Computationally Sound, Automated Proofs for Security Protocols

Véronique Cortier[1,⋆] and Bogdan Warinschi[2,⋆⋆]

[1] Loria, CNRS, Nancy, France
fax: (+33) 3 83 27 83 19
cortier@loria.fr
[2] Computer Science Department,
University of California at Santa Cruz, USA
bogdan@soe.ucsc.edu

Abstract. Since the 1980s, two approaches have been developed for analyzing security protocols. One of the approaches relies on a computational model that considers issues of complexity and probability. This approach captures a strong notion of security, guaranteed against all probabilistic polynomial-time attacks. The other approach relies on a symbolic model of protocol executions in which cryptographic primitives are treated as black boxes. Since the seminal work of Dolev and Yao, it has been realized that this latter approach enables significantly simpler and often automated proofs. However, the guarantees that it offers have been quite unclear.

In this paper, we show that it is possible to obtain the best of both worlds: fully automated proofs and strong, clear security guarantees. Specifically, for the case of protocols that use signatures and asymmetric encryption, we establish that symbolic integrity and secrecy proofs are sound with respect to the computational model. The main new challenges concern secrecy properties for which we obtain the first soundness result for the case of active adversaries. Our proofs are carried out using Casrul, a fully automated tool.

1 Introduction

Security protocols are short programs designed to achieve various security goals, such as data privacy and data authenticity, even when the communication between parties takes place over channels controlled by an attacker. Their ubiquitous presence in many important applications makes designing and establishing the security of such protocols a very important research goal. Unfortunately, attaining this goal seems to be quite a difficult task, and many of the protocols that had been proposed have been found to be flawed.

Starting in the early '80s, two distinct and quite different methods have emerged in an attempt to ground the security of protocols on firm, rigorous mathematical foundations.

⋆ Véronique Cortier's work was partly supported by the ACI Jeunes Chercheurs Crypto and the RNTL project PROUVE-03V360.

⋆⋆ Bogdan Warinschi was partly supported by the National Science Foudation Grants CCR-0204162 and CCR-0208800.

M. Sagiv (Ed.): ESOP 2005, LNCS 3444, pp. 157–171, 2005.

They are generically known as the computational (or the cryptographic) approach and the symbolic (or the Dolev-Yao) approach.

Under the computational approach, the security of protocols is based on the security of the underlying primitives, which in turn is proved assuming the hardness of solving various computational tasks such as factoring or taking discrete logarithms. The main tools used for proofs are *reductions*: to prove a protocol secure one shows that a successful adversary against the protocol can be efficiently transformed into an adversary against some primitive used in its construction. Here, quantification is universal over *all* possible probabilistic polynomial-time (p.p.t.) adversaries and the execution model that is analyzed is specified down to the bit-string level. Proofs in the computational model imply strong guarantees (security holds in the presence of an *arbitrary* probabilistic polynomial-time adversary). At the same time however, security reductions for even moderately-sized protocols become extremely long, difficult, and tedious.

The central characteristics of the symbolic approach are an abstract view of the execution and a significantly limited adversary. More precisely, in this model, the implementation details of the primitives are abstracted away, and the execution is modeled only symbolically. Furthermore, the actions of the adversary are quite constrained. For instance, it is postulated that it can recover the plaintext underlying a ciphertext only if it can derive the appropriate decryption key. The resulting execution models are rather simple and can easily be handled by automated tools. In fact, many security proofs have already been carried out using model checkers [16] and theorem provers [19]. Unfortunately, the high degree of abstraction and the limited adversary raise serious questions regarding the security guarantees offered by such proofs, especially from the perspective of the computational model.

Recently, a significant research effort has been directed at bridging the gap between the two approaches [3, 18, 5, 17]. The idea is to determine condition under which symbolic analysis is sound with respect to standard computational models. This path promises tremendous benefits: protocols can be analyzed and proved secure using the simpler, automated methods specific to the symbolic approach, yet the security guarantees are with respect to the more comprehensive computational model. In this paper we extend and apply the work of Micciancio and Warinschi [17] to demonstrate for the first time that *fully automated* security proofs with clear computational implications are indeed possible.

Specifically, our results are as follows. First, we give a language for specifying protocols. The syntax of our language is close to that of Casrul and allows the use of random nonces, digital signatures and public-key encryption. For protocols specified in this language we give two kinds of executions for protocols. Each of these models considers a powerful *active* adversary that controls and potentially tampers with the communication in an unbounded number of sessions of the protocol executed by honest users. The first model is a computational model in which the adversary is an arbitrary p.p.t. algorithm. The second model is symbolic, and the adversary is a typical Dolev-Yao adversary. One crucial property of the latter model is that it actually coincides with the execution semantics used by an existing automated tool called Casrul. We then link the two models in several ways.

Our first contribution (Theorem 1) is a soundness theorem for proofs of trace properties: if *all* symbolic traces of a protocol satisfy a certain predicate (*i.e.* the protocol

is secure in the symbolic model), then the concrete traces satisfy the same predicate with overwhelming probability against p.p.t. adversaries (*i.e.* the protocol is secure in the computational model). Our result is a proper extension of a similar theorem of [17] to protocols that besides nonces and public-key encryption also use digital signatures.

Our second main result concerns soundness of secrecy proofs. This issue is significantly more challenging since unlike in the case of trace properties, secrecy is formalized in quite different ways in the two models that we consider: inability of deriving the secret in the formal world[1] and indistinguishability of adversary's views in the computational world. Nevertheless, we are able to prove that in the case of nonces, symbolic secrecy implies computational secrecy.

Although our theorems justify formal analysis as used in Casrul [9], we also briefly considered other automatic tools, such as Proverif [7], Casper [16], and Securify [10] and we strongly believe that similar soundness results could be obtained for these tools also. While our choice was mainly determined by our familiarity with Casrul (one of the authors is a close collaborator of the team that develops Casrul) an additional factor was that most of the tools dedicated to an unbounded number of sessions allow only for proofs of secrecy and not for authenticity.

Related work. The rationale behind the need for soundness theorems was outlined by Abadi [1] and the first such result was obtained by Abadi and Rogaway [3]. Quite a few other results followed, and here we recall those that are closest to our work. These include the soundness theorem for secrecy properties given by Abadi and Rogaway for symmetric encryption in the presence of passive adversaries [3]. Another results is that of Laud [14] who shows soundness of confidentiality properties for symmetric encryption in a model with a fixed number of sessions. A soundness result for trace properties was proved by Micciancio and Warinschi [17] for a language that used random nonces and public-key encryption. In this paper we extend their work to also include digital signature and ciphertext forwarding. Soundness of trace properties for an even richer language that includes in addition symmetric encryption and authentication was given by Backes, Pfitzmann, and Waidner [5] and work in progress is aimed at achieving soundness for secrecy of symmetric keys [4]. While it is conceivable that building upon these results at least partial automation of symbolic proofs can be achieved, this work still remains to be carried out.

