Computationally Sound, Automated Proofs for Security Protocols

Véronique Cortier, Bogdan Warinschi

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Computationally Sound, Automated Proofs for Security Protocols

Véronique Cortier and Bogdan Warinschi

N° 5341

Octobre 2004

Thème ?

ISSN 0249-6399 ISRN INRIA/RR--5341--FR+ENG

Computationally Sound, Automated Proofs for Security Protocols

Véronique Cortier* and Bogdan Warinschi

Thème ? —
Projet Cassis

Rapport de recherche n° 5341 — Octobre 2004 — 23 pages

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.

Key-words: security protocols, computational security, cryptography, secrecy, automated proof.

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

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy (France)
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

Preuves automatiques des protocoles de sécurité, correction vis-à-vis du modèle probabiliste

Résumé : Depuis les années 1980, deux approches ont été développées pour analyser les protocoles de sécurité. L'une de ces approches repose sur un modèle probabiliste qui prend en compte des aspects de complexité et de probabilité. Cette approche permet de traiter une notion forte de sécurité, garantissant la sécurité contre n'importe quelle attaque probabiliste polynomiale. L'autre approche repose sur un modèle d'exécutions symbolique dans lequel les primitives cryptographiques sont considérées comme des boites noires. Depuis les premiers travaux de Dolev-Yao, on a montré que cette dernière approche permet des preuves beaucoup plus simples et souvent automatiques. Cependant, les garanties offertes par cette approche sont souvent peu claires.

Dans ce papier, nous montrons qu'il est possible d'obtenir le meilleur des deux mondes : des preuves complètement automatiques et des garanties fortes et claires de sécurité. Plus précisément, pour des protocoles utilisant des signatures et du chiffrement asymétrique, nous avons établi que les preuves symboliques de secret et d'authentification étaient correctes vis-à-vis du modèle probabiliste. Le principal apport de cet article est le traitement des propriétés de sécurité pour lesquelles nous obtenons le premier résultat de correction dans le cas d'un adversaire actif. La vérification de nos propriétés d'authentification et de secret est faite à l'aide de Casrul, un outil complètement automatique.

Mots-clés : protocoles de sécurité, modèles probabilistes, cryptographie, secret, preuves automatiques.

Computationally Sound, Automated Proofs for Security Protocols

Véronique Cortier[1] and Bogdan Warinschi[2]

[1] Loria, CNRS, Nancy, France
[2] Computer Science Department, University of California at Santa Cruz, USA

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 cryptographic 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. 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. Two important implications stem from these features: 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 a very 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 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 [13] and theorem provers [16]. Unfortunately,

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

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 linking the two approaches via *computational soundness* theorems for symbolic analysis [3, 15, 5, 14]. Potentially, justifying symbolic proofs with respect to standard computational models has 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 demonstrate for the first time that *fully automated* security proofs with strong computational implications are possible. Our road-map is the following. First, we give a language for writing protocols that use random nonces, digital signatures and public-key encryption. We then give two kinds of executions for protocols, each performed in the presence of a powerful *active* adversary that controls and potentially tampers with the communication between an unbounded number of sessions of the protocol. 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). We note that many important integrity property such as entity and message authentication can be written as trace properties.

Our second main result concerns soundness of secrecy proofs. What makes this issue more challenging is that, unlike for trace properties, the formalization of secrecy properties is very different between the two models. On the one hand, in the symbolic model a message is said to be secret if the adversary cannot derive it[3]. On the other hand, the usual formalizations for secrecy in the computational model seem much stronger: a message is secret if it is impossible for the adversary not only to compute it, but also to compute *any partial information* about the message. Nevertheless, in the case of nonces we are able to prove a soundness theorem stating that symbolic secrecy implies computational secrecy.

We provide a computational justification for the proofs carried out using Casrul. We want to mention, however, that we have also briefly considered other automatic tools, such as Proverif, Casper and Securify. We strongly believe that soundness results similar to ours can be obtained for these tools. Our choice was determined by our familiarity with Casrul, one of us being a close collaborator of its developers.

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]. Out of the soundness results that have been published since, we only mention those 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 [11] who shows soundness of

[3] 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.

confidentiality properties for symmetric encryption in a model with a priorly fixed number of sessions. A soundness result for trace properties was proved by Micciancio and Warinschi [14] 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 introduce 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 the 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 section we briefly recall the syntax of digital signature schemes and that of public-key encryption schemes and introduce a notion of security regarding their joint use in cryptographic protocols.

DIGITAL SIGNATURES, PUBLIC-KEY ENCRYPTION AND THEIR SECURITY. In this paper we will use a generic digital signature scheme $\mathcal{DS} = (\mathsf{K_s}, \mathsf{Sig}, \mathsf{Vf})$ and a generic public-key encryption scheme $\mathcal{AE} = (\mathsf{K_e}, \mathsf{Enc}, \mathsf{Dec})$. A description of the algorithms and of their functionality can be found in Appendix A.

Traditionally, security is defined for each individual primitive. Since the protocols that we analyze in this paper 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 [10] (for digital signature schemes) and from [17] and [6] (for asymmetric encryption). The precise definition can be found in Appendix A. Here we only give an overview. We associate to a digital signature scheme $\mathcal{DS}$, an asymmetric encryption scheme $\mathcal{AE}$, an adversary $\mathcal{A}$, a bit b and a security parameter η an experiment. 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 happen only with negligible probability[4] (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 [10] 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*. We emphasize that both these security notions are standard in the computational setting.

