Pattern-Based Abstraction for Verifying Secrecy in Protocols[⋆]

Liana Bozga, Yassine Lakhnech, and Michael Périn

Verimag
2 av. de Vignate 38610 Gieres, France.
{lbozga,lakhnech,perin}@imag.fr

Abstract. We present a method based on abstract interpretation for verifying secrecy properties of cryptographic protocols. Our method allows to verify secrecy properties in a general model allowing an unbounded number of sessions, an unbounded number of principals and an unbounded size of messages. As abstract domain we use sets of so-called *pattern terms*, that is, terms with an interpreted constructor, *Sup*, where a term *Sup* (t) is meant for the set of terms that contain t as sub-term. We implemented a prototype and were able to verify well-known protocols such as for instance Needham-Schroeder-Lowe (0.02 sec), Yahalom (12.67 sec), Otway-Rees (0.02 sec), Skeme (0.06 sec) and Kao-Chow (0.07 sec).

1 Introduction

At the heart of almost every computer security architecture is a set of cryptographic protocols that use cryptography to encrypt and sign data. They are used to exchange confidential data such as pin numbers and passwords, to authentify users or to guarantee anonymity of principals. It is well known that even under the idealized assumption of perfect cryptography, logical flaws in the protocol design may lead to incorrect behavior with undesired consequences. Maybe the most prominent example showing that cryptographic protocols are notoriously difficult to design and test is the Needham-Schroeder protocol for authentification. It has been introduced in 1978 [23]. An attack on this protocol has been found by G. Lowe using the CSP model-checker FDR in 1995 [18]; and this led to a corrected version of the protocol [19]. Consequently there has been a growing interest in developing and applying formal methods for validating cryptographic protocols [20,8]. Most of this work adopts the so-called Dolev and Yao model of intruders. This model assumes perfect cryptographic primitives and a nondeterministic intruder that has total control of the communication network and capacity to forge new messages. It is known that reachability is undecidable for cryptographic protocols in the general case [13], even when a bound is put on the size of messages [12]. Because of these negative results, from the point of view of verification, the best we can hope for is either to identify decidable sub-classes

⋆ This work has been partially supported by the RNTL project EVA.

as in [3,24,21] or to develop correct but incomplete verification algorithms as in [22,17,15].

In this paper, we present a correct but, in general, incomplete verification algorithm to prove secrecy without putting any assumption on messages nor on the number of sessions. Proving secrecy means proving that secrets are not revealed to unauthorized agents. The main contribution of our paper is an original method for proving that a secret is not revealed by a set of rules that model how the initial set of messages known by the intruder evolves. The method is based on the notion of "the secret being *guarded*, or *kept under the hat* of a safe message". For example, suppose that our secret is the nonce N_B and that the key K_B^{-1} – the inverse of K_B – is not known by the intruder. Then, any message that contains N_B and that is encrypted with K_B is a guard for N_B. For instance, N_B is guarded in the message $\{\{N_A, N_B\}_{K_B}\}_{K_I}$ by the sub-message $\{N_A, N_B\}_{K_B}$. The idea is then to compute a set of guards that will keep the secret unrevealed in all sent messages and such that the inverses of the keys used in the guards are also secrets. The difficulty here is that the set of guards is, in general, infinite. Therefore, we use *terms* to represent sets of guards. For instance the term $\{x, x_s\}_{K_B}$ says that the secret will be guarded in any message $\{m, m'\}_{K_B}$, where the secret is not a sub-message of m [1] but may be a sub-message of m'. The problem is, however, that there might be a rule $\{I, y\}_{K_B} \rightarrow y$ that will send y unencrypted to the intruder if (s)he produces the message $\{I, y\}_{K_B}$. Hence, the term $\{x, x_s\}_{K_B}$ will guard the secret except when x is I. Thus, our abstract domain consists of pairs $(\mathcal{G}, \mathcal{B})$ of terms. Those in $\mathcal{G}$ correspond to good messages that guard the secrets, whereas those in $\mathcal{B}$ denote "bad exceptions", that is, the particular instances of terms in $\mathcal{G}$ that do not guard the secrets. A weakness of terms is, however, that variables appear only at the leafs, and hence, they do not allow to describe, for instance, the set of terms that share a common sub-term. To mitigate this weakness, we introduce *pattern terms*, that is, terms with an interpreted constructor, *Sup*, where a term *Sup*(t) is meant for the set of terms that contain t as sub-term.

We developed a prototype in Caml that implements this method. We have been able to verify several protocols taken from [6] including, for instance, Needham-Schroeder-Lowe (0.02 sec), Yahalom (12.67 sec), Otway-Rees (0.02 sec), Skeme (0.06 sec) and Kao-Chow (0.07 sec).

Related work. Dolev, Even and Karp introduced [10] the class of ping-pong protocols and showed its decidability. The restriction put on these protocols are too restrictive and none of the protocols of [6] falls in this class. Recently, Comon, Cortier and Mitchell [7] extended this class allowing pairing and binary encryption while the use of nonces still cannot be expressed in their model. Reachability is decidable for the bounded number of sessions [3,24,21] or when nonce creation is not allowed and the size of messages is bounded [12]. These assumptions are rarely justified in practice.

[1] unless it is guarded by an other term.

Type systems and type-checking have also been advocated as a method for verifying security protocols (e.g. [1,16,2]). Although, these techniques can handle unbounded protocols they are as far as we know not yet completely automatic. Closest to our work are partial algorithms based on abstract interpretation and tree automata that have been presented in [22,17,15]. The main difference is, however, that we do not compute the set of messages that can be known by the intruder but a set of guards as explained above. Our method can handle unbounded protocols fully automatically with the price that it may discover false attacks. Interesting enough is that does not happen on any of the practical protocols we tried (see Table 1).

2 Preliminary

If $n \in \mathbb{N}$ then we denote by $\mathbb{N}_n$ the set $\{1, \cdots, n\}$. Let $\mathcal{X}$ be a countable set of variables and let $\mathcal{F}^i$ be a countable set of function symbols of arity i, for every $i \in \mathbb{N}$. Let $\mathcal{F} = \bigcup_{i \in \mathbb{N}} \mathcal{F}^i$. The set of *terms over* $\mathcal{X}$ *and* $\mathcal{F}$, denoted by $\mathcal{T}(\mathcal{X}, \mathcal{F})$, is the smallest set containing $\mathcal{X}$ and closed under application of the function symbols in $\mathcal{F}$, i.e., $f(t_1, \cdots, t_n)$ is a term in $\mathcal{T}(\mathcal{X}, \mathcal{F})$, if $t_i \in \mathcal{T}(\mathcal{X}, \mathcal{F})$, for $i = 1, \cdots, n$, and $f \in \mathcal{F}^n$. As usual, function symbols of arity 0 are called constant symbols. *Ground terms* are terms with no variables. We denote by $\mathcal{T}(\mathcal{F})$ the set of ground terms over $\mathcal{F}$.