The rest of the paper is structured as follows. In Section 2 we briefly recall digital signatures and public-key encryption schemes. We present the protocol syntax in Section 3 and the two execution models in Section 4. In Section 5 we define generic security properties and prove our soundness theorems for trace and secrecy properties. Section 6 concludes with a discussion regarding the implications of our results on the proofs done with Casrul.

2 Computational Cryptography

In this paper we will use a generic digital signature scheme $\mathcal{DS} = (\mathsf{K_s}, \mathsf{Sig}, \mathsf{Vf})$ given, as usual, by algorithms for key generation, signing and verifying. Also, we consider an

[1] Secrecy can alternatively be defined using an equivalence based formulation, as in the spi-calculus [2] for example, but in this paper we concentrate on the formulation used in Casrul.

arbitrary public-key encryption scheme $\mathcal{AE} = (\mathsf{K_e}, \mathsf{Enc}, \mathsf{Dec})$ given by algorithms for key generation, encryption and decryption. For a precise specification of their syntax we refer to [11].

Traditionally, security is defined for each individual primitive separately. Since the protocols that we aim to analyze may use both encryption and digital signatures, it is more convenient to define the security of signatures and encryption when used simultaneously, in a multi-user environment. We develop a formal model for security that mixes definitional ideas from [13] (for digital signature schemes) and from [20] and [6] (for asymmetric encryption). Here, we only give an overview of the definition. The precise definition can be found in [11]. We consider an experiment parametrized by a digital signature scheme $\mathcal{DS}$, an asymmetric encryption scheme $\mathcal{AE}$, an adversary $\mathcal{A}$, a bit b and a security parameter η. In this experiment the adversary $\mathcal{A}$ has access to an *oracle* denoted $\mathcal{O}_{\mathcal{DS},\mathcal{AE}}(b, \eta)$. The adversary issues the following requests in any order and any number of times:

- creation of keys: the oracle generates (internally) keys for encryption, decryption, signing, and verifying and returns the public keys (*i.e.* keys for encryption and for verifying) to the adversary.
- signature request: the adversary can request signatures on any message it chooses, under any of the secret signing keys that has been generated. The oracle computes such a signature and returns it to the adversary.
- encryption requests: here the adversary submits a pair of messages (m_0, m_1), specifies an encryption key that has been generated and obtains from the oracle the encryption of m_b under that key.
- decryption requests: the adversary can require to see the decryption of any ciphertext of his choosing, provided that the ciphertext has not been obtained from the encryption oracle.

The goal of the adversary is to produce a valid signature on some message which it did not query to the oracle (*i.e.* break the signature scheme), or determine what is the selection bit b with probability significantly better than $1/2$ (*i.e.* break the encryption).

If for all p.p.t. adversaries either of the above events happens only with negligible probability[2] (in the security parameter), then we say that $\mathcal{DS}$ and $\mathcal{AE}$ are jointly secure. Although this is a new measure of security intended for analyzing security of encryption and that of signing when used simultaneously, it is easy to prove that it is implied by standard requirements on the individual primitives. More precisely, it is easy to show that if the digital signature scheme $\mathcal{DS}$ is existentially unforgeable under chosen-message attack [13] and if $\mathcal{AE}$ is secure in the sense of indistinguishability under chosen-ciphertext attacks (IND-CCA) then $\mathcal{DS}$ and $\mathcal{AE}$ are jointly secure.

3 Protocol Syntax

We consider protocols specified in a language similar to the one of Casrul [21] allowing parties to exchange messages built from identities and randomly generated nonces using

[2] A function is said to be negligible if it grows slower than the inverse of any polynomial.

public key encryption and digital signatures. Consider an algebraic signature Σ with the following sorts. A sort ID for agent identities, sorts SKey, VKey, EKey, DKey containing keys for signing, verifying, encryption, and decryption respectively. The algebraic signature also contains sorts Nonce, Label, Ciphertext, Signature, and Pair for respectively nonces, labels, ciphertexts, signatures, and pair. The sort Label is used in encryption and signatures to distinguish between different encryption/signature of the same plaintext. The sort Term is a supersort containing all other sorts, except SKey and DKey. There are nine operations: the four operations ek, dk, sk, vk are defined on the sort ID and return the encryption key, decryption key, signing key, and verification key associated to the input identity. The two operations ag and adv are defined on natural numbers and return labels: these labels are used to differentiate between different encryptions (and signatures) of the same plaintext, created by the honest agents or the adversary. We distinguish between labels for agents and for the adversary since they do not use the same randomness. The other operations that we consider are pairing, public key encryption, and signing with the following ranges and domains.

- $\langle \_ , \_ \rangle : \mathsf{Term} \times \mathsf{Term} \to \mathsf{Pair}$
- $\{\_\}^{\_}_{\_} : \mathsf{EKey} \times \mathsf{Term} \times \mathsf{Label} \to \mathsf{Ciphertext}$
- $[\_]^{\_}_{\_} : \mathsf{SKey} \times \mathsf{Term} \times \mathsf{Label} \to \mathsf{Signature}$

Protocols are specified using the algebra of terms constructed over the above signature from a set X of sorted variables. Specifically, $\mathsf{X} = \mathsf{X}.n \cup \mathsf{X}.a \cup \mathsf{X}.c \cup \mathsf{X}.s \cup \mathsf{X}.l$, where $\mathsf{X}.n, \mathsf{X}.a, \mathsf{X}.c, \mathsf{X}.s, \mathsf{X}.l$ are sets of variables of sort nonce, agent, ciphertext, signature, and labels respectively. Furthermore, $\mathsf{X}.a$ and $\mathsf{X}.n$ are as follows. If $k \in \mathbb{N}$ is some fixed constant representing the number of protocol participants, w.l.o.g. we fix the set of agent variables to be $\mathsf{X}.a = \{A_1, A_2, \ldots, A_k\}$, and partition the set of nonce variables, by the party that generates them. Formally: $\mathsf{X}.n = \cup_{A \in \mathsf{X}.a} \mathsf{X}_n(A)$ and $\mathsf{X}_n(A) = \{X_A^j \mid j \in \mathbb{N}\}$. This partition avoids to specify later, for each role, which variables stand for generated nonces and which variables stand for expected nonces.

The messages that are sent by participants are specified using terms in $\mathsf{T}_\Sigma(\mathsf{X})$, the free algebra generated by X over the signature Σ. The individual behavior of each protocol participant is defined by a *role* that describes a sequence of message receptions/transmissions. A k-party protocol is given by k such roles.

Definition 1 (Roles and protocols). *The set* Roles *of roles for protocol participants is defined by* $\mathsf{Roles} = ((\{\mathsf{init}\} \cup \mathsf{T}_\Sigma(\mathsf{X})) \times (\mathsf{T}_\Sigma(\mathsf{X}) \cup \{\mathsf{stop}\}))^*$.
A k-party protocol is a mapping $\Pi : [k] \to \mathsf{Roles}$, where $[k]$ denotes the set $\{1, 2, \ldots, k\}$.

