by $\mathsf{ek}(id)$ we model a secret function mapping a TPM's identity to the endorsement key pair. We then use the operators sk and pk to access the secret and the public key. The function 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.

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,

Table 6 TPM/Host and issuer in the join protocol

$\mathsf{tpmjoin} :=\ let\ f = \mathsf{h}(\mathsf{pair}(\mathsf{seed}(id), cnt))\ in$
$\nu v.$
$let\ U = \mathsf{bl}(f, v)\ in$
$let\ \zeta_I = \mathsf{h}(\mathsf{pair}(n_1, bsn_I))\ in$
$let\ N_I = \mathsf{exp}(\zeta_I, f)\ in$
$let\ zkp = \mathsf{ZK}(f, v; U, N_I, \zeta_I; F_{\mathsf{join}})\ in$
$\overline{pub}\langle \mathsf{pair}(id, zkp)\rangle.$
$pub(encn).$
$let\ nonce = \mathsf{dec}_{\mathsf{sym}}(encn, \mathsf{sk}(\mathsf{ek}(id)))\ in$
$\overline{pub}\langle \mathsf{h}((U, nonce))\rangle$
$pub(x).$
$let\ cert = \mathsf{unbl}(x, v)\ in$
$if\ \mathsf{blindver}(cert, f, \mathsf{pk}(\mathsf{issuerK})) = \mathsf{true}\ then$
$\mathsf{event}\ \mathsf{JOINED}(id, cnt, cert).$
$\overline{och}\langle cert\rangle$

$\mathsf{issuer} :=\ !\, pub(msg).$
$let\ id = \mathsf{first}(msg)\ in$
$let\ zkp = \mathsf{snd}(msg)\ in$
$if\ \mathsf{Ver}_{2,3}(F_{\mathsf{join}}; zkp) = \mathsf{true}\ then$
$let\ U = \mathsf{Public}_1(zkp)\ in$
$\nu nonce.$
$\overline{pub}\langle \mathsf{enc}_{\mathsf{asym}}(nonce, \mathsf{pk}(ek(id)))\rangle.$
$pub(hashn).$
$if\ hashn = \mathsf{h}((U, nonce))\ then$
$let\ N = \mathsf{Public}_2(zkp)\ in$
$let\ \zeta = \mathsf{Public}_3(zkp)\ in$
$if\ \zeta = \mathsf{h}(\mathsf{pair}(n_1, bsn_I))\ then$
$let\ cert = \mathsf{blsign}(U, \mathsf{sk}(\mathsf{issuerK}))\ in$
$\mathsf{event}\ \mathsf{CERTIFIED}(id)$
$\overline{pub}\langle cert\rangle$

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:

TPM — $\mathsf{ZK}(f, cert; N, \zeta, \mathsf{pk}_I, m; F_{\mathsf{sign}})$ → Verifier

Table 7 TPM/Host and verifier in the DAA-sign protocol

$\mathsf{daasigna} :=$
$\nu\zeta.$
$let\ f = \mathsf{h}(\mathsf{pair}(\mathsf{seed}(id), cnt))\ in$
$let\ N = \mathsf{exp}(\zeta, f)\ in$
$let\ zkp = \mathsf{ZK}(f, cert; N, \zeta, \mathsf{pk}(\mathsf{issuerK}), m; F_{\mathsf{sign}})\ in$
$\mathsf{event}\ \mathsf{DAASIGNA}(id, cnt, m).$
$\overline{pub}\langle zkp\rangle$

$\mathsf{daavera} :=$
$pub(zkp).$
$if\ \mathsf{Ver}_{2,4}(F_{\mathsf{sign}}; zkp) = \mathsf{true}\ then$
$let\ N = \mathsf{Public}_1(zkp)\ in$
$let\ \zeta = \mathsf{Public}_2(zkp)\ in$
$if\ \mathsf{Public}_3(zkp) = \mathsf{pk}(\mathsf{issuerK})\ then$
$let\ m = \mathsf{Public}_4(zkp)\ in$
$\mathsf{event}\ \mathsf{DAAVERA}(m)$

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 host. 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.

