"Alexandru Ioan Cuza" University of Iaşi
Faculty of Computer Science

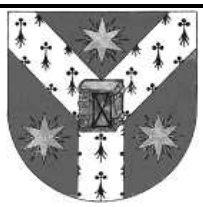
TECHNICAL REPORT

Decidability and Complexity Results for Security Protocols

Ferucio L. Ţiplea
Constantin Enea
Cătălin V. Bîrjoveanu

TR 05-02, June 2005

ISSN 1224-9327



Universitatea "Alexandru Ioan Cuza" Iaşi
Facultatea de Informatică
Str. Berthelot 16, 6600-Iaşi, Romania
Tel. +40-32-201090, email: bibl@infoiasi.ro

Decidability and Complexity Results for Security Protocols [*] [†]

Ferucio L. Ţiplea, Constantin Enea, and Cătălin V. Bîrjoveanu

Department of Computer Science
"Al.I.Cuza" University of Iaşi
Iaşi, Romania
E-mail: {fltiplea,cenea,cbirjoveanu}@infoiasi.ro

Abstract

Security protocols are prescribed sequences of interactions between entities designed to provide various security services across distributed systems. Security protocols are often wrong due to the extremely subtle properties they are supposed to ensure. Deciding whether or not a security protocol assures secrecy is one of the main challenge in this area.

In this paper we survey the most important decidability and complexity results regarding the secrecy problem for various classes of security protocols, such as bounded protocols, finite-sessions protocols, normal protocols, and tagged protocols. All the results are developed under the same formalism. Several flawed statements claimed in the literature are corrected. Simplified proofs and reductions, as well as slight extensions of some known results, are also provided.

1 Introduction

A security protocol specifies a set of rules under which a sequence of messages is exchanged between two or more parties in order to achieve security goals such as authentication or establishing new shared secrets. Examples of security protocols include the Kerberos authentication scheme used to manage encrypted passwords on clusters of interconnected computers, Secure Sockets Layer used by internet browsers and servers to carry out secure internet transactions etc.

As Needham and Schroeder pointed out in [17], "security protocols are prone to extremely subtle errors that are unlikely to be detected in normal operation" (see also

[*] The research reported in this paper was partially supported by ECO-NET 08112WJ/2004-2005 and by National University Research Council of Romania, grants CNCSIS 632/28/2004 and CNCSIS 632/50/2005.

[†] A version of this paper was presented in the 1st NATO Advanced Research Workshop "Verification of Infinite State Systems with Applications to Security" VISSAS 2005, Timişoara (Romania), March 17-22, 2005.

[13]). Much work has been done to develop methods to ensure their correctness, including [6, 3, 15, 18, 24, 27]. Although there are many formalisms for modeling security protocols, most of them are based on the same model of adversary capabilities, suggested by Dolev and Yao [6]. The seminal work of Burrows, Abadi, and Needham who developed the BAN authentication logic [3] was one of the first attempts to make reasoning about security protocols more systematic. Meadows' NRL Analyzer [15], which is an equational rewriting tool, works very nicely for secrecy properties where the problem can be couched in terms of whether or not the intruder can deduce certain sensitive information. In 1996, Paulson introduced the inductive approach with automated support provided by his Isabelle proof assistant [18]. With this formalism, security properties are stated as predicates over traces, and they are proved inductively. The CSP approach can also be used to construct models of security protocols, which consists of a number of communicating components and are thus well suited to analysis [24]. The strand spaces approach [27] associates strands to model the behavior of the participants.

One of the main question one may ask when dealing with security protocols is the following one: How difficult is to prove that a security protocol assures secrecy? This is a central problem for a couple of years and, in the meantime, several results have been developed. Even and Goldreich [10] showed that secrecy is NP-hard for ping-pong protocols and, relaxing the definition so that the operators can operate on half words causes the secrecy to become undecidable.

Secrecy for security protocols is undecidable. Many proofs have been proposed to this, based on the halting problem for Turing/counter machines, Post's correspondence problem, reset Petri nets etc. If security protocols are restricted by bounding the number of nonces, the message lengths, the number of sessions etc., decidability of secrecy can be gained. In almost all these cases, the complexity of deciding secrecy is quite high. For example, the initial secrecy problem, which is a particular case of the secrecy problem, is DEXPTIME-complete for bounded protocols without freshness check [9], while the secrecy problem is NP-complete for finite-session protocols [23]. There are other types of restrictions as well, which lead to decidability of secrecy [14, 22, 20, 25].

The main problem one encounters when a complete and clear image has to be drawn about all these results is with the more or less significant differences between the formalisms used to model security protocols. From this point of view, expressing all these results under the same formalism would be preferable. This was in fact the starting point of this paper. We have adopted the formalism in [19] with slight modifications in order to tackle the secrecy problem for security protocols (Section 2). Secrecy is shown undecidable (Section 3) by exhibiting a reduction from the Post correspondence problem (our reduction corrects the one in [9] which is flawed). Two cases are taken into consideration: infinitely many nonces but bounded-length messages, and finitely many nonces but arbitrary-length messages, and in both cases secrecy is shown undecidable. Section 4 deals with restrictions aimed to produce security protocols with a decidable secrecy problem. The first subclass of such protocols is that of bounded protocols. When freshness check is not required, the initial secrecy problem for bounded protocols is DEXPTIME-complete [9]. The main differences between the proof in [9] and the one in this paper are:

- precise bounds proving the membership to DEXPTIME are provided;
- DEXPTIME-hardness is showed by exhibiting a simpler reduction than the one in [9].

We also point out the main difficulty which appears when one wants to extend this result to the secrecy problem for bounded protocols without freshness check.

The finite-session restriction gives rise to a subclass of security protocols whose secrecy problem is NP-complete [23]. The approach in this paper differs from the one in [23] by:

- the secrecy problem in this paper is more general;
- generation of fresh nonces and short-term keys is explicitly allowed in this paper;
- the NP-hardness is showed by a different reduction based on 3-SAT, which allows to handle the secrecy problem in its generality (the approach in [23] uses "temporary secrets" which are later revealed in order the intruder be able to get the "big secret").

The class of normal protocols in [20] is based on a very interesting equivalence relation on terms but, unfortunately, the results are seriously flawed. We provide details regarding this matter and we discuss two different approaches under which the secrecy problem for normal protocols with finitely many nonces becomes decidable. The class of tagged protocols [22] is the last one considered in this paper, for which decidability of secrecy under infinitely many nonces and arbitrary-length messages is obtained.

2 Modeling Security Protocols

Many models for security protocols have been proposed in the literature, starting with the Dolev-Yao model [6]. Thus, [7] introduces a formalism based on multiset rewriting, [27] proposes strand spaces, Paulson uses the inductive method to prove properties of security protocols [18] etc. Under these formalisms, several decidability and complexity results regarding secrecy have been obtained. However, the main problem one encounters when a complete and clear image has to be drawn about all these results is with the more or less significant differences between these formalisms. From this point of view, expressing all these results under the same formalism would be preferable.

In what follows we will adopt the model proposed in [19] with slight modifications, and we will use it in order to discuss the decidability and complexity of the secrecy problem for various classes of security protocols.

Protocol signatures and terms A *protocol signature* consists of three sets $\mathcal{S} = (\mathcal{A}, \mathcal{K}, \mathcal{N})$, where $\mathcal{A}$ is a finite set of *agent names* (or shortly, *agents*), $\mathcal{K}$ is an at most countable set of *keys*, and $\mathcal{N}$ is an at most countable set of *nonces* [1]. It is assumed that:

[1] Abbreviation for "number once used".

- $\mathcal{A}$ contains a special element denoted by I and called the *intruder* [2] [3]. All the other elements are called *honest agents* and we denote by Ho their set;
- $\mathcal{K} = \mathcal{K}_0 \cup \mathcal{K}_1$, where $\mathcal{K}_0$ is a set of *short-term keys* and $\mathcal{K}_1$ is a finite set of *long-term keys* shared by agents. K_{AB} will denote the long-term key shared by two distinct agents A and B, and $\mathcal{K}_A$ stands for the set of long-term keys known by A (i.e., keys shared by A with other participants, *public keys* [4], and A's *private key*);
- some honest agents are provided from the beginning with some *secret information*, not known to the intruder, and given as nonces or short-term keys. Therefore, we denote by $Secret_A$ the set of secret information the agent A is provided from the beginning. It may be the case that distinct honest agents share the same piece of secret information. That is, $Secret_A$ and $Secret_B$ might not be disjoint for some honest agents A and B;
- the intruder is provided from the beginning with a nonce n_I and a key $K_I \in \mathcal{K}_0$. They play the role of a generic set of new data the intruder may generate in the course of a run, and it is assumed that they cannot be generated by any honest agent [5].

The set of *basic terms* is $\mathcal{T}_0 = \mathcal{A} \cup \mathcal{K} \cup \mathcal{N}$. The set $\mathcal{T}$ of *terms* is defined inductively by:

- every basic term is a term;
- if t_1 and t_2 are terms, then (t_1, t_2) is a term;
- if t is a term and K is a key, then $\{t\}_K$ is a term [6] [7].

We extend the construct (t_1, t_2) to $(t_1, \ldots, t_n)$ as usual by letting

$$(t_1, \ldots, t_n) = ((t_1, \ldots, t_{n-1}), t_n),$$

for all $n \geq 3$. Sometimes, parenthesis will be omitted.

[2] An all-powerful intruder is assumed, who can read any message and block further transmission, can impersonate any honest agent, can decompose messages into parts and remember them (including decrypting any message for which the intruder has obtained the key), can generate fresh data as needed, and can compose new messages from known data and send.

However, the intruder cannot generate honest agents' secrets, nor it can break encryption.

[3] As it has been pointed out in [26], any group of Dolev-Yao intruders colluding with one another cannot cause more attacks than a single intruder acting alone.

[4] In public key cryptography, any key is a pair of two keys, $(K, \overline{K})$, where K is public and $\overline{K}$ is private.