3 Protocol Syntax

We consider protocols specified in a language similar to the one of [18] allowing parties to exchange messages built from identities and randomly generated nonces using public key encryption and digital signatures. We consider an algebraic signature with the following sorts. A sort ID for agent identities, sorts $\mathsf{SKey}, \mathsf{VKey}, \mathsf{EKey}, \mathsf{DKey}$ 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. There are nine operations: the four operations $\mathsf{ek}, \mathsf{dk}, \mathsf{sk}, \mathsf{vk}$ 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 number and return labels for agents and 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.

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

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 $\mathsf{X}_n(A) = \{X_A^j \mid j \in \mathbb{N}\}$.

ROLES AND PROTOCOLS. 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 pro-

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

tocol participant is defined by a *role* describing a sequence of message receptions/transmissions, and a k-party protocol is given by k such roles.

Definition 1 (Roles and protocols). *We define the set* Roles *of roles for protocol participants by* $\mathsf{Roles} = ((\{\mathsf{init}\} \cup \mathsf{T}_\Sigma(\mathsf{X})) \times (\mathsf{T}_\Sigma(\mathsf{X}) \cup \{\mathsf{stop}\}))^*$.
An 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. Notice that 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 conformed 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 [12] 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* defined by using two important ingredients. First is the knowledge that a principal has. 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. Second, we specify what is the set of messages that an agent can compute during the protocol execution by using the deduction relation $\vdash_a$ defined in Figure 1. Intuitively, a protocol is executable if it can be implemented. This requires in particular that any sent message (corresponding to some r_k^j) is always deducible from the previously received messages (corresponding to $l_1^j, \ldots, l_k^j$). It also requires the agents are effectively able to perform the equality tests implicitly defined by the repetitions of variables in the $l_1^j, \ldots, l_k^j$. A precise definition may found in Appendix B.

4 Execution Models

We consider two types of executions for protocols. In the symbolic setting 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.

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

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

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

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

Fig. 1. Deduction rules for agents.

4.1 Formal Execution Model

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

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

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 ids 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, $a_1, a_2, \ldots, a_k$ 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 of the role $\Pi(i)$ and $p \in \mathbb{N}$ is the control point of the program. Three transitions are allowed.

- $(\mathsf{SId}, f, H) \xrightarrow{\mathbf{corrupt}(a_1,\ldots,a_l)} (\mathsf{SId}, f, \cup_{1 \le j \le l} \mathbf{kn}(a_j) \cup H)$. 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:

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

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 $\mathsf{sid} \in \mathsf{SId}$, $m \in T^f$, and H' and f' are defined as follows. We define $f'(\mathsf{sid}') = f(\mathsf{sid}')$ for every $\mathsf{sid}' \in \mathsf{SId\_}\{\mathsf{sid}\}$. 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 substitution θ such that $m = l^j_p\sigma\theta$. 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 $\mathsf{SymbTr} = \mathsf{SID} \times (\mathsf{SID} \to ([\mathsf{X} \to T^f] \times \mathbb{N} \times \mathbb{N})) \times 2^{T^f}$.

We now rule out the execution traces which are invalid, in the sense that the adversary manages to perform some computation which he should not have been able to do, like for instance performing decryptions with a key which he does not possess. In fact, the deductions that an adversary can make, denoted by $\vdash$ are the same as for the honest parties, given in Figure 1 (where we replace $\vdash_a$ by $\vdash$) with three modifications. First, we allow the adversary to compute the message that is being signed. The reason for this addition is that concrete implementation of the signature scheme may reveal the entire message. Therefore we add the rule:

$$\frac{S \vdash [m]^l_{\mathsf{sk}(b)}}{S \vdash m}.$$

In addition, the encryption and signing rules have to be replaced by:

$$\frac{S \vdash_a \mathsf{ek}(b) \quad S \vdash_a m}{S \vdash_a \{m\}^{\mathsf{adv}(i)}_{\mathsf{ek}(b)}}\, i \in \mathbb{N} \qquad \frac{S \vdash_a \mathsf{sk}(b) \quad S \vdash_a m}{S \vdash_a [m]^{\mathsf{adv}(i)}_{\mathsf{sk}(b)}}\, i \in \mathbb{N}$$

Definition 2. *A symbolic execution trace* $(\mathsf{SId}_1, f_1, H_1) \ldots (\mathsf{SId}_n, f_n, H_n)$ *is* valid *if*

– $H_1 = \mathsf{SId}_1 = \emptyset$, $(\mathsf{SId}_1, f_1, H_1) \to (\mathsf{SId}_2, f_2, H_2)$ *for one of the three transitions described above and for every* $1 \le i \le n$, $(\mathsf{SId}_i, f_i, H_i) \to (\mathsf{SId}_{i+1}, f_{i+1}, H_{i+1})$ *for one of the two last transitions described above;*

– *and 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 $\mathsf{sid}_1 = (1, 1, (a_1, a_2))$, $n_2 = n(a_2, 1, 1)$, $n_3 = n(a_3, 1, 1)$ 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 $\sigma_2(X^1_{A_1}) = n_3$.

The following lemma gives a useful characterization of non-valid adversaries. The proof can be found in Appendix C.