A tree tr is a function from a finite subset of ω^* to $\mathcal{X} \cup \mathcal{F}$ such that $tr(u) \in \mathcal{F}^n$ iff $u \cdot j \in dom(tr)$, for every $j \in \{0, \cdots, n-1\}$. We identify terms with trees by associating to each term t a tree $Tr(t)$ as follows: (1) if x is a variable, then $dom(Tr(x)) = \{\epsilon\}$ and $Tr(x)(\epsilon) = x$, (2) if f is a constant symbol, then $dom(Tr(f)) = \{\epsilon\}$ and $Tr(f)(\epsilon) = f$ and (3) for a term $t = f(t_0, \cdots, t_{n-1})$, $dom(Tr(t)) = \{\epsilon\} \cup \bigcup_{i=0}^{n-1} i \cdot dom(Tr(t_i))$, where $\cdot$ is word concatenation extended to sets, $Tr(t)(\epsilon) = f$ and $Tr(t)(i \cdot u) = Tr(t_i)(u)$. Henceforth, we tacitly identify the term t with $Tr(t)$. The elements of $dom(t)$ are called *positions* in t. We use $\prec$ to denote the prefix relation on ω^*. We write $t(p)$ to denote the symbol at position p in t and $t_{|p}$ to denote the subterm of t at position p, which corresponds to the tree $t_{|p}(x) = t(p \cdot x)$ with $x \in dom(t_{|p})$ iff $x \cdot p \in dom(t)$.

3 Models for Cryptographic Protocols

In this section, we describe how we model cryptographic protocols and give a precise definition of the properties we want to verify. We begin by describing the messages involved in a protocol model.

3.1 Messages

The set of messages is denoted by $\mathcal{M}$ and contains terms constructed from constant symbols and the function symbols **encr** : $\mathcal{M} \times \mathcal{K} \rightarrow \mathcal{M}$ and **pair** : $\mathcal{M} \times \mathcal{M} \rightarrow \mathcal{M}$. Constant symbols are also called atomic messages and are defined as follows:

1. *Principle names* are used to refer to the principles in a protocol. The set of all principles is $\mathcal{P}$.
2. *Nonces* can be thought as randomly generated numbers. As no one can predict their values, they are used to convince for the freshness of a message. We denote by $\mathcal{N}$ the set of nonces.
3. *Keys* are used to encrypt messages. We have the following atomic keys for each $p_1, \cdots, p_r \in \mathcal{P}^r$ where *pbk*, *pvk* and *smk* stand respectively for *public*, *private* and *symmetric* keys:

$$pbk(p_1, \cdots, p_r) \mid pvk(p_1, \cdots, p_r) \mid smk(p_1, \cdots, p_r).$$

   We denote by $\mathcal{AK}(p_1, \cdots, p_r)$ this set of keys and let $\mathcal{K} = \bigcup_{\boldsymbol{p} \in \mathcal{P}^+} \mathcal{AK}(\boldsymbol{p})$ denote the set of all keys. The key $pbk(p_1, \cdots, p_r)$ is an inverse of the key $pvk(p_1, \cdots, p_r)$ and vice versa; and a key $smk(p_1, \cdots, p_r)$ is its self-inverse. If k is a key then we use k^{-1} to denote its inverse. Moreover, as usual, we write K_A instead of $pbk(A)$, K_A^{-1} instead of $pvk(A)$ and K_{AB} instead of $smk(A, B)$.

For the sake of simpicity we left out the signatures and hash functions but we can easlily handle them in our model.

Let $\mathcal{A} = \mathcal{P} \cup \mathcal{N} \cup \mathcal{K}$ and $\mathcal{F} = \mathcal{A} \cup \{\mathbf{encr}, \mathbf{pair}\}$. As usual, we write (m_1, m_2) for $\mathbf{pair}(m_1, m_2)$ and $\{m\}_k$ instead of $\mathbf{encr}(m, k)$. *Message terms* are the elements of $\mathcal{T}(\mathcal{X}, \mathcal{F})$, that is, terms over the atoms $\mathcal{A}$, a set of variables $\mathcal{X}$ and the binary function symbols **encr** and **pair**. *Messages* are ground terms in $\mathcal{T}(\mathcal{X}, \mathcal{F})$, i.e, $\mathcal{M} = \mathcal{T}(\mathcal{F})$.

3.2 The Intruder Model

In this section, we describe how an intruder can create new messages from already known messages. We use the most commonly used model, introduced by Dolev and Yao [11], which is given by a formal system $\vdash$. The intruder capabilities for intercepting messages and sending (fake) messages are fixed by the operational semantics. Thus, the *derivability of a message* m from a set E of messages, denoted by $E \vdash m$, is described by the following axiom and rules:

– If $m \in E$ then $E \vdash m$.
– If $E \vdash m_1$ and $E \vdash m_2$ then $E \vdash \mathbf{pair}(m_1, m_2)$. This rule is called pairing.
– If $E \vdash m$ and $E \vdash k \in \mathcal{K}$ then $E \vdash \mathbf{encr}(m, k)$. This is called encryption.
– If $E \vdash \mathbf{pair}(m_1, m_2)$ then $E \vdash m_1$ and $E \vdash m_2$. This is called projection.
– If $E \vdash \mathbf{encr}(m, k)$, $E \vdash k'$ and k and k' are inverses then $E \vdash m$. This is called decryption.

Notations. For a term t, we use the notation $E \not\vdash t$ to denote that no instance of t is derivable from E, that is, for no substitution $\sigma : \mathcal{X} \to \mathcal{M}$, we have $E \vdash \sigma(t)$.

We now define *critical* and *non-critical* positions in a message. The idea is that since there is no way to deduce from an encrypted message the key with which it has been encrypted, the key position in messages of the form $\mathbf{encr}(m, k)$

is not critical[2]. Formally, given a term t, a position p in t is called *non-critical*, if there is a position q such that $p = q \cdot 2$ and $t(q) = \mathbf{encr}$; otherwise it is called *critical*. We will also use the notation $s \in_c m$ to denote that s appears in m at a critical position, i.e., there exists $p \in dom(m)$ such that p is critical and $m_{|p} = s$.

We also use the notation $E \not\vdash^{\in_c} t$ to denote that no message derivable from E contains an instantiation of t at a critical position, that is, for every message m if $E \vdash m$ then $\sigma(t) \notin_c m$, for any σ. The relation $\not\vdash^{\in_c}$ is naturally extended to set of terms.

3.3 Representation of the Protocol

We use in this paper a simple protocol representation that can be proved to be a safe approximation of a more realistic model [5]. The main idea behind this representation is the following abstraction. First, we fix an arbitrary session where the same principal, say A, plays the (say two) different roles in the protocol. Then, we identify the intruder I with all principles other than A. Moreover, we identify all sessions in which A is not involved. Concerning the other sessions, that is, those where A is involved, we identify:

– all sessions where A plays both roles and which are different from the fixed session,
– all sessions where A plays the first role while the second role is played by an other principal,
– all sessions where A plays the second role while the first role is played by a different principal.