We assume that a protocol specification is such that $\Pi(j) = ((l_1^j, r_1^j), (l_2^j, r_2^j), \ldots)$, the j'th role in the definition of the protocol being executed by player A_j. Each sequence $((l_1, r_1), (l_2, r_2), \ldots) \in \mathsf{Roles}$ specifies the messages to be sent/received by the party executing the role: at step i, the party expects to receive a message conforming to l_i and returns message r_i. We wish to emphasize however that terms l_i^j, r_i^j are not actual messages but specify how the message that is received and the message that is output should look like.

Example 1. The Needham-Schroeder-Lowe protocol [15] is specified as follows: there are two roles $\Pi(1)$ and $\Pi(2)$ corresponding to the sender's role and the receiver's role.

$$\begin{aligned} A \to B:\ & \{N_a, A\}_{\mathsf{ek}(B)} \\ B \to A:\ & \{N_a, N_b, B\}_{\mathsf{ek}(A)} \\ A \to B:\ & \{N_b\}_{\mathsf{ek}(B)} \end{aligned}$$

$$\Pi(1) = (\mathsf{init}, \{X^1_{A_1}, A_1\}^{\mathsf{ag}(1)}_{\mathsf{ek}(A_2)}), (\{X^1_{A_1}, X^1_{A_2}, A_2\}^{L}_{\mathsf{ek}(A_1)}, \{X^1_{A_2}\}^{\mathsf{ag}(1)}_{\mathsf{ek}(A_2)})$$
$$\Pi(2) = (\{X^1_{A_1}, A_1\}^{L_1}_{\mathsf{ek}(A_2)}, \{X^1_{A_1}, X^1_{A_2}, A_2\}^{\mathsf{ag}(1)}_{\mathsf{ek}(A_1)}), (\{X^1_{A_2}\}^{L_2}_{\mathsf{ek}(A_2)}, \mathsf{stop})$$

EXECUTABLE PROTOCOLS. Clearly, not all protocols written using the syntax above are meaningful. We only consider the class of *executable protocols*, *i.e.* protocols for each role can be implemented in an executable program, using only the local knowledge of the corresponding agent. This requires in particular that any sent message (corresponding to some r^j_k) is always deducible from the previously received messages (corresponding to $l^j_1, \ldots, l^j_k$). A precise definition may found in [11].

4 Execution Models

In this section we give a symbolic and a computational execution model for the protocols specified using the syntax defined in the previous section. In the symbolic model the honest parties and the adversary exchange elements of a certain term algebra; the adversary can compute its messages only following the standard Dolev-Yao restrictions. In the concrete execution model, the messages that are exchanged are bit-strings and the honest parties and the adversary are p.p.t. Turing machines.

4.1 Formal Execution Model

In the formal execution model, messages are terms of the free algebra T^f defined by:

$$\begin{aligned} T^f ::= {} & \mathbb{N} \mid a \mid \mathsf{ek}(a) \mid \mathsf{dk}(a) \mid \mathsf{sk}(a) \mid \mathsf{vk}(a) \mid n(a,j,s) && a \in \mathsf{ID}, j, s \in \mathbb{N} \\ & \langle T^f, T^f \rangle \mid \{T^f\}^{\mathsf{ag}(i)}_{\mathsf{ek}(a)} \mid \{T^f\}^{\mathsf{adv}(i)}_{\mathsf{ek}(a)} \mid [T^f]^{\mathsf{ag}(i)}_{\mathsf{sk}(a)} \mid [T^f]^{\mathsf{adv}(i)}_{\mathsf{sk}(a)} && a \in \mathsf{ID}, i \in \mathbb{N} \end{aligned}$$

If A is a variable, or constant of sort agent, we define its knowledge by $\mathbf{kn}(A) = \{\mathsf{dk}(A), \mathsf{sk}(A)\} \cup \mathsf{X}_n(A)$ *i.e.* an agent knows its secret decryption and signing key as well as the nonces it generates during the execution. The formal execution model is a state transition system. A *global state* of the system is given by (Sid, f, H) where H is a set of terms of T^f representing the messages sent on the network and f maintains the local states of all sessions ids Sid. Session identities are tuples of the form $(n, j, (a_1, a_2, \ldots, a_k)) \in (\mathbb{N} \times \mathbb{N} \times \mathsf{ID}^k)$, where $n \in \mathbb{N}$ identifies the session, the names $a_1, a_2, \ldots, a_k$ are the identities of the parties that are involved in the protocol and j is the index of the role that is executed in this session. Mathematically, f is a function $f : \mathsf{Sid} \to ([\mathsf{X} \to T^f] \times \mathbb{N} \times \mathbb{N})$, where $f(\mathsf{sid}) = (\sigma, i, p)$ is the local state of session sid. The function σ is a partial instantiation of the variables occurring in role $\Pi(i)$ and $p \in \mathbb{N}$ is the control point of the program. Three transitions are allowed.

$$\frac{}{S \vdash m}\, m \in S \qquad \frac{}{S \vdash b, \mathsf{ek}(b), \mathsf{vk}(b)}\, b \in \mathsf{X}.a \qquad \text{Initial knowledge}$$

$$\frac{S \vdash m_1 \quad S \vdash m_2}{S \vdash \langle m_1\, , m_2 \rangle} \qquad \frac{S \vdash \langle m_1\, , m_2 \rangle}{S \vdash m_i}\, i \in \{1,2\} \qquad \text{Pairing and unpairing}$$

$$\frac{S \vdash \mathsf{ek}(b) \quad S \vdash m}{S \vdash \{m\}^{\mathsf{adv}(i)}_{\mathsf{ek}(b)}}\, i \in \mathbb{N} \qquad \frac{S \vdash \{m\}^{l}_{\mathsf{ek}(b)} \quad S \vdash \mathsf{dk}(b)}{S \vdash m} \qquad \text{Encryption and decryption}$$

$$\frac{S \vdash \mathsf{sk}(b) \quad S \vdash m}{S \vdash [m]^{\mathsf{adv}(i)}_{\mathsf{sk}(b)}}\, i \in \mathbb{N} \qquad \frac{S \vdash [m]^{\mathsf{ag}(i)}_{\mathsf{sk}(b)}}{S \vdash [m]^{\mathsf{adv}(j)}_{\mathsf{sk}(b)}}\, i,j \in \mathbb{N} \qquad \frac{S \vdash [m]^{l}_{\mathsf{sk}(b)}}{S \vdash m} \qquad \text{Signature}$$

Fig. 1. Deduction rules for the formal adversary; here S is an arbitrary set of formal terms