Lemma 1. *Let S be a set of messages and m be a message. If $S \not\vdash m$ then:*

1. *either there exists a subterm $[t]_k$ of m which is not a subterm of terms in S,*
2. *or there exists a subterm t of m, i.e. $m|_p = t$ for some path p, such that for every path $p' \leq p$, $S \not\vdash m|_{p'}$, and t appears under an encryption in S, i.e. there exist a term $m' \in S$ and contexts C and C' such that $m' = C[\{C'[t]\}_k]$ with $S \not\vdash k^{-1}$.*

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 possible 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): SId is the set of session ids and f maintains the local state of each session. 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.

Let us 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 $\sigma : \mathsf{X} \to \mathcal{C}^\eta$ is defined as follows:

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

  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 $[t]^l_{\mathsf{sk}(A_i)}$ 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 $rs^{\mathsf{sid}}(t, A_i, l)$ 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.
  For instance, if l^p_i is $\{t\}^l_{\mathsf{ek}(A_i)}$, the incoming message m is parsed as follows: if the message is not a ciphertext then the parsing procedure ends; otherwise, the public key pk used for producing m is recovered from m, the identity a to which the key corresponds is computed as $\mathsf{ek}^{-1}(pk)$ and the assignment $[A_i \mapsto a]$ is added to σ'. Next, the decryption key corresponding to pk is computed as $\mathsf{sk}(a)$[5] and the message m is decrypted with this key. Let m' be the result of the decryption procedure. Then, the parsing procedure is recursively called with inputs t and m'. For the case of signatures, the parsing procedure also verifies the validity of the signature (recall that both the message and the verification key can be recovered from the signature).
  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

[5] Notice that this can be done only if and only if $a = a_i$

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, Sld_1), $(f_2, \mathsf{Sld}_2), \ldots$, 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 $\mathcal{SI}d = \mathbb{N} \times [k] \times (\mathcal{C}^{\eta}.a)^k$ then, we denote by ConcTr the set of all possible concrete traces: $\cup_{\eta}(\mathcal{SI}d \times [\mathcal{SI}d \to [\mathsf{X} \to \mathcal{C}^{\eta}])^*$.

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 renamings of the variables. More precisely, let

$$t^s = (\mathsf{Sld}_1^s, f_1, H_1), \ldots, (\mathsf{Sld}_n^s, f_n, H_n) \quad \text{and}$$

$$t^c = (\mathsf{Sld}_1^c, g_1), \ldots, (\mathsf{Sld}_n^c, g_n)$$

be a symbolic and a concrete execution trace. Trace t^c is a *concrete instantiation* of t^s (or alternatively t^s is a *symbolic representation* of t^c) and we write $t^s \preceq t^c$ if there exists an injective function $c : T^f \to \mathcal{C}^{\eta}$ such that for every $i \in [n]$ it holds that $\mathsf{Sld}_i^s = \mathsf{Sld}_i^c$ and for every $\mathsf{sid} \in \mathsf{Sld}_i^s$ if $f_i(\mathsf{sid}) = (\sigma^{\mathsf{sid}}, i^{\mathsf{sid}}, p^{\mathsf{sid}})$ and $g_i(\mathsf{sid}) = (\tau^{\mathsf{sid}}, j^{\mathsf{sid}}, q^{\mathsf{sid}})$ then $\tau^{\mathsf{sid}} = c \circ \sigma^{\mathsf{sid}}$, $i^{\mathsf{sid}} = j^{\mathsf{sid}}$ and $p^{\mathsf{sid}} = q^{\mathsf{sid}}$.

For $\mathsf{P} \subseteq \mathsf{SymbTr}$ we denote by $\mathsf{c}(\mathsf{P})$ 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 2. *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 the details of the proof can be found in Appendix D, and here we only sketch its main aspects.

The proof works in two steps. First, we explain how each concrete execution trace of a protocol $\mathsf{Exec}^c_{\Pi(R_\Pi),\mathcal{A}(R_\mathcal{A})}$ determines a unique symbolic trace t^s. We construct t^s by tracing the queries made be 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. (Notice that this corresponds to the first case in Lemma 1 that characterizes invalid symbolic adversaries.) 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 second case in Lemma 1. Roughly, in this case 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 completely partition the sample space of adversary $\mathcal{A}$ and 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 execution model, trace properties are predicates on the global execution traces, but the definition of satisfaction is specific to the models. We now give these definitions and state our first main theorem which is 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.* $\mathsf{Exec}^s(\Pi) \subseteq \mathsf{P}^s$.

One example of such trace properties is mutual entity authentication. Informally, a trace of a protocol is a "good" mutual entity authentication trace, if for any two identities a and b, if a has finished a session of the protocol with intended partner b, then b has finished a session with intended partner a. Using this characterization, we say that a protocol is a secure mutual authentication protocol if all its traces are good.

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, where 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 non-negligible probability.

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

Theorem 1. *Let Π be an executable protocol and $\mathsf{P}^s \subseteq \mathsf{SymbTr}$ be an arbitrary symbolic trace property and $\mathsf{P}^c \subseteq \mathsf{ConcTr}$ be a computational security property such that $\mathsf{c}(\mathsf{P}^s) \subseteq \mathsf{P}^c$. 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 $\mathsf{c}(\mathsf{P}^s) \subseteq \mathsf{P}^c$ 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 2, 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.* $\Pi \models^\mathsf{c} \Pi^\mathsf{c}$.

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.

