A New Knowledge Representation Strategy for Cryptographic Protocol Analysis

Ivan Cibrario B.[1], Luca Durante[1], Riccardo Sisto[2], and Adriano Valenzano[1]

[1] Istituto di Elettronica e di Ingegneria dell'Informazione e delle Telecomunicazioni
c/o Politecnico di Torino, C.so Duca degli Abruzzi 24
I-10129 Torino, Italy
durante,valenzano,ivan.cibrario@polito.it
[2] Dipartimento di automatica e Informatica
Politecnico di Torino, C.so Duca degli Abruzzi 24
I-10129 Torino, Italy
sisto@polito.it

Abstract. The formal verification of security properties of a cryptographic protocol is a difficult, albeit very important task as more and more sensible resources are added to public networks. This paper is focused on model checking; when adopting this approach to the problem, one challenge is to represent the intruder's knowledge in an effective way. We present an intruder's knowledge representation strategy that supports the full term language of spi calculus and does not pose artificial restrictions, such as atomicity or limited maximum size, to language elements. In addition, our approach leads to practical implementation because the knowledge representation is incrementally computable and is easily amenable to work with various term representation languages.

1 Introduction

The formal verification of security properties of cryptographic protocols is a difficult task; one possible approach is to use model checking, which has already been explored in previous papers such as [5,8,10,11]. When using this approach, one of the challenges is to find a compact and efficient representation of the intruder's knowledge, which plays a central role in modeling the protocol behavior in an hostile environment, without sacrificing the expressive power of the specification language. Previous papers only give partial solutions to this problem, the most common approach being to restrict the way in which intruder messages can be built. For example, a common restriction is to force encryption keys to be atomic [5,16].

This paper presents a novel intruder's knowledge representation strategy that achieves both the goals of compactness and implementation efficiency; in particular, the proposed representation is largely independent from the term representation language chosen, i.e. supports all main cryptographic message contruction operators, including the full term language of spi calculus [1]. Moreover, it keeps

H. Garavel and J. Hatcliff (Eds.): TACAS 2003, LNCS 2619, pp. 284–298, 2003.
© Springer-Verlag Berlin Heidelberg 2003

the intruder's knowledge in a minimized, canonical form, and is shown to be incrementally computable.

Having a canonical intruder's knowledge representation is especially important when state exploration techniques are used to check security properties. In fact, simpler non-canonical representations, such as for example the plain set of messages the intruder has access to when eavesdropping honest agents, may lead to state proliferation, because different sets of known messages may correspond to the same intruder knowledge. For example, knowing a symmetric key k and the encryption of a data item m under k is equivalent to knowing the cleartext m and k.

This representation has been used in a broader framework, presented in [9] and sketched in [8], where automatic testing equivalence verification of spi calculus specifications is described. There, the intruder's knowledge representation is further empowered by symbolic techniques; one of their functions is to prevent state space explosion associated with the intruder's ability to synthesize complex keys.

This paper presents the proposed intruder's knowledge representation by means of the spi calculus specification language. Since the expressive power of the message specification section of this language is better than or equivalent to the one of most other formalisms for cryptographic protocols, the adoption of spi calculus as our reference specification language is not restrictive.

The paper assumes that the reader is familiar with basic cryptography, and is structured as follows: section 2 presents the syntax of spi calculus, and informally describes its semantics. In section 3, we discuss our knowledge representation strategy, while in sections 4 and 5 we outline its computational complexity and work out some examples, respectively. Last, section 6 discusses related works and draws some conclusions.

2 Spi Calculus

The spi calculus is defined in [1] as an extension of the π calculus [15] with cryptographic primitives. It is a process algebraic language designed for describing and analyzing cryptographic protocols. These protocols heavily rely on cryptography and on message exchange through communication channels; accordingly, the spi calculus provides powerful primitives to express cryptography and communication.

This section summarizes the syntax and describes the language's semantics informally; the language used in this paper fully conforms to the spi calculus definition found in [1], with the naming conventions outlined in Tab. 1.

A spi calculus specification is a system of independent processes, executing in parallel; they synchronize via message-passing through named communication channels. The spi calculus has two basic language elements: terms, to represent data, and processes, to represent behaviors.

Terms can be either atomic elements, i.e. names, including the special name 0 representing the integer constant zero, and variables, or compound terms built

Table 1. Naming Conventions

m	ranges over names
x and y	range over variables
P, Q and R	range over processes
σ and ρ	range over terms
Σ	ranges over sets of terms

using the term composition operators listed in Tab. 2. Names may represent communication channels, atomic keys and key pairs, nonces (also called *fresh names*) and any other unstructured data.

If a term σ occurs as a sub-expression of another term ρ, then σ is called a *subterm* of ρ; moreover, any term σ is a subterm of itself.

Table 2. Term syntax of spi calculus

σ, ρ	terms
m	name
(σ, ρ)	pair
0	zero
$\mathrm{suc}(\sigma)$	successor
x	variable
$\mathrm{H}(\sigma)$	hashing
$\{\sigma\}_\rho$	shared-key encryption
$\sigma^+,\ \sigma^-$	public/private part
$\{[\sigma]\}_\rho$	public-key encryption
$[\{\sigma\}]_\rho$	private-key signature

The informal meaning of the composition operators is as follows:

- (σ, ρ) is the *pairing* of σ and ρ. It is a compound term whose components are σ and ρ. Pairs can always be freely split into their components.
- $\mathrm{suc}(\sigma)$ is the *successor* of σ. This operator has been introduced mainly to represent successors over integers, but it can be used, more generally, as the abstract representation of an invertible function on terms.
- $\mathrm{H}(\sigma)$ is the *hashing* of σ. $\mathrm{H}(\sigma)$ represents a function of σ that cannot be inverted.
- Term $\{\sigma\}_\rho$ is the ciphertext obtained by encrypting σ under key ρ using a shared-key cryptosystem.
- σ^+ and σ^- represent respectively the public and private half of a key pair σ. σ^+ cannot be deduced from σ^- and vice versa.
- $\{[\sigma]\}_\rho$ is the result of the public-key encryption of σ with ρ.
- $[\{\sigma\}]_\rho$ is the result of the signature (private key encryption) of σ with the private key ρ.