Identifying sessions means also identifying the nonces and keys used in these sessions. This leaves us with a system where we still have unbounded number of messages as the size of the messages is not bounded. To summarize: We model a protocol as a set of transitions that can be taken in any order and any number of times. We also apply the safe and exact abstraction that consists in considering only one honest principal and one dishonest principal (the intruder). The proof that this abstraction is safe and actually also exact is given in [9].

Thus, a protocol is represented by a pair $(\mathcal{C}, \mathcal{R})$ consisting of:

– a predicate $\mathcal{C}$ on sets of messages describing the messages that the intruder may initially know, and
– a finite set $\mathcal{R}$ of rule schemata of the form $t_1 \to t_2$, where the t_i's are terms.

The constraint $\mathcal{C}$ can be used to describe freshness properties. For instance, if N is a nonce then the condition $E \not\vdash^{\in_c} N$ states that N is fresh with respect to E. In a companion paper [5], we describe how an abstract procotol description $(\mathcal{C}, \mathcal{R})$ is derived from a concrete one by abstract interpretation.

Notice that we do not lose generality by considering transitions of the form $t \to t'$ instead of transitions with more than one term on the left-hand side, as

[2] For the insider, the critical position corresponds, for instance, to the subterm relation in the strand space model [14,25].

a transition $t_1, \cdots, t_n \to t'$ can be encoded as $(t_1, (t_2, \cdots, (t_{n-1}, t_n) \cdots) \to t'$. Actually, our verification tool uses this encoding as it increases precision of the results.

Let us turn our attention to the semantics of a protocol $(\mathcal{C}, \mathcal{R})$. A *run* of $(\mathcal{C}, \mathcal{R})$ is given by a finite sequence of sets of messages E_i of the form

$$E_0 \xrightarrow{r_1} E_1 \cdots E_{n-1} \xrightarrow{r_n} E_n$$

with $n \geq 0$ such that E_0 satisfies the constraint $\mathcal{C}$ and such that, for each $i = 1, \cdots n$, there is a substitution $\rho_i : \mathcal{X} \to \mathcal{M}$ such that $E_{i-1} \vdash \rho(t_1)$ and $E_i = E_{i-1} \cup \{\rho(t_2)\}$, where $t_1 \to t_2 = r_i$.

In other words, considering E_{i-1} as the current knowledge of the intruder, a rule $t_1 \to t_2 = r_i$ can be taken if there is an instance $\rho(t_1)$ of t_1 that is derivable from E_{i-1}. The effect of applying the rule culminates in adding the message $\rho(t_2)$ to the knowledge of the intruder. Notice how following Bolignano [4] communication is modeled through the knowledge of the intruder. That is, the intruder can intercept messages use them to create fake messages and deliver these to the principals.

3.4 Secrecy Modeling

Our goal is to determine whether secrecy property holds: for instance whether the intruder can get a secret s. That is, does a run exist that leads to a set E of messages from which s is derivable.

A *secret* is given by a message m. A protocol P satisfies the secrecy property defined by m, denoted by $\not\vdash_P m$, if it does not admit any run $E_0 \xrightarrow{r_1} E_1 \cdots E_{n-1} \xrightarrow{r_n} E_n$ such that $E_n \vdash m$. The definition of secrecy can be pointwisely extended to a set of secrets.

Example 1. The Needham-Schroeder protocol for authentification can be described as follows using the usual informal notation for cryptographic protocols:

$$\begin{array}{l} A \to B : \{A, N_1\}_{K_B} \\ B \to A : \{N_1, N_2\}_{K_A} \\ A \to B : \{N_2\}_{K_B} \end{array}$$

Intuitively, A plays the role of the initiator of the session; while B is a responder.

In our model which yields an over-approximation of the possible runs of the protocol, we can describe the Needham-Schroeder protocol by the following rules. We write $\frac{t_1}{t_2}$ *instead of* $t_1 \to t_2$.

session (A, I)	*session* (I, A)	*the fixed session* (A, A)	*other sessions* (A, A)
$\frac{-}{\{A,N_1^{AI}\}_{pbk(I)}}$;	$\frac{-}{\{I,N_I\}_{pbk(A)}}$;	$\frac{-}{\{A,N_1\}_{pbk(A)}}$;	$\frac{-}{\{A,N_1^{AA}\}_{pbk(A)}}$;
$\frac{\{A,y\}_{pbk(I)}}{\{y,N_I\}_{pbk(A)}}$;	$\frac{\{I,y\}_{pbk(A)}}{\{y,N_2^{IA}\}_{pbk(I)}}$;	$\frac{\{A,y\}_{pbk(A)}}{\{y,N_2\}_{pbk(A)}}$;	$\frac{\{A,y\}_{pbk(A)}}{\{y,N_2^{AA}\}_{pbk(A)}}$;
$\frac{\{N_1^{AI},z\}_{pbk(A)}}{\{z\}_{pbk(I)}}$;	$\frac{\{N_I,z\}_{pbk(I)}}{\{z\}_{pbk(A)}}$;	$\frac{\{N_1,z\}_{pbk(A)}}{\{z\}_{pbk(A)}}$;	$\frac{\{N_1^{AA},z\}_{pbk(A)}}{\{z\}_{pbk(A)}}$;

Each rule corresponds to a transition of the Needham-Schroeder protocol instantiated w.r.t. the nonces and the principals of an abstract session.

Beyond these rules, the verification problem is defined by a constraint $\mathcal{C}(E)$ on the initial knowledge of the intruder, e.g., $E \not\vdash^{\in_c} \{\ N_1,\ N_2,\ pvk(A)\ \}$ and a secrecy property defined by the set of messages $\{N_2, pvk(A)\}$.

4 Verification Based on Patterns Keeping Secrets

Throughout this section we assume that we are given a protocol $P = (\mathcal{C}, \mathcal{R})$ and a set of secrets defined by a set $\mathcal{S}$ of messages. We present an algorithm that allows to verify that a protocol preserves a set of secrets.

4.1 Hat-Messages: Messages Keeping Secrets under a Hat

If a principal A wants to protect a secret s, then he has to use a key whose inverse is not known by the intruder in order to encrypt every occurence of s in every message sent. The secret s itself need not be directly encrypted. Indeed, it should be enough that it only appears as part of encrypted messages. The basic idea of our method is to compute the set of encrypted messages that protect the secrets. As we will see, encryption with a safe key is not always sufficient to protect a secret in every message, as honest principals following the protocol can unwillingly help the intruder in decrypting the message.

In order to develop this idea formally we need to introduce a few definitions. Recall that critical and non-critical positions as well as the notation $\in_c$ has been introduced in Section 3.2.