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 $\mathsf{sid}_p = (s, i, (a_1, \ldots, a_k))$ where $a_1, \ldots, a_k$ are non corrupted agents (*i.e.* none of them appear in the $\mathbf{corrupt}$ query that may occur at the beginning), we have $H_n \not\vdash n(a_i, j, s)$. If this is the case, we write $\Pi \models^s \mathsf{SecNonce}^s(i, j)$.

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. The goal of the adversary is to guess b, *i.e.* to determine which of the two nonces was used in the execution. We set the outcome of the experiment to 1 if $b = d$ and we set it to 0 otherwise, and define the advantage of the adversary 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 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 $\Pi \models^c \mathsf{SecNonce}(i, j)$.*

Proof. It turns out that the proof of this theorem is essentially contained in that of Lemma 2: assuming the existence of a computational adversary $\mathcal{A}$ that contradicts the secrecy of some nonce that is symbolically secret, we show that a variant of the adversary $\mathcal{B}_2$ breaks the joint security of $\mathcal{DS}$ and $\mathcal{AE}$.

6 Automated Proof using Casrul

In this section we describe the automated tool Casrul [8] 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). It can be obtained from:
`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 signature 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.

Since the syntax of Casrul does not use labels for encryption, we have to ensure that security proofs for protocols in the model that does not use labels are sound w.r.t. to the models with labels. In the following we argue that this is true for a large class of trace properties, that include many formulations of authentication and secrecy. Namely, we consider the class of trace properties that are independent of the labels, *i.e.* are such that whenever a trace t with labels does not satisfy the property, the trace $\bar{t}$, obtained from t by removing the labels, does not satisfies the property either. Indeed, let us consider a trace property P that is label independent and a protocol such that the protocol without labels satisfies P. Let us show that the protocol with labels also satisfies P. By contradiction, assume there exists a valid trace t with labels such that $t \notin P$. Since P is label independent, we have $\bar{t} \notin P$. We have to prove that $\bar{t}$ is still a valid trace. First, the agents still accept the same messages since they only do equality tests and removing labels can only produce more equalities. Second, if a message with labels was deducible by the intruder, than the corresponding message without labels is also deducible, which allows to conclude that $\bar{t}$ is valid.

We note that authentication between participants and data authentication are label independent. 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 [12] protocol is a secure mutual authentication protocol (file `NSPK_LOWE3.hlpsl`) implies the same property, but in the computational model.

Similarly, the secrecy of nonces is also a label independent property. Thus, 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 [12] (file `NSPK_LOWE2.hlpsl`) and in the SPLICE protocol [19] (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 [7] or Securify [9]. 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.

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 universally composable cryptographic library. Cryptology ePrint Archive, Report 2003/015, 2003. http://eprint.iacr.org/.
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. 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.
8. 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.
9. V. Cortier. *A guide for Securify*. RNTL EVA project, Report n. 13, December 2003.
10. 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.
11. 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.
12. 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.
13. G. Lowe. Casper: A compiler for the analysis of security protocols. In *Proc. of 10th Computer Security Foundations Workshop (CSFW'97)*. IEEE Computer Society Press, 1997.
14. 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.
15. 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.
16. L. Paulson. Mechanized proofs for a recursive authentication protocol. In *Proc. of the 10th Computer Security Foundations Workshop (CSFW'97)*, pages 84–95. IEEE Computer Society Press, 1997.
17. C. Rackoff and D. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *CRYPTO'91*, pages 433–444, 1992.
18. M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *Proc. of the 14th Computer Security Foundations Workshop (CSFW'01)*, pages 174–190. IEEE Computer Society Press, 2001.
19. 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.

A Joint security of digital signatures and public-key encryption in a multi-user setting

SYNTAX OF DIGITAL SIGNATURES AND PUBLIC-KEY ENCRYPTION. The three algorithms $(\mathsf{K_s}, \mathsf{Sig}, \mathsf{Vf})$ that comprise digital signature scheme $\mathcal{DS}$ are as follows. The randomized algorithm $\mathsf{K_s}$ on input some security parameter η returns a pair of signing and verification keys $(\mathsf{sk}, \mathsf{vk})$. The signing algorithm takes as input a message $m \in \{0,1\}^*$ and a signing key sk and returns a signature σ. The validity of σ can be verified by running the verification algorithm on input (vk, m, σ), and it is required that for any m and sk, if $\sigma \stackrel{\$}{\leftarrow} \mathsf{Sig}(\mathsf{sk}, m)$ then $\mathsf{Vf}(\mathsf{vk}, m, \sigma) = 1$.

The algorithms $(\mathsf{K_e}, \mathsf{Enc}, \mathsf{Dec})$ of a public-key encryption scheme are as follows. The key generation algorithm takes as input a security parameter η and returns a pair of encryption and decryption keys $(\mathsf{ek}, \mathsf{dk})$. The encryption algorithm is also randomized. It takes as input an encryption key ek and some *plaintext* $m \in \{0,1\}^*$ and returns $c \in \{0,1\}^*$, an encryption of m. The message m can be recovered from the *ciphertext* c by running the decryption algorithm with inputs the decryption key dk and c.

We define the joint security of digital signature scheme $\mathcal{DS}$ and encryption scheme $\mathcal{AE}$ by using an experiment involving an adversary $\mathcal{A}$ and an oracle $\mathcal{O}_{\mathcal{AE},\mathcal{DS}}(\eta, b)$. The oracle is parametrized by a security parameter η and a *selection bit* b. The oracle maintains a counter k corresponding to the total number of users in the system, and four vectors of keys $\mathbf{pke}, \mathbf{ske}, \mathbf{sks}, \mathbf{pks}$ which on position i contain the keys for encryption, decryption, signing and verifying corresponding to player i.