[5] It should be noticed that the intruder does not get more power if he is allowed to generate infinitely many nonces. This is because the nonce n_I can be used to generate $\{n_I\}_{K_I}, \{\{n_I\}_{K_I}\}_{K_I}, \ldots$.

[6] Sometimes, encryption by *composed keys* (non-atomic keys) is necessary. For instance, in many cases, symmetric keys are built up from shared secrets and other exchanged data in the protocol. In order to be able to model such cases, the construct $\{t\}_K$ should be extended by allowing encryption by arbitrary terms, $\{t\}_{t'}$.

[7] The length of a term is defined as usual, by taking into consideration that pairing and encryption are operations. Thus, $|t| = 1$ for any $t \in \mathcal{T}_0$, $|(t_1, t_2)| = |t_1| + |t_2| + 1$, for any terms t_1 and t_2, and $|\{t\}_K| = |t| + 2$, for any term t and key K.

Given a term t, $Sub(t)$ is the set of all *subterms* of t (defined as usual), and $ESub(t)$ is the subset of all *encrypted subterms* of t. That is,

$$ESub(t) = \{t' \in Sub(t) | (\exists t'' \in \mathcal{T})(\exists K \in \mathcal{K})(t' = \{t''\}_K)\}.$$

These two notations are extended to sets of terms by union.

The *perfect encryption assumption* we adopt [6] states that a message encrypted with a key K can be decrypted only by an agent who knows the corresponding inverse of K [8], and the only way to compute $\{t\}_K$ is by encrypting t with K.

Actions Security protocols are usually specified as sequences of communications of the form $A \rightarrow B : t$. Such communications say that A sends t to B, and B receives t from A.

For analysis of security protocols, a third party should be taken into consideration, namely the intruder. In such a case, a communication $A \rightarrow B : t$ says that A sends t to B, but it is not guaranteed that B will receive it (B may receive a different message composed by the intruder). From this point of view, it seems that a decomposition of the communication $A \rightarrow B : t$ into two actions, has to be taken into consideration:

- a *send action*, performed by A and denoted $A!B : t$, and
- a *receive action*, performed by B and denoted $B?A : t$.

From a technical point of view, it is useful to specify, along with any send action, the set of nonces and short-term keys freshly generated by A to compose t. Therefore, a *send action* will be of the form $A!B : (M)t$, while a *receive action* will be of the form $A?B : t$. In both cases,

- A is assumed an honest agent [9],
- $A \neq B$,
- $t \in \mathcal{T}$ is the *term of the action*, and
- $M \subseteq Sub(t) \cap (\mathcal{N} \cup \mathcal{K}_0)$ is the *set of new terms of the action* [10].

Given an action a, we shall denote by $M(a)$ the set

$$M(a) = \begin{cases} M, & \text{if } a = A!B : (M)t \\ \emptyset, & \text{if } a = A?B : t. \end{cases}$$

Moreover, $t(a)$ stands for the term of a. We will simply write $A!B : t$ whenever $M = \emptyset$.

Denote by Act the set of all actions, and by $Act(A)$ the set of all actions performed by A, that is, $Act(A) = \{A!B : (M)t | B \in \mathcal{A}\} \cup \{A?B : t | B \in \mathcal{A}\}$. For a sequence

[8] The inverse of a symmetric key K is K itself, and the inverse of a public key K is $\overline{K}$.

[9] With this formalism, the intruder cannot perform explicitly any send or receive action. This is not a limitation of the intruder's capabilities because any message sent (received) by an honest agent is intercepted by (comes from) the intruder (see also the computation rule).

[10] The terminology will be clear when the computation rule is discussed.

of actions $w = a_1 \cdots a_l$ and an agent A, we define the *restriction of* w *to* A as being the sequence obtained from w by removing all actions not in $Act(A)$. Denote this sequence by $w|_A$. The notations $M(a)$ and $t(a)$ are extended to sequences of actions by union.

Protocols A *protocol* is a triple $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w)$, where

- $\mathcal{S}$ is a protocol signature;
- $\mathcal{C}$ is a subset of $\mathcal{T}_0$, called the set of *constants* of $\mathcal{P}$;
- w is a non-empty sequence of actions, called the *body* of the protocol, such that no action in w contains the intruder.

Constants are publicly known elements in the protocol that cannot be re-instantiated (as it will be explained a little bit later). As usual, $\mathcal{C}$ does not include private keys, elements in $Secret_A$ for any honest agent A, n_I, K_I, or elements in $M(w)$.

Any non-empty sequence $w|_A$, where A is an agent, is called a *role* of the protocol. A role specifies the actions a participant should perform in a protocol, and the order of these actions.

Substitutions and events A security protocol specifies the legal actions to be performed between generic participants in the protocol. The application of the protocol in various real cases is done by instantiating the protocol. Moreover, in a real environment, multiple instantiations of the same protocol may run.

Instantiations of a protocol are given by *substitutions*, which are functions which map agents to agents, short-term keys to short-term keys, and nonces to arbitrary terms. A substitution which maps nonces to nonces is called a *well-typed substitution*.

Substitutions are homomorphically extended to terms, actions, and sequences of actions. For instance, if σ is a substitution and $t = \{t'\}_{K_{AB}}$ is a term, then $\sigma(t) = \{\sigma(t')\}_{K_{\sigma(A)\sigma(B)}}$.

A substitution σ is called:

- *suitable for an action* $a = AxB : y$ if
  - $\sigma(A)$ is an honest agent [11];
  - $\sigma(A) \neq \sigma(B)$;
  - σ maps distinct nonces from $M(a)$ into distinct nonces, distinct keys into distinct keys, and it has disjoint ranges for $M(a)$ and $Sub(t(a)) - M(a)$;
- *suitable for a sequence of actions* if it is suitable for each action in the sequence;
- *suitable for a subset* $\mathcal{C} \subseteq \mathcal{T}_0$ if it is the identity on $\mathcal{C}$.

[11] Each action is instantiated into a legitimate action.

An *event* of a protocol $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w)$ is any triple $e_i = (u, \sigma, i)$, where $u = a_1 \cdots a_l$ is a role of $\mathcal{P}$, σ is a substitution suitable for u and $\mathcal{C}$, and $1 \leq i \leq l$. $\sigma(a_i)$ is the *action of the event* e_i. As usual, $act(e_i)$ ($t(e_i)$, $M(e_i)$) stands for the the action of e_i (term of e_i, set of new terms of e_i). The event e_i is *well-typed* if the substitution σ is well-typed.

The *local precedence relation* on events is defined by

$$(u, \sigma, i) \rightarrow (u', \sigma', i')$$

if and only if $u' = u$, $\sigma' = \sigma$, and $i' = i + 1$, provided that $i < |u|$. $\stackrel{+}{\rightarrow}$ is the transitive closure of $\rightarrow$. Given an event e, ${}^{\bullet}e$ stands for the *set of all local predecessors of* e, i.e.,

$${}^{\bullet}e = \{e' | e' \stackrel{+}{\rightarrow} e\}.$$

Message generation rules Following [18], for a given a set X of terms denote by $analz(X)$ the least set of terms that satisfies the following properties:

1. $X \subseteq analz(X)$;
2. if $(t_1, t_2) \in analz(X)$, then $t_1, t_2 \in analz(X)$; (decomposition)
3. if $\{t\}_K, K^{-1} \in analz(X)$, then $t \in analz(X)$; (decryption)
4. if $\{\{t\}_K\}_{K^{-1}} \in analz(X)$, then $t \in analz(X)$, (simplification)

and by $synth(X)$ the least set of terms that satisfies the following properties:

1. $X \subseteq synth(X)$;
2. if $t_1, t_2 \in synth(X)$, then $(t_1, t_2) \in synth(X)$;
3. if $t, K \in synth(X)$, then $\{t\}_K \in synth(X)$.

Moreover, $\overline{X}$ stands for $synth(analz(X))$. The following proposition states the basic properties of these operators.

Proposition 2.1 Let X be a set of terms and σ be a substitution. Then, the following properties hold true:

(1) $X \subseteq analz(X) \cap synth(X)$;

(2) If $X \subseteq X'$ then $analz(X) \subseteq analz(X')$ and $synth(X) \subseteq synth(X')$;

(3) $analz(analz(X)) = analz(X)$;

(4) $\overline{\overline{X}} = analz(\overline{X}) = \overline{X}$;

(5) $\sigma(analz(X)) \subseteq analz(\sigma(X))$;

(6) $\sigma(synth(X)) \subseteq synth(\sigma(X))$;

(7) $\sigma(\overline{X}) \subseteq \overline{\sigma(X)}$.

States and runs A *state* of a protocol gives information about the knowledge of all participants at a given moment in the evolution of the protocol. Therefore, it is defined as an indexed set $s = (s_A | A \in \mathcal{A})$, where $s_A \subseteq \mathcal{T}$, for any agent A. By $Sub(s)$ we will denote the set $Sub(s) = Sub(\bigcup_{A \in \mathcal{A}} s_A)$.

The *initial state* of a protocol $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w)$ is $s_0 = (s_{0A} | A \in \mathcal{A})$, where:

- $s_{0A} = \mathcal{A} \cup \mathcal{C} \cup \mathcal{K}_A \cup Secret_A$, for any honest agent A;
- $s_{0I} = \mathcal{A} \cup \mathcal{C} \cup \mathcal{K}_I \cup \{n_I, K_I\}$.

Actions in a protocol define a transition relation between states of the protocol as follows. Given two states s and s' and an action a, we write $s[a\rangle s'$ if and only if:

1. if a is of the form $A!B : (M)t$, then:
   (a) (*enabling condition*) $t \in \overline{s_A \cup M}$ and $M \cap Sub(s) = \emptyset$;
   (b) $s'_A = s_A \cup M \cup \{t\}$, $s'_I = s_I \cup \{t\}$, and $s'_C = s_C$ for any $C \in \mathcal{A} - \{A, I\}$;
2. if a is of the form $A?B : t$, then:
   (a) (*enabling condition*) $t \in \overline{s_I}$;
   (b) $s'_A = s_A \cup \{t\}$ and $s'_C = s_C$, for all $C \in \mathcal{A} - \{A\}$.