Definition 1. *We call* hat-term *any term of the form* $\mathbf{encr}(t, k)$*. It is called* hat-message *if it is a ground term. Then, a secret s is protected by a set of hat-messages H in a set of messages M, denoted by $M\langle H\rangle s$, if the following condition is satisfied:*

> *for all messages $m \in M$, for all critical positions p in m with $m_{|p} = s$, there exists a position $q \prec p$ such that $m_{|q} \in H$.* □

Example 2. According to our definition, the hat-message $\mathbf{encr}(s, k)$ protects the secret s in the messages $\mathbf{encr}(\mathbf{encr}(s, k), k')$ but it does not protect it neither in the message $\mathbf{encr}(\mathbf{encr}(s, k'), k)$ nor in $\mathbf{pair}(\mathbf{encr}(s, k), s)$. Furthermore, even if the key k is part of the secrets, it does not need to be protected in the previous messages for it never appears in critical position. □

The notion of a message being protected by a hat-message introduced above does not take into account the capabilities of the intruder to decompose and compose new messages. We have to give particular care to the treatment of composed secrets as they can be obtained either by composition or decomposition. To do so, let us define the weak closure under decomposition of a term. Let T be a set terms. Then, T is called *weakly closed under decomposition*, or just weakly closed for short, if for any term $f(t_1, \cdots, t_n) \in T$ there is some $i \in \mathbb{N}_n$ such that $t_i \in T$. Then, a set T' is called a *weak closure* of T, if it is weakly closed, contains T and no proper sub-set of it satisfies these properties. Now, we are ready to extend the notion of a message being protected by a hat-message taking into account the intruder.

Definition 2. *A set $\mathcal{S}$ of messages is called* strongly protected *by $\mathcal{H}$ in M, denoted by $M[\mathcal{H}]\mathcal{S}$, if the following conditions are satisfied:*

1. *$M \vdash m \Rightarrow m\langle\mathcal{H}\rangle\mathcal{S}$, i.e., the secrets in $\mathcal{S}$ must be protected in any message that the intruder can deduce from M.*
2. *$\mathcal{S}$ is weakly closed.*
3. *$Keys(\mathcal{H})^{-1} \subseteq \mathcal{S}$ where $Keys(H)^{-1} = \{k^{-1} \mid \mathbf{encr}(t, k) \in H\}$, i.e., the inverses of all the keys used in hat-messages are also secrets.* □

Intuitively, Condition (2) ensures that the intruder will always miss at least one part of a secret preventing him from deducing it by composition. Condition (3) ensures that the intruder will not be able to decrypt a secret protected by a hat-message.

Our main motivation in introducing $M[\mathcal{H}]\mathcal{S}$ lays in the following proposition that states that adding a message m in which the secrets of $\mathcal{S}$ are protected preserves strong protection of $\mathcal{S}$.

Proposition 1. *Let $M, \mathcal{S} \subseteq \mathcal{T}(\mathcal{F})$ be sets of messages, $\mathcal{H}$ be a set of hat-messages and m be a message. If $M[\mathcal{H}]\mathcal{S}$ and $m\langle\mathcal{H}\rangle\mathcal{S}$ then $M \cup \{m\}[\mathcal{H}]\mathcal{S}$.* □

The proof of this proposition is presented in [5]; it does not rely on the fact that keys are atomic. Actually, our method can be extended to cover the case of non atomic keys.

Let now $r = t_1 \rightarrow t_2$ be a rule in $\mathcal{R}$ (recall that we have fixed an arbitrary protocol $P = (\mathcal{C}, \mathcal{R})$. We say that the pair $(\mathcal{S}, \mathcal{H})$ composed of a set of secrets and a set of hat-messages is *stable w.r.t. a rule* r, if for every subtitution σ, we have $\sigma(t_2)\langle\mathcal{H}\rangle\mathcal{S}$. It is stable w.r.t. a set of rules $\mathcal{R}$ if it stable w.r.t. to each rule in $\mathcal{R}$. Then, using Proposition 1, we can prove by induction the following theorem:

Theorem 1. *Let $\mathcal{S}$ be a set of secrets and $\mathcal{H}$ be a set of hat-messages. If $(\mathcal{S}, \mathcal{H})$ is stable w.r.t. all rules in $\mathcal{R}$ and $E_0[\mathcal{H}]\mathcal{S}$, for every set of messages E_0 that satisfies $\mathcal{C}$, then $\not\vdash_P \mathcal{S}$, i.e., the secrets in $\mathcal{S}$ are preserved by P.*

According to Theorem 1, given a protocol $P = (\mathcal{C}, \mathcal{R})$ and a set $\mathcal{S}$ of secrets, if we can find a set $\mathcal{H}$ of hat-messages and a set $\mathcal{S}'$ of secrets such that:

– the constraint $\mathcal{C}$ on E_0 – the messages initially known by the intruder – implies $E_0[\mathcal{H}]\mathcal{S}'$,
– $\mathcal{S} \subseteq \mathcal{S}'$, and
– $(\mathcal{S}', \mathcal{H})$ is stable w.r.t. $\mathcal{R}$.

we can conclude that the secrets in $\mathcal{S}$ are preserved by $P = (\mathcal{C}, \mathcal{R})$. In this section, we develop an algorithm that computes a stable pair $(\mathcal{S}', \mathcal{H})$. This is done in two steps. First, we develop a semantic version of the algorithm in which we do not consider questions related to representing sets of hat-messages. Then, we define a symbolic representation for hat-messages and develop a symbolic algorithm.

4.2 A Semantic Version of the Verification Algorithm

In Figure 1, we present an algorithm that computes a pair $(\mathcal{S}, \mathcal{H})$ which is stable w.r.t. the rules of the protocol. It uses the function *Closure* that associates to a set T of terms a weak closure of T. The algorithm takes a set of rules $\mathcal{R}$, a set of secrets $\mathcal{S}$ and a set of hat-messages $\mathcal{H}$ as input. It is a fixpoint computation, starting with $(\mathcal{S}, \mathcal{H})$. If it terminates, it returns an augmented set of secrets $\mathcal{S}'$ and a set $\mathcal{H}'$ such that $\mathcal{H}' \subseteq \mathcal{H}$. We now explain intuitively the clue point of the algorithm. Let us take a rule $t_p \rightarrow t_c$ in $\mathcal{R}$, a substitution $\sigma : \mathcal{X} \rightarrow \mathcal{T}(\mathcal{F})$ and a hat-message h such that h protects a secret in $\sigma(t_p)$. If the secret is not protected by any hat-messages in $\sigma(t_c)$ – the conclusion of the instanciated rule, then the hat-message h is not safe indeed and it must be removed from the set of hat-messages. Actually, even when the inverse of the keys used in the hat-messages are unknown by the intruder, the secret can be unwillingly revealed by a principal. Think for instance of a protocol with $\{(y, x)\}_{pbk(A)} \rightarrow \{x\}_{pbk(y)}$ as a rule of principal A. On reception of the message $\{(I, \mathit{Secret})\}_{pbk(A)}$, the principal A will respond $\{\mathit{Secret}\}_{pbk(I)}$ thus unwillingly decrypting the secret for the intruder. So, $\{(I, \mathit{Secret})\}_{pbk(A)}$ is a particular case where $pbk(A)$ does not protect the secret and it must be removed from the set of hat-messages. Case 2 in the algorithm considers the case where a secret is not protected in the conclusion and the premisse does not contain a secret. In this case, the apparently harmless premise is as compromising as the secret, and hence, is added to the set of secrets. The following proposition summarizes the properties of the algorithm.