At any point in time, the adversary can add a player to the system via the query $\mathsf{newuser}$. The result of the command is that the oracle creates appropriate keys for a new user by running the key generation algorithms for the encryption scheme $\mathcal{AE}$ and digital signature scheme $\mathcal{DS}$. The public key for encryption and verification are returned to the adversary. When the adversary issues the query $\mathsf{sign}(i, M)$ it obtains in return a signature on message M under the signing key of user i. For defining the security of encryption we adopt the left-right oracle style for the multi-user setting [6]. The adversary is allowed to submit pairs of equal-length messages M_0, M_1, together with the identity i of a player. The oracle returns an encryption of message M_b (determined by the selection bit b that parameterizes the oracle). Finally, the adversary can see decryption of ciphertext C of his own choosing under the decryption key of player i by submitting query $\mathsf{decrypt}(i, C)$. We require that an adversary never queries to user i a ciphertext C obtained as an encryption query to user i. The precise definition of how the oracle functions is in Figure 2.

Remark 1. We remark that our oracle does not capture *all* possible behaviors of honest parties. The oracle that we define can be used for security analysis of *all* protocols for which the messages sent by the honest parties can be simulated using adversary queries. Protocols that can not be analyzed include those which use encryptions of signing keys, or signatures on the secret decryption keys.

Joint security for encryption scheme $\mathcal{AE}$ and digital signature scheme $\mathcal{DS}$ is defined via the experiment in Figure 3. Here an adversary $\mathcal{A}$ interacts with the oracle $\mathcal{O}_{\mathcal{AE},\mathcal{DS}}(\eta, b)$ for some selection bit chosen at random. The goal of the adversary is to forge a signature on a message (the first kind of forgery), or to guess the selection bit b.

Oracle $\mathcal{O}_{\mathcal{AE},\mathcal{DS}}(b,\eta)$
Set $k \leftarrow 0$;
Initialize vectors $\mathbf{pke}, \mathbf{ske}, \mathbf{pks}, \mathbf{sks}$;
On query newuser do:
  $k \leftarrow k+1$;
  $(\mathbf{pke}[k], \mathbf{ske}[k]) \leftarrow \mathsf{K_e}(1^\eta)$
  $(\mathbf{pks}[k], \mathbf{sks}[k]) \leftarrow \mathsf{K_s}(1^\eta)$
  Return $\mathbf{pke}[k], \mathbf{pks}[k]$
On query $\mathsf{encrypt}(i, M_0, M_1)$ do:
  If $k < i$ return $\perp$
    Otherwise return $\mathsf{Enc}(\mathbf{pke}[i], M_b)$
On query $\mathsf{decrypt}(i, C)$ do:
  Return $\mathsf{Dec}(\mathbf{ske}[i], C)$
On query $\mathsf{sign}(i, M)$ do:
  If $k < i$ return $\perp$
    Otherwise return $\mathsf{Sig}(\mathbf{sks}[i], M)$

Fig. 2. An oracle for joint encryption and signing; a valid adversary is not allowed to issue a query $\mathsf{decrypt}(i, C)$ if C was the result of an encryption query $\mathsf{encrypt}(i, M_0, M_1)$

Experiment $\mathbf{Exp}^{\text{es-sec}}_{\mathcal{A},\mathcal{AE},\mathcal{DS}}(\eta)$
  $b \xleftarrow{\$} \{0,1\}$
  Run $\mathcal{A}^{\mathcal{O}_{\mathcal{AE},\mathcal{DS}}(\eta,b)}$
  If $\mathcal{A}$ outputs $(1,(i,m,\sigma))$ and
    (i,m) was never queried
    $\mathsf{Vf}(\mathbf{pks}[i], m, \sigma) = 1$
  Then Return 1
  If $\mathcal{A}$ outputs $(2,d)$ and $d = b$ then Return 1
  Otherwise Return 0

Fig. 3. Experiment for defi ning security of encryption and signing in a multi-user setting

We say that using the asymmetric encryption scheme $\mathcal{AE}$ together with the digital signature scheme $\mathcal{DS}$ in a multi-user setting is secure if for all p.p.t. adversaries $\mathcal{A}$

$$\Pr\left[\,\mathbf{Exp}^{\text{sec}}_{\mathcal{A},\mathcal{AE},\mathcal{DS}}(\eta) = 1\,\right] \leq \frac{1}{2} + \nu(\eta)$$

B Formal Definition of Executable Protocols

Definition 3. *A protocol* $\Pi : [n] \to \mathsf{Roles}$ *with* $\Pi(j) = ((l_1^j, r_1^j), \ldots, (l_{k_j}^j, r_{k_j}^j))$ *is executable if:*