- $(\mathsf{SId}, f, H) \xrightarrow{\mathbf{corrupt}(a_1,\ldots,a_l)} (\mathsf{SId}, f, \cup_{1\le j\le l}\mathbf{kn}(a_j) \cup H)$. The adversary corrupts parties by outputting a set of identities. He receives in return the secret keys corresponding to the identities. It happens only once at the beginning of the execution.
- The adversary can initiate new sessions: $(\mathsf{SId}, f, H) \xrightarrow{\mathbf{new}(i,a_1,\ldots,a_k)} (\mathsf{SId}', f', H')$ where H', f' and SId' are defined as follows. Let $s = |\mathsf{SId}| + 1$, be the session identifier of the new session, where $|\mathsf{SId}|$ denotes the cardinality of SId. H' is defined by $H' = H \cup \{(s, i, (a_1, \ldots, a_k))\}$ and $\mathsf{SId}' = \mathsf{SId} \cup \{(s, i, (a_1, \ldots, a_k))\}$. The function f' is defined as follows.
  - $f'(\mathsf{sid}) = f(\mathsf{sid})$ for every $\mathsf{sid} \in \mathsf{SId}$.
  - $f'(s, i, (a_1, \ldots, a_k)) = (\sigma, i, 1)$ where σ is a partial function $\sigma : \mathsf{X} \to T^f$ and:

$$\begin{cases} \sigma(A_j) & = a_j & 1 \le j \le k \\ \sigma(X^j_{A_i}) & = n(a_i, j, s) & j \in \mathbb{N} \end{cases}$$

  We recall that the principal executing the role $\Pi(i)$ is represented by A_i thus, in that role, every variable of the form $X^j_{A_i}$ represents a nonce generated by A_i.
- The adversary can send messages: $(\mathsf{SId}, f, H) \xrightarrow{\mathbf{send}(\mathsf{sid},m)} (\mathsf{SId}, f', H')$ where $\mathsf{sid} \in \mathsf{SId}$, $m \in T^f$, H', and f' are defined as follows. We define $f'(\mathsf{sid}') = f(\mathsf{sid}')$ for every $\mathsf{sid}' \neq \mathsf{sid}$. We denote $\Pi(j) = ((l^j_1, r^j_1), \ldots, (l^j_{k_j}, r^j_{k_j}))$. $f(\mathsf{sid}) = (\sigma, j, p)$ for some σ, j, p. There are two cases.
  - Either there exists a least general unifier θ of m and $l^j_p\sigma$. Then $f'(\mathsf{sid}) = (\sigma \cup \theta, i, p+1)$ and $H' = H \cup \{r^j_p\sigma\theta\}$.
  - Or we define $f'(\mathsf{sid}) = f(\mathsf{sid})$ and $H' = H$ (the state remains unchanged).

If we denote by $\mathsf{SID} = \mathbb{N} \times \mathbb{N} \times \mathsf{ID}^k$ the set of all sessions ids, the set of *symbolic execution traces* is $\mathsf{SymbTr} = \mathsf{SID} \times (\mathsf{SID} \to ([\mathsf{X} \to T^f] \times \mathbb{N} \times \mathbb{N})) \times 2^{T^f}$.

The adversary intercepts messages between honest participants and computes new messages using the deduction relation $\vdash$ defined in Figure 1. Intuitively, $S \vdash m$ means that the adversary is able to compute the message m from the set of messages S. All

deduction rules are rather standard with the exception of the last two; for these rules some explanations are in order. The next to last rule states that given a signature on some message m, the adversary can compute new signatures on the same message. The last rules states that the adversary can recover the corresponding message out of a given signature. Both rules are needed to obtain soundness. The rules reflect capabilities that do not contradict the standard computational security definition of digital signatures, and thus are available to computational adversaries.

Then, a symbolic execution trace $(\mathsf{SId}_1, f_1, H_1), \ldots, (\mathsf{SId}_n, f_n, H_n)$ is *valid* if the messages sent by the adversary can be computed by Dolev-Yao operations, *i.e.* if, whenever $(\mathsf{SId}_i, f_i, H_i) \xrightarrow{\mathbf{send}(s,m)} (\mathsf{SId}_{i+1}, f_{i+1}, H_{i+1})$, we have $H_i \vdash m$. Given a protocol Π, the set of valid symbolic execution traces is denoted by $\mathsf{Exec}^s(\Pi)$.

Example 2. Playing with the Needham-Schroeder-Lowe protocol described in Example 1, an adversary can corrupt an agent a_3, start a new session for the second role with players a_1, a_2 and send the message $\{n(a_3,1,1), a_1\}^{\mathsf{adv}(1)}_{\mathsf{ek}(a_2)}$ to the player of the second role. The corresponding valid trace execution is:

$$(\emptyset, f_1, \emptyset) \xrightarrow{\mathbf{corrupt}(a_3)} (\emptyset, f_1, \mathbf{kn}(a_3)) \xrightarrow{\mathbf{new}(2,a_1,a_2)} (\{\mathsf{sid}_1\}, f_2, \mathbf{kn}(a_3) \cup \{\mathsf{sid}_1\})$$
$$\xrightarrow{\mathbf{send}(\mathsf{sid}_1, \{n_3,a_1\}^{\mathsf{adv}(1)}_{\mathsf{ek}(a_2)})} \left(\{\mathsf{sid}_1\}, f_3, \mathbf{kn}(a_3) \cup \{\mathsf{sid}_1, \{n_3, n_2, a_2\}^{\mathsf{ag}(1)}_{\mathsf{ek}(a_1)}\}\right),$$

where $\mathsf{sid}_1 = (1, 2, (a_1, a_2))$, $n_2 = n(a_2, 1, 1)$, $n_3 = n(a_3, 1, 1)$, and f_2, f_3 are defined as follows: $f_2(\mathsf{sid}_1) = (\sigma_1, 2, 1)$, $f_3(\mathsf{sid}_1) = (\sigma_2, 2, 2)$ where $\sigma_1(A_1) = a_1$, $\sigma_1(A_2) = a_2$, $\sigma_1(X^1_{A_2}) = n_2$, and σ_2 extends σ_1 by $\sigma_2(X^1_{A_1}) = n_3$.

4.2 Concrete Execution Model

In a concrete execution, the messages that are exchanged are bit-strings and depend on a security parameter η (which is used for example to determine the length of random nonces). We denote by $\mathcal{C}^\eta$ the set of valid messages. We denote the subsets containing values for agent identities, nonces, encryption keys, verification keys, ciphertexts, signatures, and pairs by $C^\eta.a, \mathcal{C}^\eta.n, \mathcal{C}^\eta.e, \mathcal{C}^\eta.v, \mathcal{C}^\eta.c, \mathcal{C}^\eta.s, \mathcal{C}^\eta.p$ respectively. The implementation is such that each bit-string in $\mathcal{C}^\eta$ has a unique type which can be efficiently recovered by using the function $\mathsf{type} : \mathcal{C}^\eta \to \{a, n, e, v, c, s, p\}$. The operations are implemented as follows: we assume a PKI-like setting in which the public keys of parties (those for encryption and signature verification) are accessible to all parties. We model this situation by making available to all parties the (efficiently invertible and) publicly computable functions $\mathsf{vk} : \mathcal{C}^\eta.a \to \mathcal{C}^\eta.v$ and $\mathsf{ek} : \mathcal{C}^\eta.a \to \mathcal{C}^\eta.e$ which given an agent identity return its signature verification key and encryption key respectively. In the concrete implementation, encryption, and signing are implemented with encryption scheme $\mathcal{AE} = (\mathsf{K_e}, \mathsf{Enc}, \mathsf{Dec})$ and digital signature scheme $\mathcal{DS} = (\mathsf{K_s}, \mathsf{Sig}, \mathsf{Vf})$, which we fix throughout this section. Pairing is implemented by some standard (efficiently invertible) encoding function $\langle \cdot\,, \cdot \rangle : \mathcal{C}^\eta \times \mathcal{C}^\eta \to \mathcal{C}^\eta.p$.