Proposition 2. *If the algorithm of Figure 1 applied to $(\mathcal{R}, \mathcal{S}, \mathcal{H})$ terminates, it returns $\mathcal{S}'$ and $\mathcal{H}'$ which satisfy the following conditions:*

1. *$(\mathcal{S}', \mathcal{H}')$ is stable w.r.t. $\mathcal{R}$,*
2. *$\mathcal{S} \subseteq \mathcal{S}'$, and*
3. *$Keys(\mathcal{H}')^{-1} \subseteq \mathcal{S}'$.* □

Using Proposition 2 and Theorem 1, we can prove the following corollary.

Corollary 1. *If the algorithm of Figure 1 terminates with $(\mathcal{S}', \mathcal{H}')$ as result and each set of messages E_0 that satisfies $\mathcal{C}(E_0)$ also satisfies $E_0[\mathcal{H}']\mathcal{S}'$, we can conclude $\not\models_P \mathcal{S}'$, and hence, $\not\models_P \mathcal{S}$.* □

4.3 A Symbolic Representation of Hat-Messages: Pattern Terms

To develop an effective version of our semantic algorithm, we need to represent (potentially infinite) sets of hat-messages. To do so, we introduce a symbolic representation that consists in a pair $(\mathcal{G}, \mathcal{B})$ of sets of terms over variables in $\mathcal{X} \cup \mathcal{X}_s$. Here, $\mathcal{X}_s$ denotes a new disjoint set of variables. We use $x_s, y_s, \cdots$ as typical variables in $\mathcal{X}_s$. Roughly speaking, a hat-message for a secret is an instance of a term in $\mathcal{G}$ that is not an instance of any term in $\mathcal{B}$. Therefore, we call the terms in $\mathcal{G}$ *good patterns* and those in $\mathcal{B}$ *bad patterns*. In good patterns, we use the variables in $\mathcal{X}_s$ to mark the positions in which a term containing a secret is

```
**input**: $\mathcal{R}$, a weakly closed $\mathcal{S}$ and $\mathcal{H}$
**output**: $\mathcal{S}'$, $\mathcal{H}'$ such that $(\mathcal{S}',\mathcal{H}')$ is stable w.r.t. $\mathcal{R}$.
  $\mathcal{S}' := \mathcal{S}$ ; $\mathcal{H}' := \mathcal{H}$ ;
repeat
  — first, add to the secrets the inverse of the keys used in hat-messages then,
  — compute the closure that adds to $S'$ one subpart of each compound secret of $\mathcal{S}'$
  $\mathcal{S}' := \mathcal{S}' \cup Keys(\mathcal{H}')^{-1}$ ; $\mathcal{S}' := Closure(\mathcal{S}')$ ; $\mathcal{S}_c := \mathcal{S}'$ ; $\mathcal{H}_c := \mathcal{H}'$ ;
  for each $t_p \to t_c \in \mathcal{R}$
        — compute all Dangerous Substitutions of rule $t_p \to t_c$ where a secret is
        — not kept in the conclusion
    $DS := \{\sigma : \mathcal{X} \to \mathcal{M} \mid \neg\big(\sigma(t_c)\langle\mathcal{H}'\rangle\mathcal{S}'\big)\}$ ;
        — compute the corresponding Dangerous Premisses
    $DP := \{\sigma(t_p) \mid \sigma \in DS\}$ ;
    — update the secret and hat-messages according to the dangerous premisses:
    — case 1 removes the hat-messages that appear in dangerous premisses
    for each $s \in \mathcal{S}'$, $m \in DP$ do
          — the bad hat-messages of $s$ are those which protect a secret
          — that can be disclosed by a rule
      $BHat := \{m_{|q} \mid \exists p.\ q \prec p \wedge m_{|p} = s \wedge m_{|q} \in \mathcal{H}'\}$
          — select a subset of hat-messages to be removed
      choose $BHat' \subseteq BHat$ with $BHat \neq \emptyset \Rightarrow BHat' \neq \emptyset$
          — update the set of hat-message $\mathcal{H}$
      $\mathcal{H}' := \mathcal{H}' \setminus BHat'$ ;
    od
    — case 2 adds to the secrets all bad premisses than do not contain any secret
    $newS := \{m \in DP \mid \forall s \in \mathcal{S}'.\ s \notin_c m\}$ ; $\mathcal{S}' := \mathcal{S}' \cup newS$
  od
until $(\mathcal{S}',\mathcal{H}') = (\mathcal{S}_c,\mathcal{H}_c)$
```

Fig. 1. The semantic version of the verification algorithm

allowed to appear. While in bad patterns, they are used to mark positions where it is not guaranteed that the secret is protected. For instance, if $\mathcal{G}$ contains the term $\{x_s\}_K$ and $\mathcal{B}$ contains the term $\{(A, x_s)\}_K$ then the message $\{(B, Secret)\}_K$ will be in the set represented by $(\mathcal{G}, \mathcal{B})$, while the message $\{(A, Secret)\}_K$ will not. Let us now define this symbolic representation formally.

To do so, we introduce *pattern terms* defined by the following BNF:

$$pt ::= N \mid P \mid K \mid x \mid x_s \mid \mathbf{pair}(pt_1, pt_2) \mid \mathbf{encr}(pt, K) \mid Sup\,(pt)$$

where $N \in \mathcal{N}$, $P \in \mathcal{P}$, $K \in \mathcal{K}$, $x \in \mathcal{X}$, and $x_s \in \mathcal{X}_s$. The set of pattern terms is denoted by $\mathcal{PT}(\mathcal{X} \cup \mathcal{X}_s, \mathcal{F})$. Notice that every term in $\mathcal{T}(\mathcal{X} \cup \mathcal{X}_s, \mathcal{F})$ is also a pattern term in $\mathcal{PT}(\mathcal{X} \cup \mathcal{X}_s, \mathcal{F})$. The difference between the two is that patterns terms make use of the special *Sup* function symbol.

Intuitively, as can be seen from the following definition, $Sup\,(t)$ represents all terms containing the term t as a sub-term. For instance, the terms A, $\mathbf{pair}(x, A)$, $\mathbf{encr}(A, K)$, $\cdots$ all belong to $[\![Sup\,(A)]\!]$.