1. *The protocol has the* executable decryption *property, i.e. for all* $A_j \in \mathsf{X}.a$ *the only encryption keys that are contained in terms* l_k^j *(for* $k \in [k_j]$*) are* $\mathsf{ek}(A_j)$*;*
2. *For all* $j \in [n]$*, all* $k \in [k_j]$ *and all* $A \in \mathsf{X}.a$ *we require that whenever* l_k^j *contains a signature* $[t]^l_{\mathsf{sk}(A)}$ *for some term* $t \in \mathsf{T}_\Sigma(\mathsf{X})$*, the term* t *can be computed from* $l_1^j, l_2^j, \ldots, l_k^j, \mathbf{kn}(A_j)$ *by Dolev-Yao operations, i.e.* $l_1^j, l_2^j, \ldots, l_k^j, \mathbf{kn}(A_j) \vdash_a t$*;*
3. *The messages that are sent are computable: for all* $k \in [k_j]$ *we require that* r_k^j *can be computed from* $l_1^j, l_2^j, \ldots, l_k^j, \mathbf{kn}(A_j)$ *by Dolev-Yao operations, i.e.* $l_1^j, l_2^j, \ldots, l_k^j, \mathbf{kn}(A_j) \vdash_a r_k^j$*;*
4. *For all* $j \in [n]$*, all* $k \in [k_j]$*, the variables of* r_k^j *are contained in the union of the variables of* l_k^j*,* $X.a$ *and* $X_n(A_j)$*. In addition, the terms* $l_1^j, \ldots, l_k^j$ *do not contain label variables and for any subterm* $\{m\}^l_{\mathsf{ek}(B)}$ *of* $r_1^j, \ldots, r_k^j$*,*
   - *either* l *is a label variable and for any* $\{m'\}^l_{\mathsf{ek}(B')}$ *subterm of* $r_1^j, \ldots, r_k^j$*, we have* $m = m'$ *and* $B' = B$*,*
   - *or* l *is of the form* $\mathsf{ag}(n)$ *(*$n \in \mathbb{N}$*) and* $B = A_j$ *and similarly for signatures.*

The Needham-Schroeder-Lowe protocol, described in Example 1, is executable.

C Proof of Lemma 1

Proof. Consider a maximal p path in m such that none of the terms along the path are deducible: for every $p' \leq p$, $S \not\vdash m|_{p'}$.

- Either $m|_p$ is a leaf of m, thus is a nonce (otherwise $m|_p$ would be deducible). Since $m|_p$ is not deducible, $m|_p$ has to be a nonce appearing in S thus is a subterm of terms of S.
- Or $m|_p = [t]_k$ for some k and t. Then, either $[t]_k$ is not a subterm of terms in S and we have case 1, or $[t]_k$ is a subterm of terms of S.

We are left to the case where $m|_p$ is a subterm of terms of S and $S \not\vdash m|_p$. Let p' be the minimal path such that $p' < p$ and $m|_{p'}$ is a subterm m' of terms of S. Let $t = m|_{p'}$. Since t is not deducible, t has to appear under some encryption in m': $m' = C[\{C'[t]\}_k]$ with $S \not\vdash k^{-1}$. By minimality of p', $\{C'[t]\}_k$ does not appear as subterm along the path from m to t, thus we are in case 2.

D Proof of Lemma 2

Proof. We do the proof in two steps. First we fix random coins for the protocol and for adversary and show that the resulting concrete execution trace t^{c} is the instantiation of some symbolic execution trace t^s, *i.e.* $t^s \preceq t^{\mathsf{c}}$. We construct t^s by attaching to each query made by $\mathcal{A}$ a symbolic query; the

abstract trace t^s is the result of these queries. In the second step, we prove that with overwhelming probability over the choice of the random coins, t^s is a valid symbolic execution trace, *i.e.* $t^s \in \mathsf{Exec}^s(\Pi)$.

STEP I. Fix $R_\Pi, R_\mathcal{A}$ random coins for the honest parties and for the adversary. We start the execution of $\mathcal{A}$ which we carry out normally by answering its queries. Notice that by fixing R_Π, all cryptographic keys and nonces generated by the honest parties are fixed, and also, all random coins used for encrypting and signing are also fixed. During the execution we maintain a function $\mathsf{c} : \mathcal{C}^\eta \to T^f$ mapping each bit-string that occurs to an abstract term. This mapping is initialized by first canonically mapping each agent name in $\mathcal{C}^\eta$ to a symbolic name. By abusing notation, we set $\mathsf{c}(a_i) = a_i$. Also, the concrete cryptographic keys to the corresponding symbolic keys: if sk_i, vk_i are the signing and verification keys of agent a_i then we set $\mathsf{c}(sk_i) = \mathsf{sk}(a_i), \mathsf{c}(vk_i) = \mathsf{vk}(a_i)$. Similarly for encryption keys.

The execution of $\mathcal{A}$ proceeds as described in the concrete execution model: since all keys are known, it is clear how to parse and answer the queries of the adversary. We now explain how c is updated after the queries made by $\mathcal{A}$. Session initialization queries are straightforward: on a request $\mathbf{new}(i, a_1, a_2, \ldots, a_n)$, the function c maps the nonces used in the concrete execution (recall that these nonces are uniquely determined by R_Π) to the corresponding nonce symbols used to execute the same query, but in the symbolic model.

In the case of message transmission queries (of the form) $\mathbf{send}(s, m)$, the function c is updated as follows. We first construct the parse tree of m: notice that we decrypt all ciphertexts (since the keys of all honest parties are known), and similarly, we can construct the parse tree of signatures since we from signatures we can recover the message, the key used for verification (thus implicitly, the key used for signing).