The processes for the anonymous variant is reported in Table 7. 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 $let\ \zeta = \mathsf{h}(\mathsf{pair}(n_1, bsn_V))\ in$. The corresponding verification process daaverp is defined like daavera, except that after $let\ \zeta = \mathsf{Public}_2(zkp)\ in$ we insert $if\ \zeta = \mathsf{h}(\mathsf{pair}(n_1, bsn_V))\ then$. Furthermore, to be able to formulate a more fine-grained authenticity property below, we output the more informative events $\mathsf{DAASIGNP}(id, cnt, bsn_V, m)$ and $\mathsf{DAAVERP}(m, bsn_V, N)$ instead of $\mathsf{DAASIGNA}(id, cnt, m)$ and $\mathsf{DAAVERA}(m)$, respectively.

5.3 Security properties of DAA

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

5.3.1 Authenticity of the DAA-sign protocol

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 the execution of an arbitrary number of TPMs in an arbitrary fashion (except for learning their secrets). We model this process as follows:

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

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, 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:

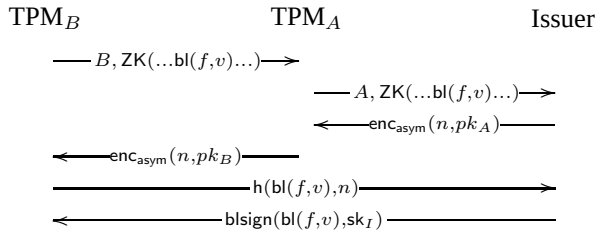
$$\mathsf{DAAVERP}(m, bsn, N) \Rightarrow \mathsf{DAASIGNP}(id, cnt, bsn, m)$$
$$\mathsf{DAAVERA}(m) \Rightarrow \big(\mathsf{DAASIGNA}(id, cnt, m) \vee \mathsf{DAASIGNP}(id, cnt, bsn, m)\big).$$

For a formal definition of these trace properties, we refer to the full version [19]. Intuitively, the first property means that if an event $\mathsf{DAAVERP}(m, bsn, N)$ occurs, then also $\mathsf{DAASIGNP}(id, cnt, bsn, m)$ 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.)

Trace properties such as the above authenticity properties can be verified with the mechanized prover ProVerif [7]. 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 the full version [19]. The tool implementing the compiler from Section 4.1 can be found at [18].

5.3.2 Authenticity of the Join Protocol

In contrast to the DAA-sign operation, after a join the issuer learns the identity of the joining party (since the joining party authenticates itself using its endorsement key). This is necessary for verifying that no rogue TPM joins but is also used to limit the number of times a given TPM can join with respect to different f-values. It is therefore a natural question whether the following authenticity property holds: $\mathsf{JOINED}(id, cnt, cert) \Rightarrow \mathsf{CERTIFIED}(id)$. As it turned out, ProVerif proves that the property is not fulfilled and finds the following attack:



The adversary corrupts TPM$_A$ and retrieves its endorsement key. When the (uncorrupted) TPM$_B$ joins, it sends a ZK proof Z containing some f-value f to the issuer and authenticates Z as coming from B. The adversary intercepts Z and sends Z to the issuer and authenticates Z as coming from A. The issuer checks the authentication and the ZK proof. Since the ID of B is not included in the ZK proof[1] the issuer accepts and issues a certificate for f. Then the adversary forwards the certificate to B and B successfully checks the certificate. After this interaction, the issuer believes to have certified A, and B has successfully joined. To the best of our knowledge, this attack was not known before. Note that this attack also violates the security guarantees given in [9].[2] Fortunately, the DAA protocol can be easily modified to exclude this attack: One simply includes the ID of the joining TPM in the ZK proof as an additional public parameter. Then the issuer checks whether the ZK proofs contains the correct ID.[3] Using ProVerif and our compiler, we could then successfully verify that the modified protocol indeed satisfies the above trace property. Note that this

[1] At least not directly. An honestly generated f-value depends on the ID, however, this is not verified in the ZK proof.

[2] They defined security via an ideal functionality. In this ideal functionality, the issuer is notified when a party joins, and a party can check whether it joined successfully (e.g., by DAA-signing a message).