Definition 3. *Given a pattern term* pt*, let* $[\![\mathrm{pt}]\!]$ *be defined as follows:*

$$\begin{array}{rcl}[\![\mathrm{pt}]\!] & = & \{\mathrm{pt}\} \quad \textit{if } \mathrm{pt} \textit{ is a constant or a variable} \\ [\![\mathbf{pair}(\mathrm{pt}_1, \mathrm{pt}_2)]\!] & = & \{\mathbf{pair}(t_1, t_2) \mid t_1 \in [\![\mathrm{pt}_1]\!], t_2 \in [\![\mathrm{pt}_2]\!]\} \\ [\![\mathbf{encr}(\mathrm{pt}_1, k)]\!] & = & \{\mathbf{encr}(t_1, k) \mid t_1 \in [\![\mathrm{pt}_1]\!]\} \\ [\![\mathit{Sup}(\mathrm{pt})]\!] & = & \{t \mid \textit{ there is a position } p \textit{ in } t \textit{ s.t. } t_{|p} \in [\![\mathrm{pt}]\!]\}\end{array}$$

We then represent a pair $(\mathcal{S}, \mathcal{H})$ by triple $\langle \mathcal{S}, \mathcal{G}, \mathcal{B} \rangle$ of finite sets of secrets, good and bad patterns. Good patterns correspond to "maybe safe" hat terms and bad patterns define "really unsafe" hat terms. The formal definition of the concretization of $\langle \mathcal{S}, \mathcal{G}, \mathcal{B} \rangle$ is given in [5].

5 A Symbolic Verification Algorithm

The symbolic algorithm is obtained from the algorithm of Figure 1 by replacing each operation by a corresponding symbolic one that operates on $\langle \mathcal{S}, \mathcal{G}, \mathcal{B} \rangle$. For the sake of presentation, we explain the symbolic algorithm in the particular case where $\mathcal{B}$ consists of terms rather than pattern terms, i.e., *Sup* does not occur in any term in $\mathcal{B}$. The extension of the algorithm to pattern terms is explained in a technical report [5].

Before presenting the algorithm we need to introduce the following definitions. Let t and t' be terms and let q be a position in t. A substitution $\sigma : \mathcal{X} \cup \mathcal{X}_s \to \mathcal{T}(\mathcal{X} \cup \mathcal{X}_s, \mathcal{F})$ is called *a q-matcher* of t' on t, if $\sigma(t') = t$ and there is a position $q' \preceq q$ such that $t'_{|q'} \in \mathcal{X}_s$. Given a set $\mathcal{G}$ of terms, we say that t q-matches in $\mathcal{G}$, if there is a term $t' \in \mathcal{G}$ and a q-matcher of t' on t. Moreover, a substitution $\sigma : \mathcal{X} \cup \mathcal{X}_s \to \mathcal{T}(\mathcal{X} \cup \mathcal{X}_s, \mathcal{F})$ is called *a q-unifier* of t' with t, if $\sigma(t') = \sigma(t)$ and one of the following conditions is satisfied: 1.) $t_{|q} \in \mathcal{X}_s$ and there is a variable $y_s \in \mathcal{X}_S$ such that $y_s \in_c t'_{|q}$ and in case y_s is in the domain of σ, a secret or a secret variable appears at a critical position in $\sigma(y_s)$ or 2.) there exists a position $q' \preceq q$ such that $t'_{|q'} \in \mathcal{X}_s$. Also, q-unification can be extended to sets of terms.

The symbolic algorithm takes as input a set of rules $\mathcal{R}$, a set of secrets $\mathcal{S}$, a set of good terms $\mathcal{G}$ and an empty set of bad terms $\mathcal{B} = \emptyset$. It computes new bad terms and secrets until the concretization of $\langle \mathcal{S}, \mathcal{G}, \mathcal{B} \rangle$ becomes stable w.r.t. all rules in $\mathcal{R}$. Let us now sketch its main steps:

1. The set $\mathcal{S}$ of secrets becomes $\mathcal{S} \cup \mathit{Keys}(\mathcal{G})^{-1}$, where $\mathit{Keys}(\mathcal{G})^{-1}$ is the set of keys of the form k^{-1} such that $\mathbf{encr}(t, k)$ is in $\mathcal{G}$.
2. Given a rule $t_p \to t_c$ in $\mathcal{R}$, we have to consider all possible critical occurrences of a secret in the conclusion t_c. This is done by replacing variables in t_c by secret variables. In other words, we consider all rules in $\mathcal{R}' = \{\sigma(t_p) \to \sigma(t_c) \mid t_p \to t_c \in \mathcal{R},\ \sigma : \mathit{var}(t_c) \to \mathcal{X}_s\}$.
3. Given a rule $t_p \to t_c$ in $\mathcal{R}'$, the algorithm computes the set of dangerous substitution DS as follows. A substitution $\sigma : \mathit{var}(t_c) \to \mathcal{T}(\mathcal{X} \cup \mathcal{X}_s, \mathcal{F})$ is *dangerous* if there is a critical position p in t_c such that $t_{c|p} \in \mathcal{X}_s \cup \mathcal{S}$ and the following condition holds: for every positions p_i, q_i such that $q_i.p_i = p$, if

$t_{c|q_i}$ p_i-matches in $\mathcal{G}$, then σ can be completed into a p_i-unifier of $t_{c|q_i}$ in $\mathcal{B}$, meaning there are $b \in \mathcal{B}$ and σ' with $dom(\sigma') \subseteq var(b)$ such that $\sigma \cup \sigma'$ is a p_i-unifier of $t_{c|q_i}$ with b. Then,

$$DS := \{\sigma : var(t_c) \rightarrow \mathcal{T}(\mathcal{X} \cup \mathcal{X}_s, \mathcal{F}) \mid \sigma \text{ is dangerous}\}$$

Example 3 below illustrates the computation of dangerous substitutions.

4. The set of *dangerous premises* is given by $DP = \{\sigma(t_p) \mid \sigma \in DS\}$. The new bad terms are the subterms of a dangerous premise $t \in DP$ that were supposed to protect a secret in t. So, bad terms are particular instances of terms in $\mathcal{G}$ that must be removed. Formally,

$$BHat = \{t_{|q} \,|\, t \in DP,\ \exists p\ q.p \text{ critical in } t, t_{|q.p} \in \mathcal{X}_s \cup \mathcal{S},\ t_{|q}\ p\text{-matches with } \mathcal{G}\}$$

Each occurrence of $\mathcal{H}'$ in the semantic algorithm is now replaced by the set of bad terms $\mathcal{B}'$ and computing the restriction of $\mathcal{H}'$ now corresponds to the instruction $\mathcal{B}' := \mathcal{B}' \cup BHat$.