The global state of the execution is a pair (f, SId), where f is used to represent the local state of each session, and SId represents the set of session ids.

Session ids are tuples $(n, i, (a_1, a_2, \ldots, a_l))$, where $n \in \mathbb{N}$ is a unique session identifier, i is the index of the role executed in this session and $a_1, a_2, \ldots a_k \in \mathcal{C}^\eta$ are the names of the agents involved in running this session. The state function $f : \mathsf{SId} \to [\mathsf{X} \to \mathcal{C}^\eta] \times \mathbb{N} \times \mathbb{N}$, given a session id sid returns $f(\mathsf{sid}) = (\sigma, i, p)$ where σ assigns values to the variables of the program executed in this session (see the discussion regarding the execution of individual roles), i is the index of the role executed in this session and p is the program counter that keeps track of the next step to be executed in this session.

We now discuss how the execution proceeds in this setting.

- At the beginning of the execution, the adversary corrupts a set of parties via a request $\mathbf{corrupt}(a_1, a_2, \ldots)$, where $a_1, a_2, \ldots \in \mathcal{C}^\eta.a$ are agent identities. As a result, the key generation algorithms for encryption and signing are executed, the public keys are published and the secrets keys are given to the adversary.
- The adversary initiates new sessions by issuing requests $\mathbf{new}(i, a_1, \ldots, a_k)$, with $i \in [k]$ and $a_1, \ldots, a_k \in \mathcal{C}^\eta.a$. In this case, cryptographic keys are generated for those agents which do not have such keys, the (public) encryption and verification keys are published and a new session is initiated: if (SId, f) is the state of the execution prior to the request the resulting state is (SId', f') with $\mathsf{SId}' = \mathsf{SId} \cup \{\mathsf{sid}\}$, $\mathsf{sid} = (|\mathsf{SId}| + 1, i, (a_1, \ldots, a_k))$, and f' defined as follows:
  - $f'(s) = s$ for $s \in \mathsf{SId}$ (*i.e.* the local states of previous sessions stay unchanged)
  - $f'(\mathsf{sid}) = (\sigma, i, 1)$ with $\sigma : \mathsf{X} \to \mathcal{C}^\eta$ defined as follows:

$$\begin{cases} \sigma(A_j) & = a_j & 1 \le j \le k \\ \sigma(X^j_{A_i}) & = n(a_i, j, s) \xleftarrow{\$} \mathcal{C}^\eta.n & j \in \mathbb{N} \end{cases}$$

  The local state of the new session is initialized by mapping agent variables to the names of the agents selected by the adversary, and selecting random values for the nonces generated by the party executing the role.
  In addition, for each term $\{t\}^l_{\mathsf{ek}(A_j)}$ and each term $[t]^l_{\mathsf{sk}(A_j)}$ that are sent (*i.e.* occurring within some r^j_i of $\Pi(i)$) we choose random coins $re^{\mathsf{sid}}(t, A_j, l)$ and $rs^{\mathsf{sid}}(t, A_j, l)$ respectively. These coins will later be used in randomizing the encryption and signing functions in the concrete implementation.
- The third kind of queries are message transmission queries $\mathbf{send}(\mathsf{sid}, m)$, with $\mathsf{sid} \in \mathsf{SId}$ and $m \in \mathcal{C}^\eta$ which are processed in two steps:
  First, the incoming message is parsed as an instantiation of the term l^p_i, where we let (σ, i, p) be the local state $f(\mathsf{sid})$ of session sid prior to the request. The parsing is done recursively, on the structure of l^p_i, and the final result is a mapping σ' assigning values in $\mathcal{C}^\eta$ to the variables occurring in l^p_i. To facilitate the parsing procedure, we assume that 1) from any valid ciphertext it is easy to recover the key used for encryption (which is public) and 2) from any valid signature, it is easy to recover the message that was signed and the verification key that needs to be used for verifying. Both these requirements can be easily achieved by tagging the signatures and the ciphertext with the appropriate information.

  In the second step, the local state of sid is updated and a protocol message is computed and returned to the adversary. If the parsing procedure fails at any point (the types of the term and of the bit-string do not match, or a ciphertext is invalid

etc) then the local state of sid remains unchanged. This is also the case if there exists some variable $X \in \mathsf{X}$ for which σ and σ' assign different values. Otherwise, the local store is updated to $\sigma = \sigma \cup \sigma'$ and the answer is computed by replacing each variable X in r_i^p with $\sigma(X)$ and replacing the encryptions and signatures with their computational counterparts, *i.e.* with the randomized functions Enc and Sig.

The execution model that we described above uses randomization: the adversary is probabilistic, and the honest parties use randomization for generating nonces, encryptions, and signatures. It can be shown that if the adversary A runs in polynomial-time, then the honest parties use a number of coins that is a polynomial in the security parameter. In the following, for a fixed adversary $\mathcal{A}$ we denote by $\{0,1\}^{p_{\mathcal{A}}(\eta)}$, resp. by $\{0,1\}^{g_{\mathcal{A}}(\eta)}$, the spaces from where the adversary, resp. the honest parties, draw the coins used in the execution. Notice that each pair of random coins $(R_{\mathcal{A}}, R_{\Pi}) \in \{0,1\}^{p_{\mathcal{A}}(\eta)} \times \{0,1\}^{g_{\mathcal{A}}(\eta)}$ determines a unique sequence of global states (f_1, SId_1), $(f_2, \mathsf{SId}_2), \ldots$, called the *concrete trace* determined by random coins $(R_{\Pi}, R_{\mathcal{A}})$ and which we denote by $\mathsf{Exec}_{\Pi(R_{\Pi}),\mathcal{A}(R_{\mathcal{A}})}(\eta)$. If the set of all possible session ids is $\mathcal{SId} = \mathbb{N} \times [k] \times (\mathcal{C}^{\eta}.a)^k$ then, we denote by ConcTr the set of all possible concrete traces: $\cup_{\eta}(\mathcal{SId} \times [\mathcal{SId} \to [\mathsf{X} \to \mathcal{C}^{\eta}])^*$.