[3] Note that the statement F_{join} is not changed, we do not prove anything about the ID. However, similar to the message m in the signing operation, the attacker cannot replace the ID by some other ID without producing a new proof from scratch.

modification can be applied to the original DAA protocol from [9] without loosing efficiency by including the ID of the TPM in the hash value $c := \mathsf{h}(c_h \| n_t)$ (see the protocol description in [9]).

Besides the property $\mathsf{JOINED}(id, cnt, cert) \Rightarrow \mathsf{CERTIFIED}(id)$, there are a few more properties that are related to the authenticity of the join protocol. First,

$$\mathsf{CERTIFIED}(id, \exp(z, \mathsf{h}(\mathsf{pair}(\mathsf{seed}(id_2), cnt)))) \Rightarrow$$
$$\mathsf{STARTJOIN}(id, \exp(z, \mathsf{h}(\mathsf{pair}(\mathsf{seed}(id), cnt)))) \wedge id{=}id_2$$
$$\vee \quad \mathsf{LEAKTPM}(id) \wedge \mathsf{LEAKTPM}(id_2).$$

Here the event $\mathsf{STARTJOIN}(id, f)$ is defined to be raised as soon as an honest TPM tries to join with ID id and f-value f (i.e., this event is raised at the beginning of tpmjoin). Intuitively this property means, if some process joins using an f-value that belongs to id_2 but the issuer believes that id joined, then (i) the join was performed by an honest TPM id and $id_2 = id$, or (ii) the adversary has corrupted both TPMs id and id_2 (i.e., accessed their internal secrets). Further, we want that it is only possible that a message is successfully signed using an f-value belonging to an ID id if the TPM with that ID has either been certified or corrupted.

$$\mathsf{DAAVERA}(m, \exp(z, \mathsf{h}(\mathsf{pair}(\mathsf{seed}(id), cnt)))) \Rightarrow$$
$$\mathsf{CERTIFIED}(id, N) | \mathsf{LEAKTPM}(id).$$

$$\mathsf{DAAVERP}(m, bsn, \exp(z, \mathsf{h}(\mathsf{pair}(\mathsf{seed}(id), cnt)))) \Rightarrow$$
$$\mathsf{CERTIFIED}(id, N) | \mathsf{LEAKTPM}(id).$$

Note that we cannot guarantee that if the TPM id is corrupted then it uses an f-value that belongs to it. It can use an f-value that belongs to another corrupted TPM. So the above property only guarantees that a corrupted TPM cannot "steal" the f-value of another uncorrupted TPM. (As done in the attack described above.)

All these property are proven by ProVerif not to hold in the original DAA protocol but to hold in our modified protocol.

5.3.3 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:

$$\begin{aligned} P_i := \; & \mathsf{leak} \mid \\ & (let\ (id, cnt, och) = (id_1, n_1, int_1)\ in\ \mathsf{tpmjoin}) \mid \\ & (let\ (id, cnt, och) = (id_2, n_1, int_1)\ in\ \mathsf{tpmjoin}) \mid \\ & (int_1(cert_1).int_2(cert_2). \\ & let\ (id, cnt, cert) = (id_i, n_1, cert_i)\ in\ \mathsf{daasigna}) \end{aligned}$$

with

$$\mathsf{leak} := \begin{array}{l} (!pub(id).\overline{pub}\langle \mathsf{pk}(\mathsf{ek}(id))\rangle) \mid \\ \overline{pub}\langle \mathsf{pk}(\mathsf{issuerK})\rangle \mid \overline{pub}\langle \mathsf{sk}(\mathsf{issuerK})\rangle \end{array}$$

where id_1, id_2, n_1 are free names and int_1, 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, a closer inspection reveals that this property is not very general. For example, it does not cover 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 non-corrupted 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 corrupt in (1) is used to generate an infinite supply of ids of corrupted TPMs. Finally, we additionally give the observer the capability to distinguish the messages sent by the challenge TPM from the messages sent by the other processes: this is technically achieved by letting the challenge TPM use a different public channel $pubT$.