We extend the notation "$[\cdot\rangle$" to events by letting $s[e\rangle s'$ whenever $s[act(e)\rangle s'$, and we call $s[e\rangle s'$ a *computation step*.

The requirement "$M \cap Sub(s) = \emptyset$" in the enabling condition is usually called the *freshness check* requirement. When for a protocol $\mathcal{P}$ this requirement is dropped, we will say that $\mathcal{P}$ is *without freshness check* (we will come back to this in the section dedicated to bounded protocols).

A *computation* or *run* of the protocol is a sequence of computation steps,

$$s_0[e_1\rangle s_1[\cdots[e_k\rangle s_k,$$

also written as $s_0[e_1 \cdots e_k\rangle s$ or even $e_1 \cdots e_k$, such that the following properties hold true:

1. $s_{i-1}[e_i\rangle s_i$, for any $1 \leq i \leq k$;
2. ${}^\bullet e_i \subseteq \{e_1, \ldots, e_{i-1}\}$, for any $1 \leq i \leq k$ (for $i = 1$, ${}^\bullet e_i$ should be empty).

It follows from (2) above that a necessary condition for an event e to be enabled at a sequence of events is that all its local predecessors be already in the sequence.

If ξ is a run, $s_0[\xi\rangle s$, and e is an event that is enabled at s and ξe is still a run, then we will say that e is *enabled at* ξ. We will also use the notation $(s, \xi)[e\rangle(s', \xi e)$, where $s[e\rangle s'$.

A run whose events are all well-typed is called a *well-typed run*. It is clear that all messages communicated in well-typed runs are bounded-length.

The secrecy problem We say that a term $t \in \mathcal{T}_0$ is

- *secret at a state* s if $t \in analz(s_A) - analz(s_I)$, for some honest agent A;
- *secret along a run* ξ if it is secret at s, where $s_0[\xi\rangle s$. We remark that a term t is secret at any previous state once it is secret at s, but no agent or public key is secret along any run.

A run ξ is called *leaky* if there exists $t \in \mathcal{T}_0$ and a proper prefix ξ' of ξ such that t is secret along ξ' but not along ξ.

The *secrecy problem* for security protocols is the problem to decide whether a security protocol has leaky runs.

If we replace the set $\mathcal{T}_0$ by $\bigcup_{A \in Ho} Secret_A$ in all the definitions above, we obtain a particular case of the secrecy problem, called the *initial secrecy problem*. That is, the initial secrecy problem is the problem to decide whether a given security protocol has runs ξ such that $t \in analz(s_I)$ for some initial secret t, where $s_0[\xi\rangle s$.

We illustrate the above definitions by the Needham-Schroeder public key protocol [17]. As a general remark, all protocol specifications will implicitly omit I and the elements associated to it, and $Secret_A$ will be the empty set whenever it is not explicitly specified.

The Needham-Schroeder protocol is

$$\begin{array}{lcl} A \to B & : & \{x, A\}_{K_B} \\ B \to A & : & \{x, y\}_{K_A} \\ A \to B & : & \{y\}_{K_B}, \end{array}$$

where A and B are agents, K_A and K_B are the public keys of A and B, respectively, and x, y are nonces.

Using the formalism adopted in this paper this protocol is given by $\mathcal{P}_{NS} = (\mathcal{S}, \mathcal{C}, w)$, where:

- $\mathcal{S}$ consists of two honest agents A and B, their public keys $(K_A, \overline{K}_A)$ and $(K_B, \overline{K}_B)$, and two nonces x and y;
- $\mathcal{C} = \emptyset$;
- w is the following sequence:

$$\begin{array}{lcl} A!B & : & (\{x\})\{x, A\}_{K_B} \\ B?A & : & \{x, A\}_{K_B} \\ B!A & : & (\{y\})\{x, y\}_{K_A} \\ A?B & : & \{x, y\}_{K_A} \\ A!B & : & \{y\}_{K_B} \\ B?A & : & \{y\}_{K_B} \end{array}$$

The protocol $\mathcal{P}_{NS}$ has two roles, $u_1 = w|_A$ and $u_2 = w|_B$:

- u_1 is given by:

$$\begin{array}{lcl} A!B & : & (\{x\})\{x, A\}_{K_B} \\ A?B & : & \{x, y\}_{K_A} \\ A!B & : & \{y\}_{K_B} \end{array}$$

- u_2 is given by:

$$\begin{array}{lcl} B?A & : & \{x, A\}_{K_B} \\ B!A & : & (\{y\})\{x, y\}_{K_A} \\ B?A & : & \{y\}_{K_B} \end{array}$$

Two suitable substitutions σ_1 and σ_2 for $\mathcal{P}_{NS}$ are illustrated:

- $\sigma_1(A) = A$, $\sigma_1(B) = I$, $\sigma_1(x) = m$, $\sigma_1(y) = n$, and
- $\sigma_2(A) = A$, $\sigma_2(B) = B$, $\sigma_2(x) = m$, $\sigma_2(y) = n$.

An example of a run showing an attack on $\mathcal{P}_{NS}$ [13] is given by the following sequence ξ:

$$\begin{array}{llcl} (u_1, \sigma_1, 1) & A!I & : & (\{m\})\{m, A\}_{K_I} \\ (u_2, \sigma_2, 1) & B?A & : & \{m, A\}_{K_B} \\ (u_2, \sigma_2, 2) & B!A & : & (\{n\})\{m, n\}_{K_A} \\ (u_1, \sigma_1, 2) & A?I & : & \{m, n\}_{K_A} \\ (u_1, \sigma_1, 3) & A!I & : & \{n\}_{K_I} \\ (u_2, \sigma_2, 3) & B?A & : & \{n\}_{K_B} \end{array}$$

We remark that the run ξ is leaky because n is secret at the prefix

$$\xi' = (u_1, \sigma_1, 1)(u_2, \sigma_2, 1)(u_2, \sigma_2, 2)(u_1, \sigma_1, 2)$$

of ξ, but n is not secret at ξ.

3 Undecidability Results

Many authors have proved that the secrecy problem is undecidable. The techniques used vary from model to model, and they are based on the halting problem for Turing/counter machines [21], Post's correspondence problem (PCP) [9], the implication problem for existential Horn clauses [8], the reachability problem for reset Petri nets [5] etc.

In what follows we will discuss the approach based on *Post's Correspondence Problem* (PCP). This approach was first used in [10] for half-word ping-pong protocols. Later, the same approach was used in [9] accompanied by the multiset rewriting formalism for security protocols. However, the proof in [9] does not work. We will first provide a correct proof by exhibiting a reduction from PCP and then explain why the proof in [9] does not work.

A PCP instance is a set P of pairs of words over a given alphabet:

$$P = \{(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)\}.$$

A solution for P is any sequence $i_1, \ldots, i_l$ such that

$$x_{i_1} x_{i_2} \cdots x_{i_l} = y_{i_1} y_{i_2} \cdots y_{i_l}.$$

The problem consists in deciding whether or not P has solutions. It is well-known that this problem is undecidable [12].

In reducing PCP to secrecy, the main problem we encounter is with encoding words. Given a word $x = a_1 \cdots a_n$, there are mainly two ways of encoding it:

- separate letters by nonces and encrypt each triple `(nonce,letter,nonce)`. Thus, x can be encoded by the sequence

  $$\{u_0, a_1, u_1\}_K, \ldots, \{u_{n-1}, a_n, u_n\}_K,$$

  where u_i are nonces and K is a key;
- as a tuple, $x = (a_1, \ldots, (a_{n-1}, a_n) \cdots)$.

Infinitely Many Nonces and Bounded-length Messages. Given a PCP instance P over a finite alphabet Σ, define the following security protocol:

- assume that the elements of Σ are nonces. We also consider three fixed nonces e_1, e_2, and \$;
- the set of constants is $\Sigma \cup \{e_1, e_2, \$\}$;
- five distinct nonces u, u', v, v', and x, and $|x| + |y|$ distinct nonces for each pair $(x, y) \in P$, are also considered;
- the agents are A, B, C, D, C_i, D_i, E_a, F_a, G_a, and H_a, for all $1 \leq i \leq k$ and $a \in \Sigma$;
- three pairwise distinct long-term keys are used, K, L, and R. They are shared by all the honest agents;
- the protocol actions are as follows:
  - A initiates the protocol by sending the pair of nonces $\{e_1, e_2\}$:

    $$A!B \quad : \quad \{e_1, e_2\}_K$$

    The idea is that this pair of nonces will mark the left end of any sequence of words over P. For instance, if $x_1 = ab$, $y_1 = bac$, $x_2 = bba$, and $y_2 = aa$, then the following two sequences can be generated:

    $$\{e_1, b, u_1\}_L, \{u_1, b, u_2\}_L, \{u_2, a, u_3\}_L, \{u_3, a, u_4\}_L, \{u_4, b, u_5\}_L$$

    and

    $$\{e_2, a, v_1\}_R, \{v_1, a, v_2\}_R, \{v_2, b, v_3\}_R, \{v_3, a, v_4\}_R, \{v_4, c, v_5\}_R$$

    These two sequences simulate $x_2 x_1$ and $y_2 y_1$;