Besides term specification, spi calculus also offers a set of operators to build behavior expressions that, in turn, represent processes. For example, Fig. 1 shows the spi calculus specification of a very simple protocol inspired by [10]. The left hand side of the figure shows the message exchanges involved in the protocol, using the informal, intuitive representation often encountered in the literature, whereas the right hand side of the figure shows the corresponding spi calculus specification. In this protocol, two agents A and B sharing a secret key k and represented by spi calculus processes P_A and P_B, exchange two messages:

- First, A sends to B message M encrypted under key k over public channel c. This is represented, in spi calculus, by the output statement $\overline{c}\langle\{M\}_k\rangle$ in process P_A and by the corresponding input statement $c(y_1)$ in P_B; the latter statement assigns the datum just received to variable $y1$ of P_B.
- Then, B tries to decrypt the message with key k, as specified by the statement $case\ y_1\ of\ \{y_2\}_k$ in P_B and, when successful, sends back to A the hashed cleartext of M on the same channel c, with the output statement $\overline{c}\langle H(y_2)\rangle$. Process A receives this message with the input statement $c(x)$.
- Finally, A checks that the hash just received is correct with the statement $[x\ is\ H(M)]$ and proceeds with further operations on message M, represented by the unspecified process $F(M)$.

The role of the spi calculus process P_{sample} in the example is twofold:

- With the restriction operator (νk), it generates a restricted, private name k, only known to P_A, P_B and itself, to be used as the encryption key.
- It instantiates both P_A and P_B to run in parallel, by means of the parallel composition operator $|$, so that an instance of P_{sample} represents all agents involved in a session of the protocol.

$A \rightarrow B : \{M\}_k$	$P_A(M) \stackrel{\triangle}{=} \overline{c}\langle\{M\}_k\rangle.\ c(x).\ [x\ is\ H(M)]\ F(M)$
$B \rightarrow A : H(M)$	$P_B \stackrel{\triangle}{=} c(y_1).\ case\ y_1\ of\ \{y_2\}_k\ in\ \overline{c}\langle H(y_2)\rangle.\ 0$
	$P_{\text{sample}} \stackrel{\triangle}{=} (\nu k)(P_A(M) \mid P_B)$

Fig. 1. A simple spi calculus specification

3 Intruder's Knowledge Representation

Our approach to the representation of the knowledge that an intruder can acquire borrows some of the notation and concepts introduced in [5], and has some similarities with [10] and [11] as section 6 points out; however, it is more sophisticated in some respects:

- encryption and decryption keys are not restricted to be atomic.
- we focus on spi calculus to its full extent, including public/private key cryptosystems, and the related operators.
- the intruder's knowledge is always kept in a minimized form that both speeds up and simplifies processing.

Moreover, as most other researchers do, our method relies on the following well-known, *perfect encryption* assumptions:

- to extract the cleartext m from the encrypted messages $\{m\}_k$, $\{[m]\}_{k^+}$ and $[\{m\}]_{k^-}$, the corresponding decryption keys k, k^- and k^+ must be known.
- the cryptosystem has enough redundancy so that the decryption algorithm can determine whether its task succeeded in, and to prevent encryption collisions.
- it is not possible for an attacker to guess or forge any secret data item.

The intruder's model we adopted is the so-called *Dolev-Yao model*, and has been inspired by [7]. In other words, we assume that the intruder is able to:

- eavesdrop on, remove, replay and arbitrarily reorder messages sent over public communication channels.
- forge new messages with the pieces of messages already intercepted, possibly from previous protocol sessions, and inject them into the public channels.
- generate its own nonces.
- decrypt encrypted messages, provided it has got the appropriate key, and split compound cleartext messages into pieces.

Let $\mathcal{A}$ be the set of spi calculus names, including the integer constant 0, and $\mathcal{M}(\mathcal{A})$ the set of all spi calculus terms that can be built by combining the elements of $\mathcal{A}$ by means of the operators defined in Tab. 2. For simplicity, and without loss of generality, name overloading is forbidden, i.e. it is assumed that distinct elements are always identified by distinct names.

The closure of a set of terms $\Sigma \subseteq \mathcal{M}(\mathcal{A})$ is denoted $\widehat{\Sigma}$ and is defined as the set of all spi calculus terms that can be built by combining the elements of Σ by means of the operators defined in Tab. 2 and their inverses. Formally, $\widehat{\Sigma}$ is the least set of terms such that, for each σ, σ_1 and $\sigma_2 \in \mathcal{M}(\mathcal{A})$, the following closure rules hold:

$$\sigma \in \Sigma \quad \Rightarrow \sigma \in \widehat{\Sigma} \tag{1}$$

$$\sigma \in \widehat{\Sigma} \quad \Rightarrow \quad \mathrm{suc}(\sigma) \in \widehat{\Sigma} \qquad \text{(successor)} \tag{2}$$

$$\sigma_1 \in \widehat{\Sigma} \wedge \sigma_2 \in \widehat{\Sigma} \quad \Rightarrow \quad (\sigma_1,\ \sigma_2) \in \widehat{\Sigma} \qquad \text{(pairing)} \tag{3}$$

$$\sigma_1 \in \widehat{\Sigma} \wedge \sigma_2 \in \widehat{\Sigma} \quad \Rightarrow \quad \{\sigma_1\}_{\sigma_2} \in \widehat{\Sigma} \qquad \text{(sh. key encryption)} \tag{4}$$

$$\sigma \in \widehat{\Sigma} \quad \Rightarrow \quad \mathrm{H}(\sigma) \in \widehat{\Sigma} \qquad \text{(hashing)} \tag{5}$$

$$\sigma_1 \in \widehat{\Sigma} \wedge \sigma_2^+ \in \widehat{\Sigma} \quad \Rightarrow \quad \{[\sigma_1]\}_{\sigma_2^+} \in \widehat{\Sigma} \qquad \text{(pub. key encryption)} \tag{6}$$