5 Security Properties and Soundness Theorems

We are interested in two types of security properties. Integrity properties and secrecy properties. The former are quite general: for example, they encompass various forms of authentication (both for messages and entities). Our focus will be secrecy properties: we give formalizations for this kind of properties in both the formal and in the computational model, focusing on nonces. We then prove our second main result, a soundness theorem for secrecy of nonces.

5.1 Relating Symbolic and Concrete Traces

Concrete traces can be regarded as instantiations of formal traces via appropriate instantiations of the terms. More precisely, given a formal trace $t^s = (\mathsf{Sld}_1^s, f_1, H_1), \ldots, (\mathsf{Sld}_n^s, f_n, H_n)$, one can obtain a concrete execution trace $t^c = (\mathsf{Sld}_1^c, g_1), \ldots, (\mathsf{Sld}_n^c, g_n)$ on the following way. Once an injective function $c : T^f \to \mathcal{C}^{\eta}$ that maps terms to bitstrings is chosen, t^c is obtained by instantiating the local states: if $f_i(\mathsf{sid}) = (\sigma^{\mathsf{sid}}, i^{\mathsf{sid}}, p^{\mathsf{sid}})$ then $g_i(\mathsf{sid}) = (\tau^{\mathsf{sid}}, i^{\mathsf{sid}}, p^{\mathsf{sid}})$ where $\tau^{\mathsf{sid}} = c \circ \sigma^{\mathsf{sid}}$, and the session ids are unchanged: $\mathsf{Sld}_i^s = \mathsf{Sld}_i^c$. In that case, we say that t^c is a *concrete instantiation* of t^s (or alternatively t^s is a *symbolic representation* of t^c) and we write $t^s \preceq t^c$.

For $\mathsf{P} \subseteq \mathsf{SymbTr}$ we denote by $\mathsf{concrete}(\mathsf{P})$ the set $\{t^c \mid \exists t^s \in \mathsf{P}\ t^s \preceq t^c\}$ of all concrete instantiations of symbolic traces in P.

Technically, the following lemma is at the core of our results. It states that with overwhelming probability, the concrete executions traces of a protocol are instantiations of *valid* symbolic execution traces.

Lemma 1. *Let Π be an executable protocol. If in the concrete implementation the schemes $\mathcal{AE}$ and $\mathcal{DS}$ are jointly secure then for any p.p.t. algorithm $\mathcal{A}$*

$$\Pr\left[\exists t^s \in \mathsf{Exec}^s(\Pi) \mid t^s \preceq \mathsf{Exec}^c_{\Pi(R_{\Pi}),\mathcal{A}(R_{\mathcal{A}})}(\eta)\right] \geq 1 - \nu_{\mathcal{A}}(\eta)$$

where the probability is over the choice $(R_\Pi, R_\mathcal{A}) \stackrel{\$}{\leftarrow} \{0,1\}^{p_\mathcal{A}(\eta)} \times \{0,1\}^{g_\mathcal{A}(\eta)}$ *and* $\nu_\mathcal{A}(\cdot)$ *is some negligible function.*

Proof (Overview). Due to space constraints we only sketch the main aspects of the proof (details may be found in [11]).

The proof works in two steps. First, we explain how each concrete execution trace $\mathsf{Exec}^c_{\Pi(R_\Pi),\mathcal{A}(R_\mathcal{A})}$ determines a unique symbolic trace t^s. We construct t^s by tracing the queries made by the concrete adversary $\mathcal{A}$ and translating them into symbolic queries. Specifically, we map each bit-string m occurring in the execution to a symbolic term $\mathsf{c}(m)$ as follows. Agent identities, cryptographic keys and random nonces (which are quantities that are uniquely determined by R_Π) are canonically mapped to symbolic representations: for example the bit-string representing the decryption key of party a_i is mapped to $\mathsf{sk}(a_i)$. The rest of the messages are interpreted as they occur: each message m sent by the adversary is parsed (notice that all keys that are needed are already known) and its symbolic interpretation is obtained by replacing all occurring basic values (keys, nonces, identities) with their symbolic interpretation, and then replacing the concrete operations with their symbolic counterparts.

In the second step of the proof, we show that with overwhelming probability over the choice of $(R_\Pi, R_\mathcal{A})$, the trace t^s obtained as explained above is a valid execution trace. We prove this statement by contradiction: given an adversary $\mathcal{A}$ we construct three adversaries $\mathcal{B}_1, \mathcal{B}_2$ and $\mathcal{B}_3$ such that if with non-negligible probability the symbolic trace associated to the execution of $\mathcal{A}$ is not a valid Dolev-Yao trace, then at least one of the three adversaries breaks the joint security of $\mathcal{DS}$ and $\mathcal{AE}$.

The idea behind the construction of these adversaries is to execute adversary $\mathcal{A}$ as a subroutine, and use access to the oracle $\mathcal{O}_{\mathcal{DS},\mathcal{AE}}$ (to which each of the three adversaries has access) to simulate the execution of the protocol on behalf of the honest parties. Then, we show that, using the invalid query made by $\mathcal{A}$, adversary $\mathcal{B}_i$ (with $i = 1, 2, 3$) can break either the encryption, or the signing scheme, each of the three adversaries exploiting one of the following three possibilities. Adversary $\mathcal{B}_1$ is based on the assumption that the invalid query of adversary $\mathcal{A}$ contains a signature $[t]_{\mathsf{sk}(a_i)}$ under the secret key of an honest party a_i which was never sent prior in the execution. This essentially means that the corresponding concrete term is a signature forgery, and adversary $\mathcal{B}_1$ simply outputs it. Adversaries $\mathcal{B}_2$ and $\mathcal{B}_3$ correspond to the case where the adversary $\mathcal{A}$ outputs the encryption of some term t such that neither t nor the encryption can be computed by the adversary from the previous messages using only Dolev-Yao operations. In this case we show how to use the adversary $\mathcal{A}$ to determine some secret which he should not have been able to compute. This secret is a random nonce generated by some honest party in the case of adversary $\mathcal{B}_2$ and a signature also generated by an honest party, in the case of adversary $\mathcal{B}_3$. Moreover, the adversaries $\mathcal{B}_1, \mathcal{B}_2$, and $\mathcal{B}_3$ that we construct are such that their sample space partition the sample space of the experiment in which adversary $\mathcal{A}$ is executed. Therefore, if with non-negligible probability the adversary $\mathcal{A}$ has an invalid symbolic execution trace, then with non-negligible probability at least one of the adversaries $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$ breaks the joint security of $\mathcal{DS}$ and $\mathcal{AE}$ which contradicts the hypothesis of the theorem. □

5.2 Trace Properties

For both the symbolic and the computational execution model, trace properties are predicates on the global execution traces. The definition of security (i.e. when a protocol satisfies a given trace property) differs between the symbolic and the computational model. We now give these definitions and give our main result: a soundness theorem for proofs of trace properties.