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. The proof succeeds and the running time is 117 seconds on a Pentium 4, 3 GHz. A more detailed description is again given in the full version [19].

Pseudonymity can be modelled similarly to anonymity, but additional care has to be taken so that the adversary cannot induce the challenge TPM to pseudonymously sign messages in a way that the adversary can link them to the challenge signatures. Note that due to an attack on DAA by [28], the pseudonymity property does not hold if the basename of the issuer and the basename of the verifier can coincide. Therefore we use a slightly more restrictive pseudonymity property which explicitly excludes that the issuer and the verifier use the same basename. Alternatively, if we apply the correction of the protocol proposed by [28], we can prove a stronger pseudonymity property where the issuer and the verifier basename may coincide. If we use the original version of DAA and the stronger pseudonymity property, our tool finds the attack by [28]. For space reasons, we will not discuss the modelling of pseudonymity but refer to the long version [19].

6 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. 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.

[3] 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.

[4] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. 28th Symposium on Principles of Programming Languages (POPL)*, pages 104–115. ACM Press, 2001.

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

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

[7] 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.

[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.

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.

$$
\begin{array}{lll}
[1] & P_i := \mathsf{leak} \mid (pub(x).\ let\ id = \mathsf{corrupt}(x)\ in\ \overline{pub}\langle \mathsf{seed}(id)\rangle.\overline{pub}\langle \mathsf{sk}(\mathsf{ek}(id))\rangle) \mid & (1) \\
[3] & \quad (let\ (id, cnt, och) = (id_1, cnt_1, int_1)\ in\ \mathsf{tpmjoin}) \mid & (2) \\
[3] & \quad (let\ (id, cnt, och) = (id_2, cnt_2, int_2)\ in\ \mathsf{tpmjoin}) \mid & (3) \\
[4] & \quad (let\ id = id_1\ in\ \mathsf{TPMs}) \mid (let\ id = id_2\ in\ \mathsf{TPMs}) \mid & (4) \\
[3] & \quad \big(int_1(cert_1).int_2(cert_2). & (5) \\
[5] & \quad \big((!pub(m).\ let\ (id, cnt, cert, pub) = (id_i, cnt_i, cert_i, pubT)\ in\ \mathsf{daasigna}) \mid & (6) \\
[4] & \quad (!pub(m).\ let\ (id, cnt, cert) = (id_1, cnt_1, cert_1)\ in\ \mathsf{daasigna}) \mid & (7) \\
[4] & \quad (!pub(m).\ let\ (id, cnt, cert) = (id_2, cnt_2, cert_2)\ in\ \mathsf{daasigna}) \mid & (8) \\
[4] & \quad (!pub(m).\ pub(bsn_V).\ let\ (id, cnt, cert) = (id_1, cnt_1, cert_1)\ in\ \mathsf{daasignp}) \mid & (9) \\
[4] & \quad (!pub(m).\ pub(bsn_V).\ let\ (id, cnt, cert) = (id_2, cnt_2, cert_2)\ in\ \mathsf{daasignp})\big) & (10)
\end{array}
$$

[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] Omitted. Implementation of the compiler from zero-knowledge protocol descriptions into proverif-accepted specifications., 2007. Available at `http://www.geocities.com/zkapplied`.

[19] Omitted. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol (full version), 2007. Available at `http://www.geocities.com/zkapplied`.

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

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

[22] 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.

[23] 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.

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

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

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

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

[28] 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.

[29] 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.

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

A Semantics and Observational Equivalence

The semantics of the applied pi-calculus is defined on *extended processes*, which are defined as follows:

Extended Processes

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

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, i.e., it does not contain free variables. A *frame* is an extended process built up from **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 **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.

As mentioned in Section 2, the semantics 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 5 (Structural Equivalence) *Structural equivalence ($\equiv$) is the smallest equivalence relation on extended processes that satisfies the rules in Table 9 and that is closed under α-renaming, i.e., renaming of bound names and variables, and under application of evaluation contexts.*

Table 9 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 $E \vdash M = N$

In the following, 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 6 (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 $C[\_]$.*