– The addition of a new pair (x_i, y_i) is simulated by actions performed by two agents C_i and D_i. C_i receives "the last pair of nonces", appends x_i and y_i, correspondingly, and sends the "last pair of nonces":

$$\begin{array}{lcl} C_i?D_i & : & \{u, v\}_K \\ C_i!D_i & : & (\{u_1, \ldots, u_{s_i}, v_1, \ldots, v_{p_i}\}) \\ & & \{u, x_i^1, u_1\}_L, \{u_1, x_i^2, u_2\}_L, \ldots, \{u_{s_i-1}, x_i^{s_i}, u_{s_i}\}_L, \\ & & \{v, y_i^1, v_1\}_R, \{v_1, y_i^2, v_2\}_R, \ldots, \{v_{p_i-1}, y_i^{p_i}, v_{p_i}\}_R, \\ & & \{u_{s_i}, v_{p_i}\}_K \end{array}$$

where $x_i = x_i^1 \cdots x_i^{s_i}$ and $y_i = y_i^1 \cdots y_i^{p_i}$;

– two agents C and D start the verification process by changing any pair (u, v) of nonces they receive into a pair $((\$, u), (\$, v))$:

$$\begin{array}{lcl} C?D & : & \{u, v\}_K \\ C!D & : & \{(\$, u), (\$, v)\}_K \end{array}$$

– two agents E_a and F_a check whether or not any two letters on corresponding positions are the same:

$$\begin{array}{lcl} E_a?F_a & : & \{(\$, u), (\$, v)\}_K, \{u', a, u\}_L, \{v', a, v\}_R \\ E_a!F_a & : & \{(\$, u'), (\$, v')\}_K \end{array}$$

where $a \in \Sigma$;

– two agents G_a and H_a reveal a secret when a solution to P is found:

$$\begin{array}{lcl} G_a?H_a & : & \{(\$, u), (\$, v)\}_K, \{e_1, a, u\}_L, \{e_2, a, v\}_R \\ G_a!H_a & : & (\{x\})\{x\}_K \\ G_a!H_a & : & x \end{array}$$

where $a \in \Sigma$.

It is not difficult to see that P has solutions iff the protocol associated to P does not assure secrecy under well-typed runs.

The solution in [9] is wrong because it uses the same pair $\{u, v\}_K$ of nonces for the verification process. In this way, new pairs (x_i, y_i) can be appended while the verification process is not done. For example, if we consider the PCP instance $P = \{(ca, a), (b, cb)\}$, then the computation below shows that the protocol associated to P reveals the secret (the first row gives information about the step performed, the second row shows the last pair of nonces an agent receives and the last pair of nonces the agent sends, and the third/fourth row shows concatenation of the first/second coordinates of pairs in P):

append	verification	append
$\{e_1, e_2\}_K \rightarrow \{u_2, v_1\}_K$	$\{u_2, v_1\}_K \rightarrow \{u_1, e_2\}_K$	$\{u_1, e_2\}_K \rightarrow \{u_3, v_3\}_K$
$\{e_1, c, u_1\}_L, \{u_1, a, u_2\}_L$	$\{e_1, c, u_1\}_L$	$\{e_1, c, u_1\}_L, \{u_1, b, u_3\}_L$
$\{e_2, a, v_1\}_R$		$\{e_2, c, v_2\}_R, \{v_2, b, v_3\}_R$

verification	verification and secret reveal
$\{u_3, v_3\}_K \rightarrow \{u_1, v_2\}_K$	$\{u_1, v_2\}_K$
$\{e_1, c, u_1\}_L$	$(\{x\})\{x\}_K, x$
$\{e_2, c, v_2\}_R$	

However, P has no solution. This problem occurs because pairs of nonces are used both for the concatenation process and for the verification one. Our solution makes distinction between these two processes. We use pairs $\{u, v\}_K$ of nonces for concatenation and pairs $\{(\$, u), (\$, v)\}_K$ for verification.

One more remark is in order. One may say that the verification process performed by G_a and H_a can be simplified to

$$\begin{array}{lcl} G_a?H_a & : & \{(\$, e_1), (\$, e_2)\}_K \\ G_a!H_a & : & (\{x\})\{x\}_K \\ G_a!H_a & : & x \end{array}$$

However, this is wrong because C may receive $\{e_1, e_2\}_K$. In such a case, C returns $\{(\$, e_1), (\$, e_2)\}_K$ which can close the verification process although no solution exists.

The main characteristics of this simulation are:

- infinite number of nonces;
- bounded-length messages.

Therefore, we obtain the following result.

Theorem 3.1 The secrecy problem for security protocols with infinitely many nonces and bounded-length messages, is undecidable.

Finitely Many Nonces and Arbitrary-length Messages. If we model words as tuples of letters, then the protocol is pretty much the same as the one presented above. In this case, only one long-term key K is necessary, shared by all honest agents. The actions are:

- $A!B \quad : \quad \{e_1, e_2\}_K$
- $\begin{array}{lcl} C_i?D_i & : & \{u, v\}_K \\ C_i!D_i & : & \{(\ldots(u, x_i^1), \ldots, x_i^{s_i}), (\ldots(v, y_i^1), \ldots, y_i^{p_i})\}_K \end{array}$
  where $x_i = x_i^1 \cdots x_i^{s_i}$ and $y_i = y_i^1 \cdots y_i^{p_i}$;
- $\begin{array}{lcl} C?D & : & \{u, v\}_K \\ C!D & : & \{(\$, u), (\$, v)\}_K \end{array}$
- $\begin{array}{lcl} E_a?F_a & : & \{(\$, (u, a)), (\$, (v, a))\}_K \\ E_a!F_a & : & \{(\$, u), (\$, v)\}_K \end{array}$
- $\begin{array}{lcl} G_a?H_a & : & \{(\$, (e_1, a)), (\$, (e_2, a))\}_K \\ G_a!H_a & : & (\{x\})\{x\}_K \\ G_a!H_a & : & x \end{array}$

We can easily prove that P has solutions iff the protocol does not assure secrecy.

The main characteristics of this simulation are:

- finite number of nonces;
- no short term key;
- bounded-depth encryptions;
- unbounded-length messages.

Therefore, we obtain the following result.

Theorem 3.2 The secrecy problem for security protocols with finitely many nonces and arbitrary-length messages, is undecidable.

4 Decidability and Complexity Results

As we have seen in the previous section, the prominent sources of undecidability are *unbounded-length messages* and/or *unbounded number of nonces*. By imposing various (reasonable) restrictions on security protocols, decidability of secrecy can be gained. For instance, when both message length and the number of nonces is bounded, the secrecy is decidable [8]. But this is not the only case. In [16, 23], bounds on the number of sessions that can occur in any run of the protocol, lead again to decidability of secrecy. Imposing syntactic restrictions to security protocols, decidability results can be also obtained, as the one in [14, 22]. Semantic restrictions can also lead to decidability, as it is shown in [20, 25] where semantic criteria which give decidability in the presence of terms of arbitrary length but with finitely many nonces, are obtained.

In this section we will survey these results, correct some of the flawed statements with respect to them, and in some cases provide simpler proofs and reductions than the ones known from the literature.

4.1 Bounded Protocols

Let $\mathcal{P}$ be a protocol, $T \subseteq \mathcal{T}_0$ be a finite set, and $k \geq 1$. A (T, k)-*run* of $\mathcal{P}$ is any run of $\mathcal{P}$ that satisfies:

- all terms in the run are built up upon T (therefore, the number of nonces and keys is bounded);
- all messages communicated in the course of the run have length at most k.

When for a protocol $\mathcal{P}$ only (T, k)-runs are considered we will say that it is a *protocol under* (T, k)*-runs* or a (T, k)*-bounded protocol*, and denote this by $(\mathcal{P}, T, k)$. The secrecy problem for such protocols is formulated with respect to (T, k)-runs only, by taking into consideration the set T instead of $\mathcal{T}_0$.

Let $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w)$ be a (T, k)-bounded protocol. Then:

1. the number of messages communicated in the course of any (T, k)-run is bounded by
$$k^3|T|^{\frac{k+1}{2}} = 2^{3\log k + \frac{k+1}{2}\log|T|};$$

2. the number of instantiations (substitutions) of a given role u of $\mathcal{P}$ with messages of length at most k over T is bounded by
$$(2^{3\log k + \frac{k+1}{2}\log|T|})^{|u|(\frac{k+1}{2}+2)}$$
(u has exactly $|u|$ actions, and each action has at most $\frac{k+1}{2}+2$ elements that can be substituted);

3. the number of (T, k)-*events* (i.e., events that can occur in all (T, k)-runs) is bounded by
$$\begin{array}{rcl} \text{number of } (T,k)\text{-events} & \leq & \sum_{u\in role(\mathcal{P})} |u| \cdot 2^{(3\log k + \frac{k+1}{2}\log|T|)|u|(\frac{k+1}{2}+2)} \\ & \leq & \sum_{u\in role(\mathcal{P})} |u| \cdot 2^{(3\log k + \frac{k+1}{2}\log|T|)|w|(\frac{k+1}{2}+2)} \\ & = & |w| \cdot 2^{(3\log k + \frac{k+1}{2}\log|T|)|w|(\frac{k+1}{2}+2)} \\ & = & 2^{\log|w| + (3\log k + \frac{k+1}{2}\log|T|)|w|(\frac{k+1}{2}+2)} \end{array}$$
where $role(\mathcal{P})$ is the set of all roles of $\mathcal{P}$.

Define the *size* of a (T, k)-bounded protocol $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w)$ as being
$$size(\mathcal{P}) = |w| + \frac{k+1}{2}\log|T|.$$

This is a realistic measure. It takes into consideration the number of actions and the maximum number of bits necessary to represent messages of length at most k. As we can see, the number of events that can occur in all (T, k)-runs of a (T, k)-bounded protocol is exponential in $poly(size(\mathcal{P}))$, for some polynomial $poly$.

In a (T, k)-bounded protocol the set of messages the intruder can get is always bounded and, as a result, the secrecy problem is decidable.

In [8, 9] it has been shown that the initial secrecy problem for bounded protocols with no *disequality tests* is DEXPTIME-complete (the absence of disequality tests in the formalism used in [8, 9] corresponds, in the formalism we have adopted, to the absence of the freshness check). We recall below the proof of the membership of this problem to DEXPTIME [8, 9].