SYMBOLIC TRACE PROPERTIES. A symbolic trace property is a predicate on (or alternatively a subset of) the set SymbTr. We say that protocol Π satisfies the symbolic trace property $\mathsf{P}^s \subseteq \mathsf{SymbTr}$ and we write $\Pi \models^s \mathsf{P}^s$, if all valid execution traces satisfy P^s, *i.e.* $\mathsf{Exec}^s(\Pi) \subseteq \mathsf{P}^s$.

Various definitions of authentication may be expressed using such trace properties. Informally, a trace of a protocol is a "good" mutual entity authentication trace, if for any two identities a and b, if a (playing the second role of the protocol) has finished a session of the protocol with intended partner b (playing the first role of the protocol), then b has finished a session with intended partner a. Using this characterization, we say that a protocol is a secure authentication protocol if all its traces are good. Depending on which notion of authentication we consider, we may also require that for any session where a terminates, there exists exactly one corresponding session where b terminates and b must have finished before a.

COMPUTATIONAL TRACE PROPERTIES. A computational trace property is a predicate on ConcTr. We say that protocol Π satisfies the concrete security property $\mathsf{P}^c \subseteq \mathsf{ConcTr}$, and we write $\Pi \models^c \mathsf{P}^c$ if its execution traces satisfy P^c with overwhelming probability over the coins used in the execution, *i.e.* for every p.p.t. adversary $\mathcal{A}$, the probability $\Pr\left[\mathsf{Exec}_{\Pi(R_\Pi),\mathcal{A}(R_\mathcal{A})}(\eta) \notin \mathsf{P}^c\right]$ is negligible as a function of η. The probability is taken over the choice $(R_\Pi, R_\mathcal{A}) \stackrel{\$}{\leftarrow} \{0,1\}^{p_\mathcal{A}(\eta)} \times \{0,1\}^{q_\mathcal{A}(\eta)}$.

For mutual authentication, good traces are those satisfying the predicate we sketched for the symbolic model, but the definition of security for protocols is specific to the computational setting: it asks from protocol to have good traces with overwhelming probability. It thus allows for "bad" runs, but only with negligible probability.

One of our contributions is the following soundness theorem for trace properties.

Theorem 1. *Let Π be an executable protocol, $\mathsf{P}^s \subseteq \mathsf{SymbTr}$ be an arbitrary symbolic trace property and $\mathsf{P}^c \subseteq \mathsf{ConcTr}$ be a computational security property such that $\mathsf{concrete}(\mathsf{P}^s) \subseteq \mathsf{P}^c$. Then $\Pi \models^s \mathsf{P}^s$ implies $\Pi \models^c \mathsf{P}^c$.*

Proof. Let $\mathcal{A}$ be an arbitrary p.p.t. adversary for Π. We have

$$\Pr\left[\mathsf{Exec}_{\Pi(R_\Pi),\mathcal{A}(R_\mathcal{A})}(\eta) \in \mathsf{P}^c\right] \geq \Pr\left[\mathsf{Exec}_{\Pi(R_\Pi),\mathcal{A}(R_\mathcal{A})}(\eta) \in \mathsf{P}^c \wedge \exists t \in \mathsf{Exec}^s(\Pi), t \preceq \mathsf{Exec}_{\Pi(R_\Pi),\mathcal{A}(R_\mathcal{A})}(\eta)\right].$$

Since $\Pi \models^s \mathsf{P}^s$ and $\mathsf{concrete}(\mathsf{P}^s) \subseteq \mathsf{P}^c$ it follows that:

$$\Pr\left[\mathsf{Exec}_{\Pi(R_\Pi),\mathcal{A}(R_\mathcal{A})}(\eta) \in \mathsf{P}^c\right] \geq \Pr\left[\exists t \in \mathsf{Exec}^s(\Pi) \mid t \preceq \mathsf{Exec}_{\Pi(R_\Pi),\mathcal{A}(R_\mathcal{A})}(\eta)\right].$$

By Lemma 1, we deduce $\Pr\left[\mathsf{Exec}_{\Pi(R_\Pi),\mathcal{A}(R_\mathcal{A})}(\eta) \notin \mathsf{P}^c\right] \leq \nu_\mathcal{A}(\eta)$, *i.e.* $\Pi \models^c \Pi^c$. □

5.3 Secrecy Properties

In the symbolic model, secrecy is naturally expressed as a trace property: a message is secret if it cannot be derived by the adversary. In the computational model however, typical definitions are much stronger and they usually say that an attacker cannot obtain not only the secret, but also *any* partial information about the secret. In this section we give symbolic and computational definitions for the secrecy of nonces used in a protocol and prove a soundness theorem: if a nonce is deemed secret using symbolic techniques, then the nonce is secret with respect to the stronger, computational definition.

We concentrate on the case of secrecy of nonces since there is no canonical definition for secrecy of composed messages in the computational world. In addition, as noticed in [12], the definition of secrecy for keys for example has to be weaker than indistinguishability as soon as the encrypted messages contain some redundancy. However, if the keys are not used, then security of keys is similar to the security of nonces and our results yield meaningful results for symmetric key exchange.

SECRECY IN THE SYMBOLIC MODEL. Let Π be an arbitrary k-party protocol. We say that Π guarantees the *secrecy* of the nonce $X^j_{A_i} \in \mathsf{X}_n(A_i)$ if in all possible executions, each instantiation of this variable remains unknown to the adversary. Formally, this means that for every valid trace $(\mathsf{sid}_1, f_1, H_1), \ldots, (\mathsf{sid}_n, f_n, H_n)$ of the protocol, for every session id $\mathsf{sid}_p = (r, i, (a_1, \ldots, a_k))$ where $a_1, \ldots, a_k$ are honest agents (*i.e.* none of them appears in the **corrupt** query), we have $H_n \not\vdash n(a_i, j, r)$. If this is the case, we write $\Pi \models^s \mathsf{SecNonce}(i, j)$.

SECRECY IN THE COMPUTATIONAL MODEL. We define the secrecy of the nonce $X^j_{A_i}$ in protocol Π using an experiment $\mathbf{Exp}^{\mathrm{sec\_}b}_{\mathsf{Exec}_{\Pi,\mathcal{A}}}(i, j)(\eta)$ that we describe below. The experiment is parametrized by a bit b and involves an adversary $\mathcal{A}$. The input to the experiment is a security parameter η. It starts by generating two random nonces n_0 and n_1 in $\mathcal{C}^\eta.n$. Then the adversary $\mathcal{A}$ starts interacting with the protocol Π as in the experiment $\mathsf{Exec}_{\Pi,\mathcal{A}}(\eta)$: it generates new sessions, sends messages and receives messages to and from these sessions (as prescribed by the protocol). At some point in the execution the adversary initiates a session s in which the role of A_i is executed, and declares this session under attack. Then, in this session the variable $X^j_{A_i}$ is instantiated with n_b (*i.e.* one of the two nonces chosen in the beginning of the experiment, the selection being made according to the bit b). The rest of the execution is exactly as in $\mathsf{Exec}_{\Pi,\mathcal{A}}$. In the end, the adversary is given n_0 and n_1 and outputs a guess d, which is also the result of the experiment. We define the advantage of the adversary $\mathcal{A}$ by:

$$\mathbf{Adv}^{\mathrm{sec}}_{\mathsf{Exec}_{\Pi,\mathcal{A}}}(i, j)(\eta) = \Pr\left[\mathbf{Exp}^{\mathrm{sec\_}1}_{\mathsf{Exec}_{\Pi,\mathcal{A}}}(i, j)(\eta) = 1\right] - \Pr\left[\mathbf{Exp}^{\mathrm{sec\_}0}_{\mathsf{Exec}_{\Pi,\mathcal{A}}}(i, j)(\eta) = 1\right]$$

We say that nonce $X^j_{A_i}$ is computationally secret in protocol Π, and we write $\Pi \models^{\mathsf{c}} \mathsf{SecNonce}(i, j)$ if for every p.p.t. adversary $\mathcal{A}$ its advantage is negligible.