$$\sigma_1 \in \widehat{\Sigma} \wedge \sigma_2^- \in \widehat{\Sigma} \quad \Rightarrow \quad [\{\sigma_1\}]_{\sigma_2^-} \in \widehat{\Sigma} \qquad \text{(private key signature)} \tag{7}$$

$$\sigma \in \widehat{\Sigma} \quad \Rightarrow \sigma^+ \in \widehat{\Sigma} \wedge \sigma^- \in \widehat{\Sigma} \qquad \text{(key projection)} \tag{8}$$

$$\mathrm{suc}(\sigma) \in \widehat{\Sigma} \;\Rightarrow\; \sigma \in \widehat{\Sigma} \quad \text{(prec)} \tag{9}$$

$$(\sigma_1,\, \sigma_2) \in \widehat{\Sigma} \;\Rightarrow\; \sigma_1 \in \widehat{\Sigma} \wedge \sigma_2 \in \widehat{\Sigma} \quad \text{(projection)} \tag{10}$$

$$\{\sigma_1\}_{\sigma_2} \in \widehat{\Sigma} \wedge \sigma_2 \in \widehat{\Sigma} \;\Rightarrow\; \sigma_1 \in \widehat{\Sigma} \quad \text{(sh. key decryption)} \tag{11}$$

$$\{[\sigma_1]\}_{\sigma_2^+} \in \widehat{\Sigma} \wedge \sigma_2^- \in \widehat{\Sigma} \;\Rightarrow\; \sigma_1 \in \widehat{\Sigma} \quad \text{(pub. key decryption)} \tag{12}$$

$$\{[\sigma_1]\}_{\sigma_2^-} \in \widehat{\Sigma} \wedge \sigma_2^+ \in \widehat{\Sigma} \;\Rightarrow\; \sigma_1 \in \widehat{\Sigma} \quad \text{(signature check)} \tag{13}$$

$$\sigma^+ \in \widehat{\Sigma} \wedge \sigma^- \in \widehat{\Sigma} \;\Rightarrow\; \sigma \in \widehat{\Sigma} \quad \text{(key pairing)} \tag{14}$$

In principle, if Σ is the set of messages the intruder has intercepted so far, the generation of individual elements of $\widehat{\Sigma}$ (informally, the set of all messages the intruder can generate at a given point) starting from Σ can be viewed as a derivation in a natural deduction system [17].

In this respect, closure rule (1) represents the ability to derive an element from itself, closure rules (2)-(8) are equivalent to the introduction rules of the natural deduction system ($\mathcal{I}$ rules), and closure rules (9)-(14) are equivalent to its elimination rules ($\mathcal{E}$ rules).

For example, using the rule notation of [17], closure rule (4) is equivalent to the introduction rule:

$$\frac{\sigma_1 \quad \sigma_2}{\{\sigma_1\}_{\sigma_2}} \;\{\}\text{-}\mathcal{I} \text{ rule} \;. \tag{15}$$

Informally, we can read this rule as: "when both σ_1 and σ_2 are known to the intruder, then the intruder also knows about $\{\sigma_1\}_{\sigma_2}$."

Similarly, closure rule (11) is equivalent to the elimination rule:

$$\frac{\{\sigma_1\}_{\sigma_2} \;\; \sigma_2}{\sigma_1} \;\{\}\text{-}\mathcal{E} \text{ rule} \;. \tag{16}$$

Informally, we can read this rule as: "when the intruder knows both $\{\sigma_1\}_{\sigma_2}$ and σ_2, then it can succesfully perform a decryption and add σ_1 to its knowledge."

In an elimination rule, the premise that contains the operator removed by the rule is called the *major premise*, while all other premises are called *minor premises*. For example, in rule (16), $\{\sigma_1\}_{\sigma_2}$ is the major premise and σ_2 is the minor premise.

The theory of natural deduction [17] implies that, if $\sigma \in \widehat{\Sigma}$, then σ can be deduced from Σ with a *natural* deduction in *normal* form, that is, a chain of applications of $\mathcal{E}$ rules followed by a chain of applications of $\mathcal{I}$ rules, along the rules' major premises. This is not necessarily true along minor premises, so the closure of Σ under $\mathcal{E}$ rules only is not a suitable candidate to represent the intruder's knowledge, unless some additional constraints are imposed, such as the atomicity of encryption keys; this is exactly the approach adopted, for example, in [5,16]. In the following, we show that this difficulty can be overcome by introducing the concept of *minimal closure seed of* Σ and by suitably refining the derivation rules of the deduction system.

We say that a set of terms is *finite* if it contains a finite number of finite length elements. Given a finite set of terms Σ, we define the *minimal closure seed of* Σ, and denote it as $\overline{\Sigma}$, the subset of $\widehat{\Sigma}$ that satisfies the following predicates for each $a \in \mathcal{A}$, and for each σ, σ_1, $\sigma_2 \in \mathcal{M}(\mathcal{A})$:

$$a \in \overline{\Sigma} \quad \Leftrightarrow \quad a \in \widehat{\Sigma} \tag{17}$$

$$\mathrm{suc}(\sigma) \notin \overline{\Sigma} \tag{18}$$

$$(\sigma_1, \sigma_2) \notin \overline{\Sigma} \tag{19}$$

$$\{\sigma_1\}_{\sigma_2} \in \overline{\Sigma} \quad \Leftrightarrow \quad \sigma_2 \notin \widehat{\Sigma} \tag{20}$$

$$\mathrm{H}(\sigma) \in \overline{\Sigma} \quad \Leftrightarrow \quad \sigma \notin \widehat{\Sigma} \tag{21}$$

$$\{[\sigma_1]\}_{\sigma_2^+} \in \overline{\Sigma} \quad \Leftrightarrow \quad \sigma_2^+ \notin \widehat{\Sigma} \vee \sigma_1 \notin \widehat{\Sigma} \tag{22}$$

$$[\{\sigma_1\}]_{\sigma_2^-} \in \overline{\Sigma} \quad \Leftrightarrow \quad \sigma_2^- \notin \widehat{\Sigma} \vee \sigma_1 \notin \widehat{\Sigma} \tag{23}$$

$$\sigma^+ \in \overline{\Sigma} \quad \Leftrightarrow \quad \sigma \notin \widehat{\Sigma} \tag{24}$$

$$\sigma^- \in \overline{\Sigma} \quad \Leftrightarrow \quad \sigma \notin \widehat{\Sigma} \tag{25}$$