Notice that the leaves of the parse tree of m are either random nonces, party identities or cryptographic keys. The function c is already defined on agent identities, cryptographic keys and on some of the nonces. The rest of the nonces that appear must be adversary created nonces, and we simply map them to fresh nonce symbols (from the adversary's knowledge set). Mapping the rest of the subexpressions of m is a straightforward bottom-up procedure.

We denote by $t^s = \mathsf{c}(\mathsf{Exec}_{\Pi(R_\Pi),\mathcal{A}(R_\mathcal{A})}(\eta))$ the symbolic trace determined by the sequence of queries of $\mathcal{A}$ in which we replace the queries transmission queries $\mathbf{send}(s, m)$ with their symbolic counterpart $\mathbf{send}(s, \mathsf{c}(m))$. The inverse of the function c maps the trace t^s to the trace $\mathsf{Exec}_{\Pi(R_\Pi),\mathcal{A}(R_\mathcal{A})(\eta)}$ and we write $\mathsf{c}^{-1}(t^s) = \mathsf{Exec}_{\Pi(R_\Pi),\mathcal{A}(R_\mathcal{A})}(\eta)$.

STEP II. We now prove that with overwhelming probability over the choice of random coins, the trace t^s constructed above is a valid trace. Assuming the opposite, we show that the encryption scheme $\mathcal{AE}$ and digital signature schemes $\mathcal{DS}$ are not jointly secure, contradicting the hypothesis of the theorem.

For the sake of contradiction, assume there exists some adversary $\mathcal{A}$ such that the trace $t^s = \mathsf{c}(\mathsf{Exec}_{\Pi(R_\Pi),\mathcal{A}(R_\mathcal{A})}(\eta))$ is not valid with non-negligible probability. Therefore there exists some query $\mathbf{send}(s, m)$ made by $\mathcal{A}$ such that the query $\mathbf{send}(s, \mathsf{c}(m))$ is not valid in the symbolic execution. We distinguish two different cases, depending on the type of forgery that $\mathcal{A}$ outputs, as described in Lemma 1.

Case I. First, we assume that the invalid query of $\mathcal{A}$ is $\mathbf{send}(s, m)$ such that the symbolic term associated to m, *i.e.* $\mathsf{c}(m)$, contains some signature $[t]_{sk}$ which did not occur at all in the prior communication. In this case, we construct an adversary $\mathcal{B}_1$ against $\mathcal{AE}$ and $\mathcal{DS}$ as follows: it runs adversary $\mathcal{A}$ as a subroutine maintaining the global state of the execution by itself and playing the role of the honest parties during the execution described by $\mathsf{Exec}_{\Pi,\mathcal{A}}(\eta)$. The queries that $\mathcal{A}$ makes are answered using the oracle $\mathcal{O}_{\mathcal{AE},\mathcal{DS}}(b, \eta)$ to which $\mathcal{B}$ has access. Adversary $\mathcal{B}$ can create keys for the users by issuing **newuser** queries to the oracle, and w.l.o.g. we assume that the keys of user a_i are created in response to the i'th **newuser** query of $\mathcal{B}_1$. The adversary $\mathcal{B}_1$ parses the queries made by $\mathcal{A}$ using decryption queries to $\mathcal{O}_{\mathcal{AE},\mathcal{DS}}(b, \eta)$ (when these queries contain ciphertexts that need decrypted) and computes appropriate responses which can be easily done since $\mathcal{B}_1$ maintains the local state of all parties and can query the oracle $\mathcal{O}_{\mathcal{AE},\mathcal{DS}}(b, \eta)$ in the case it needs to produce signatures. The view that $\mathcal{A}$ has of the execution is precisely as in $\mathsf{Exec}_{\Pi,\mathcal{A}}(\eta)$ so at some point it will issue a request $\mathbf{send}(s, m)$ such that m contains a signature σ on some message M which did not appear at all prior to this point. If the key used for verifying σ is that of user a_i (recall that we assume that the verification key can be obtained from σ) adversary $\mathcal{B}_1$ outputs $(1, (i, M, \sigma))$ and stops. Since σ is a valid signature on M with respect to the public key of a_i, the adversary $\mathcal{B}_1$ wins in the experiment $\mathbf{Exp}^{\text{es-sec}}_{\mathcal{B},\mathcal{AE},\mathcal{DS},}(\eta)$.

Case II. Here we assume that the first invalid query output by $\mathcal{A}$ is $\mathbf{send}(s, m)$ such that $\mathsf{c}(m)$ is as follows: there exists some subterm term t of $\mathsf{c}(m)$ a message m' sent by the honest parties, two contexts C and C' such that: $m' = C[\{C'[t]\}_{\mathsf{ek}(a_i)}]$ and the term $\{C'[t]\}_{\mathsf{ek}(a_i)}$ is not a subterm that appears on the path from m to t in the parse tree of m. Let us note that since $\{C'[t]\}_{\mathsf{ek}(a_i)}$ can not be computed by the adversary, it must be the case that term t itself contains information which is secret to the adversary (or otherwise he could have built this term by himself). Since we only consider protocols in which the secret keys of the parties are never sent over the network, this secret information is either a nonce produced by a user, or a signature of a honest user. For each of these two cases, respectively, we construct adversaries $\mathcal{B}_2$ and $\mathcal{B}_3$ against the joint security of $\mathcal{AE}$ and $\mathcal{DS}$, more precisely, we show how to determine the bit b that parameterizes the oracle $\mathcal{O}_{\mathcal{DS},\mathcal{AE}}(b, \eta)$. We start by explaining the construction of $\mathcal{B}_2$ (for the case when the secret contained in m is a nonce) and explain how to extend this approach to the case of signatures. Just as for the first case, when we call a concrete message m a term, we are actually referring to the term $\mathsf{c}(m)$ associated to m by the construction described in Step I of this proof.