Theorem 4.1 The initial secrecy problem for bounded protocols without freshness check is in DEXPTIME.

Proof The following algorithm decides whether or not a bounded protocol without freshness check has leaky runs with respect to initial secrets:

`input:` bounded protocol $(\mathcal{P}, T, k)$ without freshness check;
`output:` "leaky protocol" if $\mathcal{P}$ has some leaky (T, k)-run w.r.t. initial secrets, and "non-leaky protocol", otherwise;

```
begin
  let E' be the set of all (T, k)-events;
  ξ := λ; s := s_0;
  repeat
    E := E';
    E' := ∅;
    bool := 0;
    while E ≠ ∅ do
      begin
        choose e ∈ E;
        E := E − {e};
        if (s, ξ)[e⟩(s', ξe) then
            begin
              s := s'; ξ := ξe; bool := 1;
            end
          else E' := E' ∪ {e};
      end
  until bool = 0;
  if (⋃_{A∈Ho} Secret_A) ∩ analz(s_I) ≠ ∅
      then "leaky protocol" else "non-leaky protocol"
end.
```

In the algorithm above, E' is the set of all events that could not be applied in the previous cycle. Initially, E' is the set of all events of the protocol. The boolean variable $bool$ takes the value 0 when no event in E can be applied.

The algorithm generates a "maximal run" ξ with respect to the intruder's state (that is, the state s has the property that $analz(s_I)$ is the maximum set of knowledge the intruder can get, where $s_0[\xi\rangle s$). Its correctness follows from the *persistence property* of the computation relation which, in this case, does not require freshness check (if an action is enabled at a state s, then it will be enabled at any state reachable from s; therefore, actions are persistent once they become enabled). Now, it is easy to see that the protocol is leaky with respect to initial secrets if and only if the run ξ generated by this algorithm is leaky with respect to initial secrets (i.e., $(\bigcup_{A\in Ho} Secret_A) \cap analz(s_I) \neq \emptyset$).

The algorithm terminates in exponential time with respect to the size of the protocol. Indeed, if no event in E can extend to the right the current run ξ, then the algorithm terminates. Otherwise, all events in E that can extend to the right the current run, taken in an arbitrary but fixed order, are applied in one cycle of `repeat`. The next cycle will process, in the same way, the elements of E that could not be applied at the previous cycle. Therefore, the number of cycles of `repeat` is bounded by $|E|$ (each cycle applies at least one event, except for the last one). At the first cycle of `repeat` the number of cycles in `while` is $|E|$, at the second cycle of `repeat` the number of cycles in `while` is at most $|E| - 1$, and so on. Therefore, the number of cycles in `while`, in all `repeat` cycles, is bounded by $|E| + (|E| - 1) + \cdots + 1 = \mathcal{O}(|E|^2)$. Therefore, the complexity of the algorithm is exponential in $poly(size(\mathcal{P}))$, for some polynomial $poly$, showing that the initial secrecy problem for bounded protocols without freshness

check is in DEXPTIME. □

One may ask whether the algorithm in the proof of Theorem 4.1 can be used to decide the secrecy problem for bounded protocols without freshness check. The new algorithm would generate a "maximal run" and would output "leaky protocol" iff this run is leaky. Unfortunately, such a conclusion would be wrong. In order to show that we consider the following protocol under $(T, 3)$-runs, where T is an arbitrary subset of $\mathcal{T}_0$ which includes the agents A, B, C, and D, the nonces x and y, and the long-term key K:

$$\begin{array}{lcl} A!B & : & (\{x\})\{x\}_K \\ C!D & : & (\{y\})y \end{array}$$

If the algorithm applies all the events based on the second action and then applies all the events based on the first action, a non-leaky maximal run w.r.t. the intruder's knowledge is generated; therefore, the algorithm outputs "non-leaky protocol". However, the protocol is leaky.

As a conclusion, the order in which the events are applied when freshness check is not required but "new secrets" are generated in the course of runs, is crucial.

Now, let us prove that the initial secrecy problem for bounded protocols without freshness check is DEXPTIME-hard. To this we will exhibit a simpler reduction than the one in [8, 9] based on the membership problem for unary logic programs [4]. Recall first the concept of a unary logic program. Let Σ be a set consisting of one constant symbol $\perp$ and finitely many unary function symbols, let $Pred$ be a finite set of unary predicate symbols, and x be a variable. A *unary logic program* over Σ, $Pred$, and x is a finite set of clauses of the form

$$p_0(t_0) \leftarrow p_1(t_1), \ldots, p_n(t_n)$$

or

$$p_0(t_0) \leftarrow true,$$

where $p_0, \ldots, p_n \in Pred$, and $t_0, \ldots, t_n$ are terms over $\Sigma \cup \{x\}$ with t_0 being flat, that is, $t_0 \in \{\perp, x, f(x) | f \in \Sigma - \{\perp\}\}$. Moreover, all clauses with $p_0(\perp)$ in the head have only $true$ in the body.

An *atom* is a construct of the form $p(t)$, where $p \in Pred$ and t is a term. If t is a *ground term*, that is, it does not contain x, then $p(t)$ is called a *ground atom*. A *proof tree* for a ground atom $p(t)$ under a unary logic program LP is any tree that satisfies:

- its nodes are labeled by ground atoms;
- the root is labeled by $p(t)$;
- each intermediate node which is labeled by some B has children labeled by $B_1, \ldots, B_n$, where $B \leftarrow B_1, \ldots, B_n$ is a ground instance of a clause in LP (i.e., the variable x is substituted by ground terms over Σ);
- all the leaves are labeled by $true$.

The *membership problem for unary logic programs* is the problem to decide, given a logic program LP and a ground atom $p(t)$, whether there exists a proof tree for $p(t)$ under LP. In [4] it has been proved that this problem is DEXPTIME-complete (being equivalent to the type-checking problem for path-based approximation for unary logic programs).

Theorem 4.2 The initial secrecy problem for bounded protocols is DEXPTIME-hard.

Proof Let LP be a unary logic program over some Σ, $Pred$, and x, and let $p(t)$ be a ground atom over Σ and $Pred$. Define a security protocol $\mathcal{P}$ as follows:

- to each element $e \in \Sigma \cup Pred \cup \{x\}$ associate a nonce u_e. Except for u_x, all these nonces are constants of the protocol;
- encode terms and atoms as follows:
  - $\langle e \rangle = u_e$, for all $e \in \{\bot, x\}$;
  - $\langle f(t) \rangle = (u_f, \langle t \rangle)$, for any unary function symbol f and term t;
  - $\langle p(t) \rangle = (u_p, \langle t \rangle)$, for any predicate symbol p and term t.
- consider the agents A_C, B_C, E and F, for any clause C. It is assumed that they are pairwise distinct;
- consider a key K known only by the honest agents;
- $Secret_F = \{y\}$, where y is a distinct nonce, and $Secret_X = \emptyset$, for all $X \neq F$;
- to each clause $C:\ p(t_0) \leftarrow p_1(t_1), \ldots, p_n(t_n)$ we associate the role

$$\begin{array}{lcl} A_C?B_C & : & \{\langle p_1(t_1)\rangle\}_K, \ldots, \{\langle p_n(t_n)\rangle\}_K \\ A_C!B_C & : & \{\langle p_0(t_0)\rangle\}_K \end{array}$$

- to each clause $C:\ p(t_0) \leftarrow true$ we associate the role

$$A_C!B_C \quad : \quad \{\langle p_0(t_0)\rangle\}_K$$

- the following role reveals a secret if $p(t)$ has a tree proof under LP:

$$\begin{array}{lcl} F?E & : & \{\langle p(t)\rangle\}_K \\ F!E & : & y \end{array}$$

We consider the protocol $\mathcal{P}$ under (T, k)-runs, where T is the set of all elements mentioned above (nonces, agents, the key K, and the nonce y) and k is a suitable chosen constant (the maximum length of some clause under some instantiation used to decide the membership of $p(t)$). Then, it is easily seen that $p(t)$ has a tree proof in LP if and only if the protocol $\mathcal{P}$ under (T, k)-runs reveals the secret. □

Corollary 4.1 The initial secrecy problem for bounded protocols without freshness check is DEXPTIME-complete.

DEXPTIME-hardness of the initial secrecy problem for bounded protocols has been proved in [8, 9] by exhibiting a reduction from the implication problem for Horn clauses without existential quantifiers and function symbols. Our reduction is simpler.

Summing up, bounded protocols are characterized by:

- bounded number of nonces and short-term keys;
- bounded-length messages (but the protocol may be non-well-typed);
- unbounded number of role instantiations.

4.2 Finite-session Protocols

Another way to restrict protocols in order to get decidability of secrecy is to limit the number of sessions. The first significant result in this direction was announced in [8] but no proof was provided until [9] was published. In the meantime, stronger results have been obtained [23, 1].

Protocols under 1-session Runs Let $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w)$ be a protocol. A *1-session run* of $\mathcal{P}$ is any run obtained by applying each role exactly once, under the same substitution (i.e., all events in the run are defined by using the same substitution). Therefore, any 1-session run has length $|w|$.

If a protocol is considered under 1-session runs, then only a finite number of basic terms are used. Therefore, we may consider that the set $\mathcal{T}_0$ is finite, whenever we deal with such protocols. However, no restriction on the message lengths is imposed.

Secrecy for protocols under 1-session runs is NP-complete [23]. A key ingredient in proving this is the *directed acyclic graphs representation* (abbreviated, *dag-representation*) of messages. For a given set of terms S, the dag-representation of S is the graph $\mathcal{G}_S = (\mathcal{V}_S, \mathcal{E}_S)$, where

- $\mathcal{V}_S = Sub(S)$;
- $\mathcal{E}_S = \{v_1 \stackrel{left}{\longrightarrow} v_2 | (\exists t)(v_1 = \{v_2\}_t \ \vee \ v_1 = (v_2, t))\} \cup$
  $\{v_1 \stackrel{right}{\longrightarrow} v_2 | (\exists t)(v_1 = \{t\}_{v_2} \ \vee \ v_1 = (t, v_2))\}$.