For example, if $\Sigma = \{\{[m]\}_{a^+}, a^-\}$, then $\overline{\Sigma} = \{\{[m]\}_{a^+}, a^-, m\}$, because:

- $a^- \in \overline{\Sigma}$, by rule (25), because $a \notin \widehat{\Sigma}$ and is therefore impossible to represent the key pair (a^+, a^-) with the key name a alone.
- $m \in \overline{\Sigma}$: since $a^- \in \overline{\Sigma} \subset \widehat{\Sigma}$, then a^- can be used to decrypt $\{[m]\}_{a^+}$, by rule (12). Then, being m a name, by rule (17), $m \in \overline{\Sigma}$.
- $\{[m]\}_{a^+} \in \overline{\Sigma}$, by rule (22), because $a^+ \notin \widehat{\Sigma}$, so there is no way to construct $\{[m]\}_{a^+}$ starting from other members of $\overline{\Sigma}$.

Before discussing the basic properties of $\overline{\Sigma}$, let us preliminarily define $\mathtt{r}(\sigma, \Sigma)$ as the boolean value obtained by executing the following algorithm:

```
boolean r(σ, Σ) {
  if σ ∈ Σ                then return TRUE;
  else if σ = suc(σ1)     then return r(σ1, Σ);
  else if σ = (σ1, σ2)    then return r(σ1, Σ) ∧ r(σ2, Σ);
  else if σ = {σ1}σ2      then return r(σ1, Σ) ∧ r(σ2, Σ);
  else if σ = H(σ1)       then return r(σ1, Σ);
  else if σ = {[σ1]}σ2+   then return r(σ1, Σ) ∧ r(σ2+, Σ);
  else if σ = [{σ1}]σ2-   then return r(σ1, Σ) ∧ r(σ2-, Σ);
  else if σ = σ1+         then return r(σ1, Σ);
  else if σ = σ1-         then return r(σ1, Σ);
  else (σ ∈ A \ Σ)        return FALSE;
}
```

Informally, this algorithm recursively checks whether σ can be deduced from the set Σ using *introduction* rules (2)-(7) only; in this respect, an introduction rules is a rule that builds a new term by introducing an operator between one or more simpler terms. The basic properties of $\overline{\Sigma}$ are then expressed by the following theorems:

Theorem 1. *(Finiteness) For each finite set of terms $\Sigma \subseteq \mathcal{M}(\mathcal{A})$, $\overline{\Sigma}$ is finite.*

Proof. By inspection of rules (17)-(25) it is clear that any structured element of $\overline{\Sigma}$ has at least a subterm that does not belong to $\widehat{\Sigma}$, i.e. a subterm that cannot be built by combining the elements of Σ. This means that any element of $\overline{\Sigma}$ is necessarily a subterm of an element of Σ. But since Σ is finite, the subterms of its elements are also finite. This implies that $\overline{\Sigma}$ is finite. □

Theorem 2. *(Minimality) Let $\Sigma \subseteq \mathcal{M}(\mathcal{A})$ be a finite set of terms, and $\sigma \in \overline{\Sigma}$. Then $(\overline{\Sigma \setminus \{\sigma\}}) \subset \widehat{\Sigma}$.*

Proof. From rules (1)-(14) it is clear that $(\overline{\Sigma \setminus \{\sigma\}}) \subseteq \widehat{\Sigma}$. So, it is enough to show that there is at least an element of $\widehat{\Sigma}$ that does not belong to $(\overline{\Sigma \setminus \{\sigma\}})$. Such an element is σ. It belongs to $\widehat{\Sigma}$ by rule (1), but it does not belong to $(\overline{\Sigma \setminus \{\sigma\}})$ because by rules (17)-(25) it is either a name or a structured term with at least a subterm that cannot be built from the elements of $\widehat{\Sigma}$ (which includes $\widehat{\Sigma}$ by definition). □

Theorem 3. *(Decidability) Let $\sigma \in \mathcal{M}(\mathcal{A})$ be any finite term, $\Sigma \subseteq \mathcal{M}(\mathcal{A})$ be a finite set of terms, and let us assume that $\overline{\Sigma}$ is known. Then, determining if $\sigma \in \widehat{\Sigma}$ is decidable.*

Proof. We claim that $\sigma \in \widehat{\Sigma}$ iff $\mathtt{r}(\sigma, \overline{\Sigma})$ is true. This claim can be proved by induction, proving directly the cases $\sigma \in \overline{\Sigma}$ and $\sigma \in \mathcal{A} \setminus \overline{\Sigma}$, and proceeding inductively for the other cases. Once the claim is proved, it remains to be shown that the computation of $\mathtt{r}(\sigma, \overline{\Sigma})$ takes a finite number of steps, but this descends directly from the fact that σ and $\overline{\Sigma}$ are finite, and at each recursion step the function $\mathtt{r}$ is invoked on proper subterms of its argument σ. □

The minimal closure seed $\overline{\Sigma}$ enjoys some additional properties, that make it a good candidate as a finite and minimized representation of the term generation capabilities of an intruder that has learned the set of terms Σ:

Theorem 4. *(Closure preservation) For each finite set of terms $\Sigma \subseteq \mathcal{M}(\mathcal{A})$, $\widehat{\overline{\Sigma}} = \widehat{\Sigma}$.*

Theorem 5. *(Computability) For each finite set of terms $\Sigma \subseteq \mathcal{M}(\mathcal{A})$, $\overline{\Sigma}$ can be computed in a finite number of steps.*

An obvious corollary of the above theorems is:

Corollary 1. *Let $\sigma \in \mathcal{M}(\mathcal{A})$ be any finite term and $\Sigma \subseteq \mathcal{M}(\mathcal{A})$ be a finite set of terms. Then, determining whether $\sigma \in \widehat{\Sigma}$ is decidable.*

Proof. We claim that the computation of $\overline{\Sigma}$ from Σ can be carried out by repeatedly applying closure rules (1)-(14). More precisely, let us define a *reduction rule* as a triple $R = \langle \Sigma_I,\ C,\ \Sigma_O \rangle$, where Σ_I and Σ_O are sets of terms representing respectively premises and conclusions of closure rule C.