Our second main result, captured by the following theorem, states that if a nonce is secret in the symbolic model then it is also secret in the computational model.

Theorem 2. *Let Π be an executable protocol. If the schemes $\mathcal{DS}$ and $\mathcal{AE}$ are jointly secure, then: $\Pi \models^f \mathsf{SecNonce}(i, j)$ implies $\Pi \models^c \mathsf{SecNonce}(i, j)$.*

6 Automated Proof Using Casrul

In this section we describe the automated tool Casrul [9] and discuss the implications of our results for the proofs done with Casrul.

Casrul is a system for automated verification of cryptographic protocols, developed by the Cassis group at Loria (France) available at

`http://www.loria.fr/equipes/cassis/softwares/casrul/`

It translates a protocol given in common abstract syntax into a rewrite system. The rewrite system is processed using a first order theorem prover for equational logic for the automated detection of flaws. We note that Casrul does not allow the use of signatures and labels yet. Nevertheless, both its syntax and semantics coincide with ours for *public key protocols*, *i.e.* protocols that only use pairing and asymmetric encryption, but without using labels. We believe that both labels and signatures could be easily added in Casrul.

AUTOMATED PROOF FOR COMPUTATIONAL SECURITY USING CASRUL. Casrul can be used to prove three particular types of properties: entity authentication, authentication on data and data secrecy. Here, we discuss the implications of these proofs with respect to the computational model.

The syntax of Casrul does not yet allow the use of labels for encryption. However, it can be shown that for the security properties that are typically proved with Casrul, proofs in the execution model without labels are sound w.r.t. the model where labels are used. Thus, thanks to Theorem 1, Casrul proofs of the security with respect to these properties have a clear computational interpretation. For example, the Casrul proof that the Needham-Schroeder-Lowe [15] protocol is a secure mutual authentication protocol (file `NSPK_LOWE3.hlpsl`) implies the same property, but in the computational model.

Similarly, Casrul proofs of nonce secrecy imply, via Theorem 2, the strong, computational secrecy notion that we gave in Section 5.3. For example, Casrul enables to prove the computational secrecy of nonces used in the corrected Needham-Schroeder-Lowe protocol [15] (file `NSPK_LOWE2.hlpsl`) and in the SPLICE protocol [22] (file `SPLICE2.hlpsl`).

Note that Casrul works only with a finite number of sessions, thus proofs in the computational model are obtained only for that fixed number of sessions. Nevertheless, since our proofs consider adversaries that create an unbounded of sessions, we could also obtain proofs of computational security properties by using tools dedicated to an unbounded number of sessions like Hermes [8] or Securify [10]. This would require to first prove that protocols secure in the symbolic models of Securify or Hermes are also secure in our symbolic model. We believe this to be true since their symbolic models are very similar to ours. We did not use these tools for our proofs since they only provide automatic proofs of secrecy. Automated proofs of other security properties like authentication are still under development.

Acknowledgements. We would like to thank Martin Abadi for enlightening discussions and advice, Daniele Micciancio for useful suggestions and to the anonymous referees for their helpful remarks.

References

1. M. Abadi. Taming the adversary. In *Proc. of Crypto'00*, 2000.
2. M. Abadi and A. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proc. of the 4th Conf. on Computer and Communications Security*, pages 36–47. ACM Press, 1997.
3. M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.
4. M. Backes. Personal communication.
5. M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations (extended abstract). In *Proc. of 10th ACM Conference on Computer and Communications Security (CCS'05)*, pages 220 – 230, 2003.
6. M. Bellare, A. Boldyreva, and S. Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *Proc. of Eurocrypt'00*, volume 1807 of *LNCS*, pages 259–274, 2000.
7. B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proc. of the 14th CSFW*, June 2001.
8. L. Bozga, Y. Lakhnech, and M. Perin. An automatic tool for the verification of secrecy in security protocols. In *15th Int. Conference on Computer Aided Verification (CAV 2003)*, volume 2725 of *LNCS*, pages 219–222. Springer, July 2003.
9. Y. Chevalier and L. Vigneron. A tool for lazy verification of security protocols. In *Proc. of the 16th Conf. on Automated Software Engineering (ASE-2001)*. IEEE CS Press, 2001.
10. V. Cortier. *A guide for Securify*. RNTL EVA project, Report n. 13, December 2003.
11. V. Cortier and B. Warinschi. Computationally sound, automated proofs for security protocols. Research Report RR-5341, INRIA, October 2004.
12. D.H.Phan and D. Pointcheval. Une comparaison entre deux méthodes de preuve de sécurité. In *Proc. of RIVF*, pages 105–110, 2003. In French.
13. S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal of Computing*, 17(2):281–308, April 1988.
14. P. Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *Proc. of 2004 IEEE Symposium on Security and Privacy*, pages 71–85, 2004.
15. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, volume 1055 of *LNCS*, pages 147–166. Springer-Verlag, March 1996.
16. G. Lowe. Casper: A compiler for the analysis of security protocols. In *Proc. of 10th CSFW'97*. IEEE Computer Society Press, 1997.
17. D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Theory of Cryptography Conference (TCC 2004)*, pages 133–151, Cambridge, MA, USA, February 2004. Springer-Verlag.
18. J. Mitchell, A. Ramanathan, A. Scedrov, and V. Teague. A probabilistic polynomial-time calculus for analysis of cryptographic protocols. *Electronic Notes in Theoretical Computer Science*, 45, 2001.
19. L. Paulson. Mechanized proofs for a recursive authentication protocol. In *Proc. of the 10th CSFW'97*, pages 84–95. IEEE Computer Society Press, 1997.
20. C. Rackoff and D. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *CRYPTO'91*, pages 433–444, 1992.
21. M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *Proc. of the 14th CSFW'01*, pages 174–190. IEEE Computer Society Press, 2001.
22. S. Yamaguchi, K. Okayama, and H. Miyahara. The design and implementation of an authentication system for the wide area distributed environment. *IEICE Transactions on Information and Systems*, November 1991.