The dag representation of a set of terms is unique and can be computed in polynomial time. Therefore, we can define the *dag-size* of S, denoted by $|S|_{dag}$, as the number of distinct vertices in $\mathcal{G}_S$, that is the number of elements in $Sub(S)$. For a term t, $|t|_{dag}$ stands for $|\{t\}|_{dag}$. The dag-size of a substitution σ is the maximum of the dag-sizes of $\sigma(x)$, for any $x \in \mathcal{T}_0$.

We will recall now the proof in [23] which shows that secrecy for protocols under 1-session runs is NP-complete. There are slight differences between the proof in [23] and the one below:

- with the approach in [23] compound keys are allowed, but not with the one in this paper (however, one can extend this formalism as pointed out in Section 2 in order to allow composed keys so that the proof below works in this case too);

- with the approach in [23] agents are not allowed to generate new nonces or keys, but with the one in this paper they are allowed to;

- the secrecy problem in [23] is formulated with respect to some initial secret (given from the beginning), while in this paper the secrecy problem is the one in Section 2 which is more general.

Theorem 4.3 The secrecy problem for security protocols under 1-session runs is in NP.

Proof Let $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w)$ be a security protocol under 1-session runs. As we have already mentioned, we may assume that $\mathcal{T}_0$ is finite. Let n be the *dag-size* of $\mathcal{P}$, i.e., the dag-size of the set S of all terms in the protocol together with the terms in s_0 and $\mathcal{T}_0$.

Given a substitution σ, we say that a sequence of events ξ is a *1-session sequence under* σ if $|\xi| = |w|$, each event in ξ has the form (u, σ, i) for some role u and $1 \leq i \leq |u|$, and each event in ξ appears exactly once and in the order it is in the corresponding role (i.e., if $\xi = \xi' e \xi''$ and $e = (u, \sigma, i)$, then (u, σ, j) appears in ξ', for any $j < i$).

Consider now the following non-deterministic algorithm.

`input:` protocol $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w)$ under 1-session runs;
`output:` "leaky protocol", if $\mathcal{P}$ has some leaky run;
`begin`
  `guess` a substitution σ of dag-size at most n;
  `guess` a 1-session sequence of events under σ, $e_1 \cdots e_k$, where $k = |w|$;
  `if` $e_1 \cdots e_k$ is a 1-session run `then`
    `begin`
      `let` $s_0[e_1\rangle s_1[e_2 \cdots e_k\rangle s_k$;
      `guess` $t \in \mathcal{T}_0$, $A \in Ho$, and $1 \leq i < k$;
      `if` $t \in analz(s_{iA}) - analz(s_{iI})$ and $t \in analz(s_{kI})$
        `then` "leaky protocol";
    `end`
`end.`

In [23] it has been shown that if $\mathcal{P}$ is leaky then is has a leaky 1-session run under some substitution σ of dag-size at most n [12].

The complexity of the algorithm can be determined as follows:

- a substitution σ of dag-size at most n can be chosen in time $\mathcal{O}(n^2)$ because at most n x's should be substituted and, for each x, the dag-size of $\sigma(x)$ should not exceed n;

- a 1-session sequence of events $e_1 \cdots e_k$ under σ can be chosen in time $\mathcal{O}(n)$;

[12] More precisely, if $\mathcal{P}$ is leaky then it has "minimal" leaky runs (called *normal* runs in [23]) with respect to some partial order relation. Then, [23] proves that each minimal run under some substitution σ has the property that σ is of dag-size at most n.

- given a state s and an event e such that s_A has dag-size at most n, for all agents A (including the intruder), one can check whether e is enabled at s by using at most n analysis or synthesis rules [23]. Such a rule can be chosen in time $\mathcal{O}(n^2)$. Therefore, one can check in polynomial time in n whether e is enabled at s and, as a conclusion, one can decide in polynomial time whether $e_1 \cdots e_k$ is a 1-session run. Moreover, if $s_0[e_1\rangle s_1[e_2 \cdots e_k\rangle s_k$ then s_{iA} is of dag-size at most n, for any i and agent A;

- a term $t \in \mathcal{T}_0$, an agent $A \in Ho$, and an index $1 \leq i \leq k$ can be chosen in linear time in n. Moreover, by a similar discussion as the one above, one can check in polynomial time in n whether $t \in analz(s_{iA}) - analz(s_{iI})$ and $t \in analz(s_{kI})$.

Summing up, the algorithm above performs in non-deterministic polynomial time in n. Therefore, the secrecy problem for protocols under 1-session runs is in NP. □

To prove that the problem is NP-hard, a reduction from 3-SAT is exhibited. An instance of 3-SAT consists of a set of boolean variables $\{x_1, \ldots, x_n\}$ and a boolean expression

$$D = D_1 \wedge \cdots \wedge D_k,$$

where each D_j, called a *clause* or *conjunct*, is of the form

$$D_j = \alpha_{j,1} \vee \alpha_{j,2} \vee \alpha_{j,3}$$

and $\alpha_{j,l}$ is a *literal*, i.e., either x or $\neg x$, for some variable x.

D is *satisfiable* if it can be evaluated to the truth value true under some assignment.

The theorem below follows the main idea from [23] but it is adapted to the secrecy problem as defined in Section 2.

Theorem 4.4 The secrecy problem for security protocols under 1-session runs is NP-hard.

Proof Let D be an instance of 3-SAT, as the one above. A protocol $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w)$ will be defined such that D is satisfiable if and only if the protocol $\mathcal{P}$ under 1-session runs reveals the secret. The protocol is as follows:

- A, B, C, D, A_i, B_i, $A_{j,l}$, and $B_{j,l}$ are pairwise distinct agents, for any $1 \leq j \leq k$, $1 \leq i \leq n$, and $l = 1, 2, 3$;

- K is a long-term key shared by all honest agents;

- V_i is a long-term key shared by A_i, B_i, $A_{j,l}$, and $B_{j,l}$ such that x_i defines the literal $\alpha_{j,l}$ (that is, $\alpha_{j,l}$ is either x_i or $\neg x_i$), for any i, j, and l;

- $K_{j,C}$ is a long-term key shared by $A_{j,l}$ and C, for any j and l;

- S, F, $\top$, and $\bot$ are four long-term keys. $\{S\}_\bot$ and $\{S\}_\top$ represent the truth values *false* and, respectively, *true*, while $\{F\}_\bot$ and $\{F\}_\top$ are "fake" values for the same truth values. They will be used to enforce each role be applied completely;

- the protocol actions are:
  - (initialization)

$$A!B \quad : \quad \{\{S\}_\perp\}_K, \{\{S\}_\top\}_K, \{\{F\}_\perp\}_{V_1}, \{\{F\}_\top\}_{V_1}, \cdots, \{\{F\}_\perp\}_{V_n}, \{\{F\}_\top\}_{V_n}$$

    A sends to B the real and the fake truth values, all of them encrypted. They will be intercepted by the intruder and used to generate an assignment;
  - (variable assignment)

$$\begin{array}{lcl} A_i?B_i & : & \{v\}_K \\ A_i!B_i & : & \{v\}_{V_i} \end{array}$$

    When A_i receives (from the intruder) a truth value, he assigns it to x_i by sending $\{v\}_{V_i}$. As there exists exactly one value for each i, any 1-session run will assure that all variables are instantiated in a non-redundant way;
  - (checking the truth value of clause D_j)

$$\begin{array}{lcl} A_{j,l}?B_{j,l} & : & \{\{v\}_\top\}_{V_i} \\ A_{j,l}!B_{j,l} & : & \{\{\top\}_v\}_{K_{j,C}} \end{array}$$

    if $\alpha_{j,l} = x_i$, and

$$\begin{array}{lcl} A_{j,l}?B_{j,l} & : & \{\{v\}_\perp\}_{V_i} \\ A_{j,l}!B_{j,l} & : & \{\{\top\}_v\}_{K_{j,C}} \end{array}$$

    if $\alpha_{j,l} = \neg x_i$, for each j and l.

    A clause D_j is satisfiable if at least one of its literals is evaluated to the truth value $true$. For example, if an agent $A_{j,l}$ receives an assignment $\{\{S\}_\top\}_{V_i}$ for a variable x_i and $\alpha_{j,l} = x_i$, then it returns $\{\{\top\}_S\}_{K_{j,C}}$ because, from his point of view, D_j is evaluated to the truth value *true*. However, if $A_{j,l}$ receives $\{\{F\}_\top\}_{V_i}$ then $\{\{\top\}_F\}_{K_{j,C}}$ is returned which does not count in establishing the truth value of D_j;
  - (reveal the secret)

$$\begin{array}{lcl} C?D & : & \{\{\top\}_S\}_{K_{1,C}}, \ldots, \{\{\top\}_S\}_{K_{k,C}} \\ C!D & : & (\{x\})\{x\}_K \\ C!D & : & x \end{array}$$

    When C receives confirmation that all clauses were evaluated to the truth value true, then C generates a nonce and reveals it.

It is easy to see that if D is satisfiable then the intruder can learn a secret in exactly 1-session run, and vice versa. In other words, the protocol under 1-session runs is leaky if and only if D is satisfiable. □

Corollary 4.2 The secrecy problem for security protocols under 1-session runs is NP-complete.

Protocols under m-session Runs An *m-session run* of a protocol $\mathcal{P}$, where $m \geq 1$, is any run obtained by interleaving m 1-session runs.

The analysis of a protocol $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w)$ under m-session runs comes down to the analysis of a protocol $\mathcal{P}' = (\mathcal{S}', \mathcal{C}', w')$ under 1-session runs, where:

- w' is obtained by concatenating m copies of w, each of which obtained from w by renaming the agents in a distinct way;
- $\mathcal{S}'$ and $\mathcal{C}'$ are obtained correspondingly.