5. Finally, *newS* is replaced by

$$newS := \{t \in DP \mid var(t) \cap \mathcal{X}_s = \emptyset,\ \forall s \in \mathcal{S}'.\ s \notin_c t\}$$

Example 3. We illustrates the computation of dangerous substitutions on the rule $t_p \rightarrow t_c$ given in Figure 2 and the sets of terms:

$$\begin{aligned}\mathcal{G} &= \{\ \mathbf{encr}(x_s, K_B),\ \mathbf{encr}(x_s, K_A)\ \}\\ \mathcal{B} &= \{\ \mathbf{encr}(\mathbf{pair}(I, x'_s), K_B),\ \mathbf{encr}(\mathbf{pair}(\mathbf{pair}(A, x''), x''_s), K_B)\ \}\end{aligned}$$

We consider the conclusion of the rule. The first step consists in looking for all the critical positions in the conclusion where a secret or a secret variable appears. We find one variable $y_s \in \mathcal{X}_s$ at position 01101 *in the term t_c. For y_s, there are two protecting positions ϵ and* 011*: the term* $\mathbf{encr}(x_s, K_B)$ 01101-*matches with $t_{c|\epsilon}$ and it also* 01-*matches with $t_{c|011}$. Then, we search for all substitutions that* 01101-*unify $t_{c|\epsilon}$ and* 01-*unify $t_{c|011}$ with the bad terms. Starting with position $p_1 = \epsilon$, $t_{c|\epsilon}$* 01101-*unifies with the bad term* $\mathbf{encr}(\mathbf{pair}(I, x'_s), K_B)$ *for the unifier $\sigma' = [z = I, x'_s = t_{c|01}]$. This cancels the top most protection. Then, we attempt to complete the substitution σ' so that it also cancels the protection at position $p_2 =$ 011. To do so, we try to* 01-*unify the term $\sigma'(t_c)_{|p_2} =$* $\mathbf{encr}(\mathbf{pair}(\mathbf{pair}(y, I), y_s), K_B)$ *with a bad term and succeed with the bad term* $\mathbf{encr}(\mathbf{pair}(\mathbf{pair}(A, x''), x''_s), K_B)$ *and the unifier $\sigma'' = [y = A, x'' = I, y_s = x''_s]$. The two unifiers are then composed and restricted to the domain $var(t_c)$ resulting in the substitution $\sigma = (\sigma' \cup \sigma'')_{/var(t_c)} = [y = A, z = I]$. Pursuing this process does not provide other substitutions and finally σ appears to be the only dangerous substitutions. We now look at the premise of the rule to compute the new bad terms induced by σ. The position of the secret variable y_s in t_p is protected by the good term* $\mathbf{encr}(x_s, K_A)$ *that* 01-*matches on $t_{p|0}$. However, the dangerous substitution σ tells that this protection will not work in case where y is A and z is I. Consequently, we refine the set of hat messages by removing this particular case. In our symbolic representation, this comes out to add* $\sigma(t_{p|0}) = \mathbf{encr}(\ \mathbf{pair}(\mathbf{pair}(A, I), y_s),\ K_A)$ *to the set of bad terms.* □

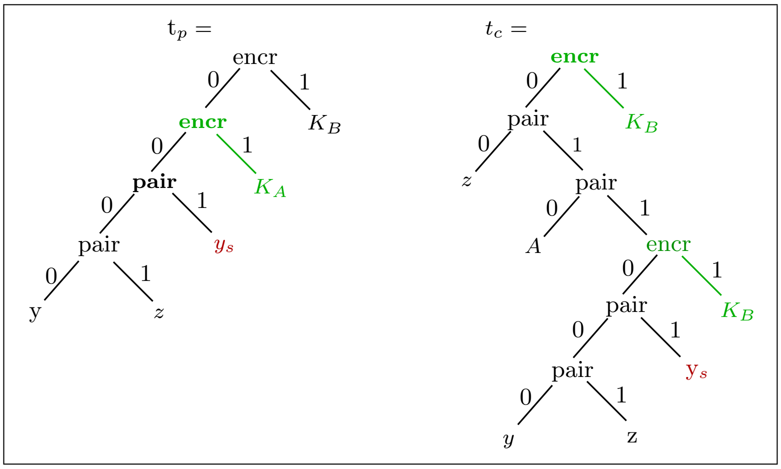

Fig. 2. Illustration of computing dangerous substitutions.

6 An Example of Verification and Experimentations

We illustrate our approach on the verification of the corrected version of the Needham-Schroeder protocol, also called Needham-Schroeder-Lowe. The correction bears on the second transition of principal B in Example 1, which becomes:

$$\begin{array}{l} A \rightarrow B : \{A, N_1\}_{K_B} \\ B \rightarrow A : \{B, N_1, N_2\}_{K_A} \\ A \rightarrow B : \{N_2\}_{K_B} \end{array}$$

We run our prototype implementation, named HERMES, with the secrets $\{N_2, K_A^{-1}, K_B^{-1}\}$, the empty set of bad patterns and the set $\mathcal{G} = \{\{x_s\}_{K_A}, \{x_s\}_{K_B}\}$ of good patterns. It terminates with the set of secrets unchanged and the set $\mathcal{B}$ of bad patterns : $\{I, x_s\}_{K_A}$, $\{A, (N_1^{AA}, Sup\,(I, x_s))\}_{K_A}$, $\{A, (N_1, Sup\,(I, x_s))\}_{K_A}$.

From this, we can conclude that the Needham-Schroeder-Lowe protocol preserves the secret N_2. Concerning, the uncorrected version of Example 1, during computation of new secrets and bad patterns, we arrive at a situation where we have to add $\{A, N_1^{AI}\}_{K_I}$ as a secret. As this message contains neither a fresh nonce nor a secret, it can not be a secret. We stop the computation and follow it back to reconstruct the attack known as “man in the middle”.

6.1 On the Termination of the Symbolic Algorithm

In this section, we present a technique that makes a depth-first implementation of the symbolic verification algorithm always terminate, at a price of safe approximation of the results. In fact, our prototype implementation of our verification

Table 1. This figure has been obtainded by running HERMES on a Pentium III 600Mhz PC under Linux 2.2.19.

Protocol Name	Result	Time (sec)
Yahalom	OK	12.67
Needham-Schroeder Public Key	Attack	0.01
Needham-Schroeder Public Key (with a key server)	Attack	0.90
Needham-Schroeder-Lowe	OK	0.02
Otway-Rees	OK*	0.02
Denny Sacco Key Distribution with Public Key	Attack	0.02
Wide Mouthed Frog (modified)	OK	0.01
Kao-Chow	OK	0.07
Neumann-Stubblebine	OK*	0.04
Needham-Schroeder Symmetric Key	Attack	0.04
ISO Symmetric Key One-Pass Unilateral Authentication	Attack	0.01
ISO Symmetric Key Two-Pass Unilateral Authentication	OK	0.01
Andrew Secure RPC	Attack	0.04
Woo and Lam	OK	0.06
Skeme	OK	0.06

* There is a known attack of the untyped version of the protocol. Discovering this type attack automatically requires to deal with non-atomic keys. This is not yet implemented in HERMES.

algorithm, named HERMES, terminates with precise results on all pratical examples of protocols we tried. That is, the results did not show any false attack (see Table 1).

A sequence $(t_i)_{i\geq 0}$ of pattern terms is called *increasing at a sequence* $(p_i)_{i\geq 0}$ of positions, if the following conditions are satisfied for every $i \geq 0$:

1. $p_i \in dom(t_i)$ and $p_i \preceq p_{i+1}$,
2. $t_0[z/p_0] = t_i[z/p_0]$, where z is fresh variable.
3. $t_{i|p_i} = t_{0|p_0}$.