For simplicity of exposition, assume for the moment that we know which of the secret nonces is the "faulty" nonce, *i.e.* we know in which of the sessions the nonce is created and to which of the variable it corresponds. For concreteness assume it is $n(\mathsf{sid}, i, j)$, *i.e.* it is created during session sid and it corresponds to variable $X^j_{A_i}$. Moreover, assume we also know which of the messages transmitted by the honest parties the term $\{C'[t]\}_{\mathsf{ek}}$ is located, and the precise position within this term where t occurs. Finally, assume that we can determine which query of the adversary contains the message m, and on which position t occurs in m.

Adversary $\mathcal{B}_2$ runs adversary $\mathcal{A}$ as a subroutine and uses his access to the oracle $\mathcal{O}_{\mathcal{AE},\mathcal{DS}}(b, \eta)$ to handle the queries made by $\mathcal{A}$. For most of the execution, this is done as we explained for the case of adversary $\mathcal{B}_1$: queries are parsed by using the decryption capabilities of the oracle, and the answers are computed using the knowledge of all the local states (which are maintained by $\mathcal{B}_2$ itself).

We now describe the crucial aspects of this simulation which is how $\mathcal{B}_2$ handles the requests of $\mathcal{A}$ that involve nonce $n(\mathsf{sid}, i, j)$. When $\mathcal{A}$ requests the initiation of session sid, all variables are initialized as usual with the exception of $X^j_{A_i}$. For this variable, adversary $\mathcal{B}_2$ selects *two* random values say n_0 and n_1. We will now explain how $\mathcal{B}$ carries out the simulation of the environment of $\mathcal{A}$ as if the value of $X^j_{A_i}$ in session sid is n_b. Notice that this is despite $\mathcal{B}$ not knowing the bit b (which, recall, parameterizes the oracle $\mathcal{O}_{\mathcal{DS},\mathcal{AE}}(b,\eta)$.) If $\mathcal{B}$ needs to return to $\mathcal{A}$ messages that contain the nonce n_b, generally, it does so using the encryption oracle. For example, say that at some point during the execution, $\mathcal{B}$ needs to return to $\mathcal{A}$ as answer the value of $\{X^j_{A_i}, [X^j_{A_i}]_{\mathsf{sk}(A_i)}\}_{\mathsf{ek}(A_t)}$ in which $X^j_{A_i}$ has value n_b.

In this case, adversary $\mathcal{B}_2$ makes signing requests to the oracle $\mathcal{O}_{\mathcal{AE},\mathcal{DS}}(b,\eta)$ and obtains signatures σ_0 and σ_1 on n_0 and n_1 respectively, under the signing key of party a_i (here we assume for simplicity that A_i is instantiated with agent identity a_i). Then it submits the pair $((n_0,\sigma_0),(n_1,\sigma_1))$ to the encryption oracle, and requires an encryption under the public key of a_t. The oracle returns precisely the encryption c of (n_b,σ_b) under $\mathsf{ek}(a_t)$, since the selection is determined by the parameter bit b. This is the ciphertext returned to $\mathcal{A}$. This is precisely how the adversary $\mathcal{B}_2$ also computes the bit-string representation of $C[\{C'[t]\}_{ek(a_i)}]$. Notice that $\mathcal{B}_2$ never needs to decrypt ciphertexts that are obtained as answers to encryption queries to the oracle, since it already knows the underlying plaintext (modulo the value for the nonce $n(\mathsf{sid}, i, j)$) and this suffices for processing these queries.

At some point, the adversary $\mathcal{A}$ will make its invalid query $\mathbf{send}(s,m)$, and from m the adversary $\mathcal{A}$ can determine the value of the term t by parsing the message m: notice that on the path from m to t there are no encryptions which were created during the execution of the protocol so the value of t, and thus the value of the bit b can be determined with non-negligible probability.

The above argument is valid, provided that $\mathcal{B}_2$ knows the nonce $n(\mathsf{sid}, i, j)$ and it also knows as well when and where during the execution various terms (which we specified in the beginning) occur. Neither one of these quantities is a priori known to $\mathcal{B}_2$, but the adversary can simply guess them: since the adversary $\mathcal{A}$ runs in probabilistic polynomial-time the total number of messages that are sent/received as well as the number of sessions is polynomial in the security parameter. This implies that the probability that all guesses are correct is non-negligible, thus so is the probability that $\mathcal{B}_2$ guesses the value of the bit b correctly.

The adversary $\mathcal{B}_3$ operates similarly: the role of the nonce $n(\mathsf{sid}, i, j)$ is played by the unknown signature. During the simulation two possible different values σ_0 and σ_1 are generated for this signature, and the simulation value used in the simulation is σ_b, where b is the selection bit of the oracle $\mathcal{O}_{\mathcal{DS},\mathcal{AE}}(b,\eta)$. When $\mathcal{A}$ makes its invalid query, which contains σ_b, adversary $\mathcal{B}_3$ recovers σ_b and therefore b, i.e. $\mathcal{AE}$ and $\mathcal{DS}$ are not jointly secure.

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399