Applying reduction step R to a finite set of terms Σ means eliminating the premises from and adding the conclusions to Σ. This is written $\Sigma \xrightarrow{R} \Sigma'$, where $\Sigma' = (\Sigma \setminus \Sigma_I) \cup \Sigma_O$ is the resulting set.

Given a finite set of terms Σ, a *reduction of* Σ is a finite sequence of application of reduction rules R_i to finite sets of terms Σ_i, denoted:

$$\Sigma_0 \xrightarrow{R_0} \Sigma_1 \xrightarrow{R_1} \Sigma_2 \ \cdots\ \Sigma_{k-1} \xrightarrow{R_{k-1}} \Sigma_k \ ,$$

such that $\Sigma_0 = \Sigma$ and $R_i \in \mathcal{R}(\Sigma_i)$, where $\mathcal{R}(\Sigma_i)$ is the set of reduction rules whose pre-conditions are satisfied by Σ_i. Below, the notation $a \to b$ means that if the pre-condition a is true, then the reduction rule b can be applied in Σ_i, that is, $b \in \mathcal{R}(\Sigma_i)$. The set $\mathcal{R}(\Sigma_i)$ is the least set such that the following relations hold:

$$\mathrm{H}(\sigma) \in \Sigma_i \ \wedge\ \mathtt{r}(\sigma, \Sigma_i) \to \langle \{\mathrm{H}(\sigma)\}, (5), \emptyset \rangle \tag{26}$$

$$\begin{aligned} \{[\sigma_1]\}_{\sigma_2^+} \in \Sigma_i \\ \wedge\, \mathtt{r}(\sigma_1, \Sigma_i) \ \wedge\ \mathtt{r}(\sigma_2^+, \Sigma_i) \end{aligned} \to \langle \{\{[\sigma_1]\}_{\sigma_2^+}\}, (6), \emptyset \rangle \tag{27}$$

$$\begin{aligned} [\{\sigma_1\}]_{\sigma_2^-} \in \Sigma_i \\ \wedge\, \mathtt{r}(\sigma_1, \Sigma_i) \ \wedge\ \mathtt{r}(\sigma_2^-, \Sigma_i) \end{aligned} \to \langle \{[\{\sigma_1\}]_{\sigma_2^-}\}, (7), \emptyset \rangle \tag{28}$$

$$\sigma^+ \in \Sigma_i \ \wedge\ \mathtt{r}(\sigma, \Sigma_i) \to \langle \{\sigma^+\}, (8), \emptyset \rangle \tag{29}$$

$$\sigma^- \in \Sigma_i \ \wedge\ \mathtt{r}(\sigma, \Sigma_i) \to \langle \{\sigma^-\}, (8), \emptyset \rangle \tag{30}$$

$$\mathrm{suc}(\sigma) \in \Sigma_i \to \langle \{\mathrm{suc}(\sigma)\}, (9), \{\sigma\} \rangle \tag{31}$$

$$(\sigma_1, \sigma_2) \in \Sigma_i \to \langle \{(\sigma_1, \sigma_2)\}, (10), \{\sigma_1, \sigma_2\} \rangle \tag{32}$$

$$\{\sigma_1\}_{\sigma_2} \in \Sigma_i \ \wedge\ \mathtt{r}(\sigma_2, \Sigma_i) \to \langle \{\{\sigma_1\}_{\sigma_2}\}, (11), \{\sigma_1\} \rangle \tag{33}$$

$$\begin{aligned} \{[\sigma_1]\}_{\sigma_2^+} \in \Sigma_i \\ \wedge\, \mathtt{r}(\sigma_2^-, \Sigma_i) \ \wedge\ \neg\mathtt{r}(\sigma_1, \Sigma_i) \end{aligned} \to \langle \{\{[\sigma_1]\}_{\sigma_2^+}\}, (12), \{\sigma_1, \{[\sigma_1]\}_{\sigma_2^+}\} \rangle \tag{34}$$

$$\begin{aligned} [\{\sigma_1\}]_{\sigma_2^-} \in \Sigma_i \\ \wedge\, \mathtt{r}(\sigma_2^+, \Sigma_i) \ \wedge\ \neg\mathtt{r}(\sigma_1, \Sigma_i) \end{aligned} \to \langle \{[\{\sigma_1\}]_{\sigma_2^-}\}, (13), \{\sigma_1, [\{\sigma_1\}]_{\sigma_2^-}\} \rangle \tag{35}$$

$$\sigma^+ \in \Sigma_i \ \wedge\ \sigma^- \in \Sigma_i \to \langle \{\sigma^+, \sigma^-\}, (14), \{\sigma\} \rangle \tag{36}$$

It can be shown by inspection that reductions preserve closures, i.e. that the following proposition holds:

Proposition 1. *if* $\Sigma \xrightarrow{R} \Sigma'$ *is a one-step reduction, then* $\widehat{\Sigma} = \widehat{\Sigma'}$.

Our initial claim can be proved by proving the following proposition:

Proposition 2. *Given a finite set of terms* Σ*, there exists a finite reduction of* Σ*:*

$$\Sigma = \Sigma_0 \xrightarrow{R_0} \Sigma_1 \ \cdots \ \Sigma_{k-1} \xrightarrow{R_{k-1}} \Sigma_k \ ,$$

such that $\Sigma_k = \overline{\Sigma}$.