Let us consider an example to clarify these definitions.

Example 4. Consider the following rule from the session (A, A) of Needham-Schroeder-Lowe protocol presented in Section 6:

$$r = \frac{\{(A, (N_1^{AA}, y))\}_{K_A}}{\{y\}_{K_A}}.$$

Consider the sequence $(\{\theta^i(I, x)\}_{K_A})_{i\geq 0}$, where $\theta(z) = (A, (N_1^{AA}, z))$. The first three terms of the sequence are $\{\theta^0(I, x)\}_{K_A} = \{(I, x)\}_{K_A}$, $\{\theta^1(I, x)\}_{K_A} = \{(A, (N_1^{AA}, (I, x)))\}_{K_A}$ and $\{\theta^2(I, x)\}_{K_A} = \{(A, (N_1^{AA}, (A, (N_1^{AA}, (I, x)))))\}_{K_A}$. The whole sequence can be obtained by iteratively computing the bad patterns induced by the rule r starting from the bad term $\{(I, x)\}_{K_A}$. Thus, a naive application of our symbolic algorithm will not terminate. On the other hand, this sequence is increasing at $(p_i = 0(11)^i)_{i\geq 0}$. Indeed, $\{\theta^i(I, x)\}_{K_A}[z/p_0] = \{z\}_{K_A}$

and $\left(\{\theta^i(I,x)\}_{K_A}\right)_{|p_i} = (I,x)$, for every $i \geq 0$. We will see now how this fact can be exploited to make the algorithm to converge. □

The clue of our technique for enforcing termination of the symbolic algorithm is expressed by the following proposition:

Proposition 3. *Let $(t_i)_{i\geq 0}$ be increasing at $(p_i)_{i\geq 0}$. Then,*

$$\bigcup_{i\geq 0} [\![t_i]\!] \subseteq \bigcup_{i<j} [\![t_i]\!] \cup [\![t_j[Sup\,(t_{j_{|p_j}})/p_j]]\!], \text{ for every } j \geq 0.$$

Example 5. Consider again our Example 4. Then, if we choose $j=1$, we obtain a set consisting of the two pattern terms $\{(I,x)\}_{K_A}$ and $\{(A,(N_1^{AA}, Sup\,(I,x)\,))\}_{K_A}$ which approximates the whole sequence $\left(\{\theta^i(I,x)\}_{K_A}\right)_{i\geq 0}$.

7 Conclusion

In this paper, we presented a method based on abstract interpretation for verifying secrecy properties of cryptographic protocols in a general model. Our method deals with unbounded number of sessions, unbounded number of principals, unbounded message depth and unbounded creation of fresh nonces. The main contribution of this paper is a verification algorithm that consists of computing an inductive invariant using patterns as symbolic representation. In case the given protocol is correct, our method provides a proof tree that can be exploited for certification. More precisely, from the obtained proof tree we can automatically deduce a proof, for instance in Coq or PVS, that can serve for certification. We are actually working on implementing this idea.

Our method can already deal with models in which we distinguish between long term and short term keys and which contain variables ranging over keys. The idea here is that short term keys can be revealed to the intruder when a session has terminated. This is not the case for long term keys. This allows a more faithful modeling of some protocols.

Our tool together with the examples of Table 1 can be experimented at the url: `http://www-verimag.imag.fr/~Liana.Bozga/eva/hermes.php`.

References

1. Martín Abadi. Secrecy by typing in security protocols. In *Theoretical Aspects of Computer Software*, volume 1281 of *LNCS*, p. 611–638, 1997.
2. Martín Abadi and Bruno Blanchet. Secrecy Types for Asymmetric Communication. In *Foundations of Software Science and Computation Structures*, volume 2030 of *LNCS*, p. 25–41, 2001.
3. Roberto M. Amadio and Denis Lugiez. On the reachability problem in cryptographic protocols. In *International Conference on Concurrency Theory*, volume 1877 of *LNCS*, p. 380–394, 2000.

4. D. Bolignano. An approach to the formal verification of cryptographic protocols. In *ACM Conference on Computer and Communications Security*, p. 106–118, 1996.
5. L. Bozga, Y. Lakhnech, and M. Périn. Abstract interpretation for secrecy using patterns. Technical report, Verimag, 2002.
6. J. Clark and J. Joacob. A survey on authentification protocol. Available at the url http://www.cs.york.ac.uk/∼jac/papers/drareviewps.ps, 1997.
7. H. Comon, V. Cortier, and J. Mitchell. Tree automata with one memory, set constraints, and ping-pong protocols. In *International Colloquium on Automata, Languages and Programming*, volume 2076 of *LNCS*, 2001.
8. H. Comon, V. Shmatikov. Is it possible to decide whether a cryptographic protocol is secure or not? *Journal of Telecommunications and Information Technology*, 2002.
9. H. Comon-Lundh and V. Cortier. Security properties: Two agents are sufficient. Technical report, LSV, 2002.
10. D. Dolev, S. Even, and R. M. Karp. On the security of ping-pong protocols. In *Advances in Cryptology*, p. 177–186, 1982.
11. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
12. N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Workshop on Formal Methods and Security Protocols*, 1999.
13. S. Even and O. Goldreich. On the security of multi-party ping pong protocols. Technical report, Israel Institute of Technology, 1983.
14. F.J.T. Fábrega, J.C. Herzog, and J.D. Guttman. Strand Spaces: Why is a Security Protocol Correct? In *IEEE Conference on Security and Privacy*, p. 160–171, 1998.
15. T. Genet and F. Klay. Rewriting for cryptographic protocol verification. In *International Conference on Automated Deduction*, volume 1831 of *LNCS*, 2000.
16. A. Gordon and A. Jeffrey. Authenticity by typing for security protocols. In *IEEE Computer Security Foundations Workshop*, p. 145–159, 2001.
17. Jean Goubault-Larrecq. A method for automatic cryptographic protocol verification. In *International Workshop on Formal Methods for Parallel Programming: Theory and Applications*, volume 1800 of *LNCS*, 2000.
18. G. Lowe. An attack on the Needham-Schroeder public-key authentification protocol. *Information Processing Letters*, 56(3):131–133, 1995.
19. G. Lowe. Breaking and fixing the Needham-Schroeder Public-Key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *LNCS*, p. 147–166, 1996.
20. C. Meadows. Invariant generation techniques in cryptographic protocol analysis. In *Computer Security Foundations Workshop*, 2000.
21. J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *ACM Conference on Computer and Communications Security*, p. 166–175, 2001.
22. David Monniaux. Abstracting Cryptographic Protocols with Tree Automata. In *Static Analysis Symposium*, volume 1694 of *LNCS*, p. 149–163, 1999.
23. R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.
24. M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *IEEE Computer Security Foundations Workshop*, 2001.
25. J. Thayer, J. Herzog, and J. Guttman. Honest Ideals on Strand Spaces. In *IEEE Computer Security Foundations Workshop*, p. 66–78, 1998.