Moreover, the dag-size of $\mathcal{P}'$ is $\mathcal{O}(n \cdot m)$, where n is the dag-size of $\mathcal{P}$. Therefore, the secrecy problem for security protocols under finite session runs is NP-complete.

4.3 Normal Protocols

Another way to get decidability of the secrecy problem is to define an equivalence relation on terms so that arbitrarily large terms over finite sets $\mathcal{T}_0$ can be reduced to "small" terms. Such an equivalence relation was proposed in [20] but, unfortunately, the results developed in that paper are wrong (more details are provided later). In what follows we will present this equivalence relation and show how it can be used to get decidability of secrecy.

First, given a key K we define an unary operation on terms, denoted t_{-K}, called *key K removal*, as follows:

- $t_{-K} = t$, for any $t \in \mathcal{T}_0$;
- $(t, t')_{-K} = (t_{-K}, t'_{-K})$;
- $(\{t\}_{K'})_{-K} = t_{-K}$, if $K' = K$, and $(\{t\}_{K'})_{-K} = (\{t_{-K}\}_{K'})$, otherwise,

for any terms t and t'.

Define now the binary relation $\equiv$ as being the smallest binary relation on terms which is closed under symmetry, transitivity, pairing and encryption, and satisfies the following properties:

(A1) $t \equiv t$;

(A2) $(t, t) \equiv t$;

(A3) $(t, t') \equiv (t', t)$;

(A4) $(t, (t', t'')) \equiv ((t, t'), t'')$;

(A5) $\{t\}_K \equiv \{t_{-K}\}_K$, for any key K.

This equivalence relation ensures two main things:

- lists of terms are transformed into sets of terms (axioms (A2) and (A4));
- if we consider a finite number of keys, the depth of the encryption operator is bounded, because each key is used at most once in a given term (axiom (A5) and closure under pairing).

For a protocol signature $\mathcal{S}$ and two substitutions σ and σ', denote $\sigma \equiv \sigma'$ if $\sigma(x) \equiv \sigma'(x)$, for all $x \in \mathcal{T}_0$. Moreover, we write $(u, \sigma, i) \equiv (u', \sigma', i')$ if $u = u'$, $i = i'$, and $\sigma \equiv \sigma'$. Extend further this notation to sequences of events $\xi = e_1 \cdots e_k$ and $\xi' = e'_1 \cdots e'_k$, and write $\xi \equiv \xi'$ if $e_i \equiv e'_i$, for all $i \leq k$.

If there exists a proof of $t \equiv t'$ which does not use (A2) and (A5), we will write $t \equiv_1 t'$.

Definition 4.1 (1) A term t is called a *redex* if it has a subterm of the form (t', t') or of the form $\{t'\}_K$ with K encrypting in t'.

(2) A term t is called *normal* if it is not $\equiv_1$-equivalent with any redex.

As we can see, a redex is a term that can be reduced by (A2) or (A5); a normal term cannot be further reduced by (A2) and (A5).

The following property of $\equiv$ can be easily proved.

Lemma 4.1 Let $\mathcal{S}$ be a protocol signature. If $\mathcal{T}_0$ is finite, then $\equiv$ is an equivalence relation of finite index on $\mathcal{T}$.

An important consequence of the result above is that, if $\mathcal{T}_0$ is finite, then there exists an upper bound $B(\mathcal{T}_0)$ for the size of all normal terms.

Definition 4.2 An event $e = (u, \sigma, i)$ of a protocol $\mathcal{P}$ is called *normal* if $t(e)$ is normal. A sequence of events ξ of $\mathcal{P}$ is called *normal* if all events appearing in it are normal.

Directly from the definition of $\equiv$ we obtain

$$analz(T \cup \{t\}) \cap \mathcal{T}_0 = analz(T \cup \{t'\}) \cap \mathcal{T}_0,$$

for any $t \equiv t'$ and $T \subseteq \mathcal{T}$, which leads to:

Lemma 4.2 If $\mathcal{P}$ is a protocol and ξ and ξ' are runs of $\mathcal{P}$ such that $\xi \equiv \xi'$, then ξ is leaky iff ξ' is leaky.

Definition 4.3 Let $\mathcal{P}$ be a protocol and ξ a run of $\mathcal{P}$. We say that ξ is an $\equiv$-*normal run* of $\mathcal{P}$ if there exists a normal run ξ' of $\mathcal{P}$ such that $\xi \equiv \xi'$.

When for a protocol $\mathcal{P}$ only $\equiv$-normal runs are considered we will say that it is a *protocol under* $\equiv$-*normal runs* or a *normal protocol* [13]. The secrecy problem for normal protocols is formulated with respect to $\equiv$-normal runs only.

Theorem 4.5 The secrecy problem for normal protocols over finite sets of basic terms is decidable.

[13] Normal protocols were defined in [20] in a different way that is seriously flawed as it will be explained a little bit later. The definition we adopted follows the same line as the one for bounded and finite-session protocols.

Proof Let $\mathcal{P}$ be a normal protocol over a finite set $\mathcal{T}_0$ of basic terms. Because any run of $\mathcal{P}$ is equivalent to a normal run and normal runs are $(\mathcal{T}_0, B(\mathcal{T}_0))$-runs, we may consider that $\mathcal{P}$ is under $(\mathcal{T}_0, B(\mathcal{T}_0))$-runs. The theorem follows now from the results obtained for bounded protocols. □

Now, let us explain why the results in [20] are wrong. In that paper, the authors say that a protocol $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w = a_1 b_1 \cdots a_l b_l)$ is normal if w is normal. Then, they claimed that any leaky run of a normal protocol is equivalent to a leaky normal run of the same protocol. However, this claim is false. For example, if we consider the protocol

$$\begin{array}{lcl} A!B & : & (\{u\})u \\ B?A & : & u \\ B!C & : & \{u\}_K \\ C?B & : & \{u\}_K \\ C!D & : & (\{v\})v \\ D?C & : & v \\ D!E & : & K \\ E?D & : & K \end{array}$$

where K is long-term key shared by all honest participants, then we can see that it is a normal protocol in the sense that each term in the protocol is normal.

Consider now five substitutions:

- σ_1 and σ_5 are the identity functions;
- $\sigma_2(u) = \{n_I\}_{K_I}$, $\sigma_2(K) = K_I$, and $\sigma_2(x) = x$ for all $x \neq u, K$;
- $\sigma_3 = \sigma_2$;
- $\sigma_4(v) = \{\{\{n_I\}_{K_I}\}_K\}_{K_I}$ and σ_4 is the identity for all the other elements.

Applying $w|_A$ under σ_1, $w|_B$ under σ_2, $w|_C$ under σ_3, $w|_D$ under σ_4, and $w|_E$ under σ_5, we obtain the following leaky run ξ (for simplicity, it is written in the protocol notation):

$$\begin{array}{lcl} A!B & : & (\{u\})u \\ B?A & : & \{n_I\}_{K_I} \\ B!C & : & \{\{n_I\}_{K_I}\}_K \\ C?B & : & \{\{n_I\}_{K_I}\}_K \\ C!D & : & (\{v\})v \\ D?C & : & \{\{\{n_I\}_{K_I}\}_K\}_{K_I} \\ D!E & : & K \\ E?D & : & K \end{array}$$

It is easy to see that there is no leaky normal run equivalent to ξ because the term in the sixth event of the normal run should be $\{\{n_I\}_K\}_{K_I}$. However, the intruder can neither generate nor obtain the term $\{n_I\}_K$.

We close the section by recalling another approach proposed in [25]. In that paper, normal events are defined as events $e = (u, \sigma, i)$ for which $\sigma(x)$ is normal for all x.

Under this definition for normal events, it can be shown that secrecy is decidable for normal protocols over finite sets of basic terms.

We also mention that the two definitions for normal events, the one in [25] and the one in this paper, are incomparable.

Summing up, decidability of secrecy for normal protocols requires finitely many nonces but leaves the possibility for arbitrary-length messages.

4.4 Tagged Protocols

Tagging [11, 2] is a syntactic operation done on the textual representation of the protocol. This operation consists in adding message identifiers, also called *tags*, to some syntactic constructions (terms) in the protocol. A *tag* is an element of a given set, disjoint of the set of terms of the protocol in question, which is associated to a base type in the protocol. For instance, if an agent B receives a message $\{A, N_A\}_K$ encrypted by a key K unknown to B, then B will not be able to check whether or not the message inside the encryption has the form `(agent,nonce)`. By tagging this message by `(agent,nonce)_key`, B is told the structure of the message.

Tagging schemes can only not be used to prevent type flaw attacks [11], but also to obtain classes of protocols with a decidable secrecy property. Such a tagging scheme has been proposed in [22].

A pair of actions $A!B : (M)t$ and $B?A : t$ is called a *matching send-receive pair*. A sequence of actions $w = a_1 \cdots a_l$ of a protocol $\mathcal{P}$ is called *send-admissible* if any send action in the sequence is enabled at the state generated by the prefix which precedes the action in w. A *well-formed protocol* is a protocol $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w)$, where $w = a_1 b_1 \cdots a_l b_l$ is a send-admissible sequence of matching send-receive pairs $a_i b_i$ of actions.

Definition 4.4 A well-formed protocol $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w = a_1 b_1 \cdots a_l b_l)$ is called *tagged* if for any $i \leq l$ and $t \in ESub(t(a_i))$ there exists a constant $c_{t,i} \in \mathcal{C}$, and for any send action a_i there exists $n_i \in M(a_i)$ such that:

1. the constants $c_{t,i}$ are pair-wise distinct;
2. $(\forall i \leq l)(\forall t \in ESub(t(a_i)))(\exists u, K)(t = \{(c_{t,i}, (n_i, u))\}_K)$.

The first property says that distinct encrypted subterms have associated distinct constants. Moreover, if an encrypted subterm occurs in two distinct send actions, then it has distinct constants. The tagging scheme ensures that no two encrypted subterms of distinct actions are unifiable.