A reduction that leads from Σ to $\overline{\Sigma}$ can be found if we keep applying reduction rules $R_i \in \mathcal{R}(\Sigma_i)$ as long as some can be applied. It can be verified by inspection that reduction rules $R_i \in \mathcal{R}(\Sigma_i)$ always add subterms of terms that are already included in Σ_i, and that the application of each $R_i \in \mathcal{R}(\Sigma_i)$ cannot produce loops. Since Σ is finite, it is guaranteed that in a finite number of steps we reach a Σ_k on which no reduction rule can be applied. When this happens, $\Sigma_k = \overline{\Sigma}$, because all the pre-conditions of relations (26)-(36) are false, which implies that Σ_k satisfies the minimal closure seed definition predicates (17)-(25).

Theorem 4 directly descends from propositions 1 and 2, while theorem 5 descends from the above two propositions and from the fact that the computation of $\mathtt{r}(\sigma, \Sigma_i)$ takes a finite number of steps. □

In analogy with the natural deduction system, and unlike [5], we allow *both* $\mathcal{I}$ and $\mathcal{E}$ rules to be applied in the computation of $\overline{\Sigma}$ from Σ, under the constraint of their pre-condition and at the expense of a greater computational complexity, which will be analyzed in section 4. However, as entailed by these theorems, this approach does neither sacrifice decidability nor computability.

Theorems 1, 2 and 4 state that the closure seed representation of a finite set of terms Σ is finite and is the minimal set of terms having the same closure of Σ, where minimality means that any element cannot be built from the other ones by means of term composition operators only, i.e. there are no redundant elements. This is a significant departure from the approach of [11,14], whose methods accumulate all the terms the intruder knows about without aiming to minimize their representation.

The proof of theorems 4 and 5 entails that if a new term ρ is added to a minimal closure seed $\overline{\Sigma}$, e.g. as a consequence of an output process, the new minimal closure seed $\overline{\overline{\Sigma} \cup \{\rho\}}$ can be incrementally computed by a reduction that starts from $\overline{\Sigma} \cup \{\rho\}$, without restarting from scratch; it can be expected that the incremental computation is far less expensive in terms of computing power.

In general, the net effect of such a reduction is to eliminate some elements from and add some other new elements to the former $\overline{\Sigma}$. We denote $\delta^{-}_{\overline{\Sigma}}(\rho)$ the set of eliminated elements and $\delta^{+}_{\overline{\Sigma}}(\rho)$ the set of added elements. Formally:

$$\delta^{-}_{\overline{\Sigma}}(\rho) = \overline{\Sigma} \setminus \left(\overline{\overline{\Sigma} \cup \{\rho\}}\right) \tag{37}$$

$$\delta^{+}_{\overline{\Sigma}}(\rho) = \left(\overline{\overline{\Sigma} \cup \{\rho\}}\right) \setminus \overline{\Sigma} \tag{38}$$

Let us now define formally how the intruder's knowledge representation is updated when a new term ρ is received by the intruder: an algorithm that computes the updated intruder's knowledge $\overline{\Sigma'}$ is the one that computes a reduction of

$\overline{\Sigma} \cup \{\rho\}$, thus determining $\delta^{+}_{\overline{\Sigma}}(\rho)$ and $\delta^{-}_{\overline{\Sigma}}(\rho)$ as sketched in the proof of theorems 4 and 5, then computes $\overline{\Sigma'} = (\overline{\Sigma} \cup \delta^{+}_{\overline{\Sigma}}(\rho)) \setminus \delta^{-}_{\overline{\Sigma}}(\rho)$.

We have now shown that $\overline{\Sigma}$ is an adequate, minimal representation of the intruder's knowledge, it is incrementally computable, and the question $\sigma \in \widehat{\Sigma}$ is decidable $\forall \sigma \in \mathcal{M}(\mathcal{A})$. So, given an intruder knowledge $\overline{\Sigma}$ and a finite term $\sigma \in \mathcal{M}(\mathcal{A})$, we can say that the intruder can produce σ iff $\sigma \in \widehat{\overline{\Sigma}}$, i.e. iff $\mathtt{r}(\sigma, \overline{\Sigma})$.

4 Computational Complexity

4.1 On the Computation of the Question $\sigma \in \Sigma$

In this and in the following sections, let $\mathrm{op}(\sigma)$ be the number of operators in term σ and $\mathrm{n}(\Sigma)$ the number of elements in set Σ. Moreover, we extend the domain of the operator $\mathrm{op}(\cdot)$ to sets of terms, by defining it as:

$$\mathrm{op}(\Sigma) = \max_{\sigma \in \Sigma}(\mathrm{op}(\sigma))$$

in that case. Assuming that the comparison between atomic terms can be carried out in constant time, and the lookup of a term in Σ is sequential, then the computational complexity to check whether $\sigma \in \Sigma$ is $O(nm)$, where $n = \mathrm{n}(\Sigma)$ and $m = \mathrm{op}(\sigma)$.

4.2 On the Computation of $\mathtt{r}(\sigma, \Sigma)$

The worst case happens when the operator's tree in σ is fully unbalanced, that is, each invocation of $\mathtt{r}(\sigma, \Sigma)$ on a compound term σ with m operators entails the recursive computation of $\mathtt{r}(\sigma_1, \Sigma)$ and $\mathtt{r}(\sigma_2, \Sigma)$, where σ_1 is atomic, and σ_2 has $m - 1$ operators.

In this case, each recursion step executes in $O(nm)$, where $n = \mathrm{n}(\Sigma)$ and $m = \mathrm{op}(\sigma)$ as shown above, and the recursion depth is m. So, the computational complexity of $\mathtt{r}(\sigma, \Sigma)$ is $O(nm^2)$, as it has also been proven in [13], in the more general framework of local inference rule sets.