There is a significant difference between this tagging scheme and the one in [11]. First of all, this tagging scheme applies only to encrypted terms, but the tagging scheme in [11] applies to any element: agent, nonce, encrypted term, pair etc. On the other side, in [11], encrypted terms with same structure are tagged in the same way no matter where are the terms. In the tagging scheme in this paragraph, encrypted terms are tagged distinctly, taking into consideration the position where they occur as well. Due to this fact, this tagging scheme cannot be used to handle *blind copies* as in the Woo-Lam protocol [28], but this is possible with the tagging scheme in [11].

Definition 4.5 Let $\mathcal{P}$ be a tagged protocol and $\xi = e_1 \cdots e_k$ be a well-typed run of $\mathcal{P}$.

(1) e_j is a *good successor* of e_i (and e_i is a *good predecessor* of e_j) if $i < j$ and one of the following conditions holds:

- $e_i \rightarrow e_j$;
- e_i is a send event, e_j is a receive event, and $ESub(t(e_i)) \cap ESub(t(e_j)) \neq \emptyset$.

(2) e_i is a *good event in* ξ if either it is the last event or it has good successors.

(3) e_i is a *bad event in* ξ if e_i is not a good event in ξ.

(4) ξ is a *good run* if all its events are good.

(5) A *good path in* $\xi = e_1 \cdots e_k$ is any sequence $e_{i_1} \cdots e_{i_r}$ such that $e_{i_j} \in \{e_1, \cdots, e_k\}$ for all $j \leq r$ and $e_{i_{j+1}}$ is a good successor of e_{i_j} for all $j < r$.

Theorem 4.6 ([22]) The secrecy problem for tagged protocols is decidable.

Proof (sketch)
Let $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w)$ be a tagged protocol. The proof follows three main steps, whose proof rely havily on the fact that the protocol is tagged.

1. It is shown that if a protocol has a leaky run then it has a well-typed leaky run.

Given a substitution σ and a nonce z, define σ_z as being the substitution $\sigma_z(x) = z$, for all $x \in \mathcal{N}$ with $\sigma(x) \notin \mathcal{N}$, and $\sigma_z(x) = \sigma(x)$, otherwise.

Given a run ξ of $\mathcal{P}$, we denote by ξ' the run obtained from ξ by replacing each substitution σ wich appears in ξ by σ_{n_I}. It can be shown that ξ' is a run of $\mathcal{P}$ which is leaky iff ξ is leaky.

2. It is shown that if a protocol has a well-typed leaky run then it has a good leaky run.

Let $\xi = s_0[e_1\rangle \cdots s_{k-1}[e_k\rangle s_k$ be a well-typed leaky run of $\mathcal{P}$ such that no proper prefix of ξ is leaky. If ξ is not a good run then we modify it as follows:

a) remove the latest bad event in ξ. Let e_i be this event and ξ' be the sequence such obtained;

b) let $T = (analz(s_{iI}) - analz(s_{(i-1)I})) \cap \mathcal{T}_0$. It is easy to see that $T \subseteq M(e_i)$. Define the substitution σ that maps every $x \in T \cap \mathcal{N}$ into n_I, every $x \in T \cap \mathcal{K}_0$ into K_I and it is the identity otherwise, and apply it to ξ';

c) repeat a) and b) with $\xi = \xi'$ until ξ' is a good run.

The good run obtained in this way is leaky.

3. It is shown that all good runs are of bounded length with respect to the number of distinct events.

Let $\xi = e_1 \cdots e_k$ be a good run of $\mathcal{P}$. Note that there might be multiple occurrences of some event in ξ, but this is not relevant for leakiness analysis because the intruder can not deduce more information if some event have more than one occurrence.

Any good path of ξ has length at most $|w|$. Therefore, the set of events of ξ is partitioned into $E_1, \ldots, E_{|w|}$ where, for all $1 \leq i \leq |w|$, E_i is the set of events e from ξ such that the shortest good path in ξ that starts with e and ends with e_k has length i.

Because an event can have at most two good predecessors, any good run has length at most $2^{|w|} - 1$.

In conclusion, to verify that a tagged protocol is leaky it is sufficient to verify a bounded set of bounded-length runs. □

Summing up, decidability of secrecy for tagged protocols leaves the possibility for infinitely many nonces and arbitrary-length messages.

5 Conclusions

By adopting, with slight modifications, the formalism in [19], we have surveyed in this paper the most important results regarding the secrecy problem for security protocols. Some flawed statements found in the literature were corrected, and slight extensions were also proposed. By using the same formalism, the paper enjoy unity and clarity for anyone who wants to have a clear image about the secrecy problem.

Two prominent problems remain open:

1. the complexity of the secrecy problem for bounded protocols without freshness check;
2. the complexity of the (initial) secrecy problem for bounded protocols with freshness check.

References

[1] R.M. Amadio, D. Lugiez, V. Vanackere, On the symbolic reduction of processes with cryptographic functions, *Theoretical Computer Science* **290(1)** (2002), 695–740.

[2] B. Blanchet, A. Podelski, Verification of Cryptographic Protocols: Tagging Enforces Termination, *Theoretical Computer Science*, 2005 (to appear).

[3] M. Burrows, M. Abadi, R. Needham, A Logic of Authentication, *Proceedings of the Royal Society*, Series A, **426(1871)** (1989), 233–271.

[4] W. Charatonik, A. Podelski, J.M. Talbot, Paths vs. Trees in Set-based Program Analysis, *Proceedings of the 27th Annual ACM Symposium on Principles of Programming Languages* 2000, 330–338.

[5] H. Comon, V. Cortier, Tree Automata with One Memory, Set Constraints and Cryptographic Protocols, unpublished paper, 2001 (`http://www.lsv.ens-cachan.fr/~comon/biblio.html`).

[6] D. Dolev, A. Yao, On the Security of Public-Key Protocols, *IEEE Transactions on Information Theory* **29** (1983), 198–208.

[7] N. Durgin, J. Mitchell, Analysis of Security Protocols, *In "Calculational System Design, Series F: Computer and System Sciences", col. 173, IOS Press*, 1999, 369–395.

[8] N. Durgin, P. Lincoln, J. Mitchell, A. Scedrov, Undecidability of Bounded Security Protocols, *Workshop on Formal Methods and security Protocols FMSP'99*, Trento (Italy), July 5, 1999.

[9] N. Durgin, P. Lincoln, J. Mitchell, A. Scedrov, Multiset Rewriting and the Complexity of Bounded Security Protocols, *Journal of Computer Security* **12** (2004), 247–311.

[10] S. Even, O. Goldreich, On the Security of Multi-Party Ping-Pong Protocols, *Technical Report* **285**, Computer Science Department, Technion, Haifa (Israel), June 1983.

[11] J. Heather, G. Lowe, S. Schneider, How to Prevent Type Flaw Attacks on Security Protocols, *Journal of Computer Security*, Vol. **1192** (2003).

[12] J.E. Hopcroft, J.E. Ullman, Introduction to Automata, Languages and Combinatorics, *Addison Wesley*, 1979.

[13] G. Lowe, Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR, *In Proceedings of TACAS, Lecture Notes in Computer Science* **1055** (1996), 147–166.

[14] G. Lowe, Towards a Completeness Result for Model Checking of Security Protocols, *Journal of Computer Security* **7** (1999), 89–146.

[15] C. Meadows, A Model of Computation for the NRL Protocol Analyzer, *In Proceedings of the 7th Computer Security Foundations Workshop, IEEE Computer Society Press*, 1994, 84–89.

[16] J.K. Millen, V. Shmatikov, Constraint Solving for Bounded-process Cryptographic Protocol Analysis, *in Proc. of the ACM Conf. on Computer and Communications Security*, 2001, 166–175.

[17] R. Needham, M. Schroeder, Using Encryption for Authentication in Large Networks of Computers, *Communications of the ACM* **21(12)** (1978), 993–999.

[18] L.C. Paulson. Proving Properties of Security Protocols by Induction, *In Proceedings of the 10th Computer Security Foundations Workshop, IEEE Computer Society Press*, 1997, 70–83.

[19] R. Ramanujam, S.P. Suresh, A Decidable Subclass of Unbounded Security Protocols, *Proc. of WITS 2003*, April 2003, 11–20.

[20] R. Ramanujam, S.P. Suresh, An Equivalence on Terms for Security Protocols, *Proc. of AVIS 2003*, April 2003, 45–56.

[21] R. Ramanujam, S.P. Suresh, Undecidability of the secrecy problem for security protocols, manuscript, July 2003 (`http://www.imsc.res.in/~jam/`).

[22] R. Ramanujam, S.P. Suresh, Decidability of Secrecy for Tagged Protocols, manuscript, September 2003 (`http://www.imsc.res.in/~jam/`).

[23] M. Rusinowitch, M. Turuani, Protocol Insecurity with Finite Number of Sessions is NP-complete, *Theoretical Computer Science*, vol. **299** (2003), 451–475.

[24] S. Schneider, Verifying Authentication Protocols with CSP, *In Proceedings of the 10th Computer Security Foundations Workshop, IEEE Computer Society Press*, 1997, 3–17.

[25] S.P. Suresh, Foundations of Security Protocol Analysis, Ph.D. Thesis, University of Madras, Nov 2003.

[26] P. Syverson, C. Meadows, I. Cervesato, Dolev-Yao is no better than Machiavelli, *The 1st Workshop on Issues in the Theory of Security WITS 2000*, University of Geneva (Switzerland), July 2000.

[27] F.J. Thayer, J.C. Herzog, J.D. Guttman, Strand Spaces: Proving Security Protocols Correct, *Journal of Computer Security* **7** (1999), 191–230.

[28] T.Y.C. Woo, S.S. Lam, A Lesson on Authentication Protocol Design, *Operating Systems Review* **28(3)** (1994), 24–37.