4.3 On the Incremental Computation of $\overline{\Sigma}$

For the sake of this discussion, and without loss of generality, let us define a *reduction step* as the simultaneous application of all independent reduction rules and let us denote it with $\rightarrow$. The incremental reduction of $\overline{\Sigma}$ after the addition of the new term ρ can be seen as a finite sequence of reduction steps starting from $\overline{\Sigma} \cup \{\rho\} = \Sigma_0$; reduction step i acts on set Σ_i and produces the (partially) reduced set Σ_{i+1}:

$$\overline{\Sigma} \cup \{\rho\} = \Sigma_0 \rightarrow \ldots \rightarrow \Sigma_i \rightarrow \Sigma_{i+1} \rightarrow \ldots$$

Let $n_i = \mathrm{n}(\Sigma_i)$ be the number of terms in Σ_i, and $m_i = \mathrm{op}(\Sigma_i)$ the maximum number of operators of terms in Σ_i. Then, we have the initial condition:

$$\begin{cases} n_0 = \mathrm{n}(\overline{\Sigma} \cup \{\rho\}) \\ m_0 = \mathrm{op}(\overline{\Sigma} \cup \{\rho\}) \end{cases}$$

In the worst case, up to n_i reduction rules can be applied at step i, one for each term in Σ_i; assuming that we can determine which reduction rule must be applied in constant time, each application of such rule entails one invocation of the $\in$ operator and up to two invocations of $\mathtt{r}(\cdot,\cdot)$; therefore, each application of a reduction rule at reduction step i has a computational complexity of $O(n_i m_i) + O(n_i m_i^2) = O(n_i m_i^2)$, and the computational complexity of reduction step i is $O(n_i^2 m_i^2)$.

At each reduction step i, whenever we remove a term a from Σ_i, and add some other terms $a_1 \dots a_n$ derived from it, the added terms will always have one operator less than the term they originated from, that is, $\mathrm{op}(\{a_1 \dots a_n\}) = \mathrm{op}(a) - 1$.

Therefore, as a result of reduction step i we remove n_i terms with m_i operators and add up to $2n_i$ terms with up to $m_i - 1$ operators; rules (34) and (35) are the only exceptions in this respect, because they do not remove any term. However, their application does not lead to the worst-case complexity because they leave in Σ_{i+1} the compound term $\{[\sigma_1]\}_{\sigma_2^+}$ or $\{[\sigma_1]\}_{\sigma_2^-}$ that cannot be further reduced, because $\sigma_1 \in \Sigma_{i+1}$ as a consequence of the application of the rule itself. So, we can write:

$$\begin{cases} n_{i+1} = 2n_i \\ m_{i+1} = m_i - 1 \end{cases}$$

After a maximum of m_0 reduction steps, Σ_{m_0} is reduced to contain only atoms and no further reductions are possible. So the computational complexity of the reduction as a whole is:

$$\sum_{i=0}^{m_0} O(n_i^2 m_i^2) = \sum_{i=0}^{m_0} O((2^i n_0)^2 (m_0 - i)^2) = O(n_0^2 2^{2m_0}) \ . \tag{39}$$

4.4 Comparison with Normal, Natural Deductions

When we assume that encryption keys are atomic, neglect public/private cryptosystems, and restrict our scope to normal, natural deductions only, as in [5], we can replace all invocations of $\mathtt{r}(\sigma, \Sigma)$ in pre-conditions (26)-(36) with the simpler check $\sigma \in \Sigma$, and we can drop out reduction rules (27)-(30) and (34)-(36).

Accordingly, the complexity of a reduction step as defined in the previous section is reduced to $O(n_i m_i)$, because function $\mathtt{r}(\cdot,\cdot)$ is never invoked in this case. So, the complexity of the reduction process as a whole reduces to:

$$\sum_{i=0}^{m_0} O(n_i m_i^2) = \sum_{i=0}^{m_0} O(2^i n_0 (m_0 - i)^2) = O(n_0 2^{m_0}) \ . \tag{40}$$

5 Examples

As an example, let us start with the minimal closure seed

$$\overline{\Sigma} = \{c, \{\{k_1\}_{k_2}\}_{k_3}, \{[m]\}_{{k_1}^+}, {k_1}^+, k_2\} \ ,$$

and let us observe the reduction process described above when the new term $\rho = k_3$ is added; in Tab. 3 the second column lists the contents of the partially reduced sets Σ_i at each reduction step, the next column recalls the reduction rule applied in that step, and the rightmost two columns list the set of elements removed from and added to Σ_i by the application of the rule, denoted Σ_I and Σ_O, respectively.

Table 3. An example of reduction

i	Σ_i	Rule	Σ_I	Σ_O
0	$\{c, \{\{k_1\}_{k_2}\}_{k_3}, \{[m]\}_{{k_1}^+}, {k_1}^+, k_2, k_3\}$	(33)	$\{\{\{k_1\}_{k_2}\}_{k_3}\}$	$\{\{k_1\}_{k_2}\}$
1	$\{c, \{[m]\}_{{k_1}^+}, {k_1}^+, k_2, k_3, \{k_1\}_{k_2}\}$	(33)	$\{\{k_1\}_{k_2}\}$	$\{k_1\}$
2	$\{c, \{[m]\}_{{k_1}^+}, {k_1}^+, k_2, k_3, k_1\}$	(29)	$\{{k_1}^+\}$	$\emptyset$
3	$\{c, \{[m]\}_{{k_1}^+}, k_2, k_3, k_1\}$	(34)	$\emptyset$	$\{m\}$
4	$\{c, \{[m]\}_{{k_1}^+}, k_2, k_3, k_1, m\}$	(27)	$\{\{[m]\}_{{k_1}^+}\}$	$\emptyset$
5	$\{c, k_2, k_3, k_1, m\}$			

In the table, rule applications are serialized, i.e. only one rule is applied at each step for clarity; in an actual implementation, all independent rules can be applied simultaneously, as outlined in the complexity analysis carried out in section 4.

Table 4 presents a reduction involving a non-atomic symmetric key; the initial intruder's knowledge is $\overline{\Sigma} = \{\{k_A\}_{k_S}, \{m\}_{\{k_B\}_{k_A}}, k_B, \mathrm{H}(m)\}$ and the added term is $\rho = k_S$. Note that in the second step, the premises of rule (33) are indeed satisfied, because $\mathtt{r}(\{k_B\}_{k_A}, \Sigma_1)$ is true, even if $\{k_B\}_{k_A} \notin \Sigma_1$.

Table 4. A reduction involving a non-atomic key

i	Σ_i	Rule	Σ_I	Σ_O
0	$\{\{k_A\}_{k_S}, \{m\}_{\{k_B\}_{k_A}}), k_B, \mathrm{H}(m), k_S\}$	(33)	$\{\{k_A\}_{k_S}\}$	$\{k_A\}$
1	$\{\{m\}_{\{k_B\}_{k_A}}, k_B, \mathrm{H}(m), k_S, k_A\}$	(33)	$\{\{m\}_{\{k_B\}_{k_A}}\}$	$\{m\}$
2	$\{k_B, \mathrm{H}(m), k_S, k_A, m\}$	(26)	$\{\mathrm{H}(m)\}$	$\emptyset$
3	$\{k_B, k_S, k_A, m\}$			

6 Concluding Remarks

Most finite [5,6,12] and infinite-state [2,3,4] protocol analysis methods based on model checking restrict encryption operators to atomic keys only.

For example, in [5] and [16], this restriction comes from the adoption of the closure of Σ under $\mathcal{E}$ rules as a representation of the intruder's knowledge.

Other approaches based on theorem proving do not pose this restriction but the tradeoff typically is between incompletness and possible non-termination of the analysis [14].

It is worth noting that support for constructed, non-atomic keys is becoming increasingly important to be able to analyze real-world protocols, since it is common for such protocols to build a symmetric key from shared secrets and other data exchanged between parties during a run of the protocol itself.

Other papers, such as [10,11], relax this restriction but neither explicitly introduce the notion of $\overline{\Sigma}$, that is, the minimized, canonical representation of the intruder's knowledge, nor analyze and exploit its properties.

The free term algebra of [14], too, allows any term to be used as an encryption key for both public-key and symmetric key encryption; however, the attacker's knowledge representation is not minimized and some other slight restrictions are in effect, such as for example the assumption that private keys are never leaked.

This assumption seems quite reasonable, but cannot easily be guaranteed by hand for complex protocols; so, we believe that such property is best checked with the aid of a formal, automated method.

By contrast, our approach does not pose any restriction on the internal structure and construction operators of encryption keys, besides those implied by spi calculus, thus supporting the full term language of spi calculus itself, even though at the expense of a greater computational complexity, as can be seen by comparing equations (39) and (40).

However, we believe that the additional expressive power and flexibility of our method more than outweighs this disadvantage in the range of values found in practice for n_0 and m_0.

Last, it should be noted that, even if we adopted the term syntax of spi calculus in this paper, our method is easily amenable to work with other term representation languages with similar sets of term composition operators.

Acknowledgments. The authors wish to thank the anonymous referees, whose valuable comments and suggestions helped to improve the quality of this paper. This work has been partially funded by the Italian National Research Council, grant number CNRC00FE45.

References

1. M. Abadi, and A. D. Gordon, "A Calculus for Cryptographic Protocols The Spi Calculus", *Digital Research Report*, vol. 149, January 1998, pp. 1–110.

2. R. Amadio, and D. Lugiez, "On the Reachability Problem in Cryptographic Protocols", *Proc. of CONCUR'2000, LNCS 1877*, pp. 380–394, Springer-Verlag, 2000.
3. M. Boreale, R. De Nicola, and R. Pugliese, "Proof Techniques for Cryptographic Processes", *Proc. of the 14th IEEE Symposium Logic In Computer Science (LICS'99)*, IEEE Computer Society Press, pp. 157–166, 1999.
4. M. Boreale, "Symbolic Trace Analysis of Cryptographic Protocols", In *Proc. 28th ICALP*, Vol. 2076 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 667–681, 2001.
5. E. M. Clarke, S. Jha, and W. Marrero, "Using state space exploration and a natural deduction style message derivation engine to verify security protocols", *Proc. of IFIP PROCOMET*, Chapman & Hall, London, 1998, pp. p.87–106.
6. E. M. Clarke, S. Jha, and W. Marrero, "Verifying security protocols with Brutus", ACM Trans. on Software Engineering and Methodology Vol. 9, No. 4, October 2000, pp. 443–487.
7. D. Dolev, and A. Yao, "On the security of public key protocols", *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
8. L. Durante, R. Sisto, and A. Valenzano, "A state-exploration technique for spi-calculus testing equivalence verification", *Proc. of FORTE/PSTV 2000*, Pisa, October 2000, pp. 155–170.
9. L. Durante, R. Sisto, and A. Valenzano, "Automatic testing equivalence verification of spi-calculs specifications", Politecnico di Torino I.R. DAI/ARC 1-02.
10. M. Fiore, and M. Abadi, "Computing Symbolic Models for Verifying Cryptographic Protocols", *Proc. of 14th IEEE Computer Security Foundations Workshop*, pp. 160–173, June 2001.
11. A. Huima, "Efficient Infinite-State Analysis of Security Protocols", *Proc. of FLOC Workshop on Formal Methods and Security Protocols*, 1999.
12. G. Lowe, "Breaking and fixing the Needham-Schroeder public-key protocol using FDR", *Proc. of TACAS'97*, Springer LNCS 1055, 1996.
13. D. A. McAllester, "Automatic Recognition of Tractability in Inference Relations", *Journal of the ACM*, Vol. 40, No. 2, April 1993, pp. 284–303.
14. J. Millen, and V. Shmatikov, "Constraint solving for Bounded-Process Cryptographic Protocol Analysis", *8th ACM Conference on Computer and Communication Security*, pages 166–175, November 2001.
15. R. Milner, J. Parrow, and D. Walker, "A Calculus of mobile processes, parts I and II", *Information and Computation*, pages 1–40 and 41–77, September 1992.
16. L. C. Paulson, "The inductive approach to verifying cryptographic protocols", *Journal of Computer Security*, Vol. 6, pp. 85–128, 1998.
17. D. Prawitz, "Natural Deduction: A Proof-Theoretical Study", Almqvist & Wiskell, 1965.