Verifying a bounded number of sessions and its complexity

Michael RUSINOWITCH [a] and Mathieu TURUANI [a]
[a] *INRIA Nancy Grand Est*

Abstract. We investigate the complexity of the protocol insecurity problem for a finite number of sessions (fixed number of interleaved runs). We show that this problem is NP-complete with respect to a Dolev-Yao model of intruders. The result does not assume a limit on the size of messages and supports asymetric and non-atomic symmetric encryption keys. We also prove that in order to build an attack with a fixed number of sessions the intruder needs only to forge messages of linear size, provided that they are represented as DAGs.

1. Introduction

Although the general protocol verification problem is undecidable [13] even in the restricted case where the size of messages is bounded [12], it is interesting to investigate decidable fragments of the underlying logics and their complexity. The success of practical verification tools indicates that there may exist interesting decidable fragments that capture many concrete security problems. Dolev and Yao have proved that for simple ping-pong protocols, insecurity can be decided in polynomial time [10]. On the other hand [12] shows that when messages are bounded and when no nonces (i.e. new data) are created by the protocol and the intruder, then the existence of a secrecy flaw is decidable and DEXPTIME-complete.

A related decidability result is presented in [14,1,20,19,4,17,16]. The authors give a procedure for checking whether an unsafe state is reachable by the protocol. Their result holds for the case of finite sessions but with no bounds on the intruder messages. The proof in [1] does not allow general messages (not just names) as encryption keys. This limitation is relaxed in [20,19,21]. The decision algorithm presented in this chapter is similar to the one in [20] but its proof has been simplified.

The main result of this chapter states that for a fixed number of interleaved protocol runs, but with no bounds on the intruder messages, the existence of an attack is NP-complete. We allow public key encryption as well as the possibility of symmetric encryption with *composed keys* i.e. with any message. Here we only consider *secrecy* properties. However *authentication* can be handled in a similar way. Hence, a protocol is considered insecure if it is possible to reach a state where the intruder possesses a secret term.

With the same proof technique it is possible to extend the result directly to various intruder models and to protocols with choice points. In particular many algebraic properties of cryptographic primitives can be covered by the approach ([5,6]). The result we

present here can also be derived through a constraint solving approach and the careful design of simplification rules to be applied to intruder constraints ([8]).

Although this result is of a theoretical flavor, it gives information of practical relevance since for its proof we have shown that in order to attack a protocol an intruder needs only to forge messages of linear size with respect to the size of the protocol. This gives a low bound for the message space to be explored when looking for a flaw e.g. with a model-checker and this explains also why many tools like the CL-ATSE [22] backend from the AVISPA Tool [2] are effective in protocol analysis: to put it informally, in the Dolev-Yao model flawed protocols can be attacked with small faked messages.

Layout of the chapter: We first introduce in Section 2 our model of protocols and intruder and give the notion of *attack* in Section 2.4. Then in Section 3 we study properties of derivations with intruder rules. This allows us to derive polynomial bounds for normal attacks in Section 4 and to show that the problem of finding a normal attack is in NP. We show in Section 5 that the existence of an attack is NP-hard.

2. The Protocol Model

We consider a model of protocols in the style of [3]. The actions of any honest principal are specified as a partially ordered list that associates to (the format of) a received message its corresponding reply. The activity of the intruder is modeled by rewrite rules on sets of messages. We suppose that the initialization phase of distributing keys and other information between principals is implicit. The approach is quite natural and it is simple to compile a wide range of protocol descriptions into our formalism. For instance existing tools such as CAPSL [18] or CASRUL [15] would perform this translation with few modifications. We present our model more formally now.

2.1. Messages

The messages exchanged during the protocol execution are built using pairing $\langle \_, \_ \rangle$ and encryption operators $\{|\_|\}^{\mathsf{s}}_{\_}$, $\{|\_|\}^{\mathsf{a}}_{\_}$. We add a superscript to distinguish between public key (^{p}) and symmetric key (^{s}) encryptions. The set of basic messages is finite and denoted by $Atoms$. It contains names for principals and atomic keys from the set $Keys$. Since we have a finite number of sessions we also assume any nonce is a basic message: we consider that it has been created before the session and belongs to the initial knowledge of the principal that generates it.

Any message can be used as a key for symmetric encryption. Only elements from $Keys$ are used for public key encryption. Given a public key (resp. private key) k, k^{-1} denotes the associated private key (resp. public key) and it is an element of $Keys$.

The messages are then generated by the following (tree) grammar:

$$msg ::= Atoms \mid \langle msg, msg \rangle \mid \{|msg|\}^{\mathsf{a}}_{Keys} \mid \{|msg|\}^{\mathsf{s}}_{msg}$$

A signature, usually denoted by $[M]_{k'}$ with a private key k' ($= k^{-1}$), is represented here ase $\{|M|\}^{\mathsf{a}}_{k'}$. For conciseness we denote by $m_1, \ldots, m_n$ the set of messages $\{m_1, \ldots, m_n\}$. Given two sets of messages M and M' we denote by M, M' the union of their elements and given a set of messages M and a message t, we denote by M, t the set $M \cup \{t\}$.

Decomposition rules		Composition rules	
$L_d(\langle a,b\rangle):$	$\langle a,b\rangle \to a,b,\langle a,b\rangle$	$L_c(\langle a,b\rangle):$	$a,b \to a,b,\langle a,b\rangle$
$L_d(\{\!|a|\!\}^{\mathsf{a}}_K):$	$\{\!|a|\!\}^{\mathsf{a}}_K, K^{-1} \to \{\!|a|\!\}^{\mathsf{a}}_K, K^{-1}, a$	$L_c(\{\!|a|\!\}^{\mathsf{a}}_K):$	$a,K \to a,K,\{\!|a|\!\}^{\mathsf{a}}_K$
$L_d(\{\!|a|\!\}^{\mathsf{s}}_b):$	$\{\!|a|\!\}^{\mathsf{s}}_b, b \to \{\!|a|\!\}^{\mathsf{s}}_b, b, a$	$L_c(\{\!|a|\!\}^{\mathsf{s}}_b):$	$a,b \to a,b,\{\!|a|\!\}^{\mathsf{s}}_b$

Table 1. Intruder Rules

2.2. Intruder

In the Dolev Yao model [10] the intruder has the ability to eavesdrop, divert and memorize messages, to compose and decompose, to encrypt and decrypt when he has the key, to generate new messages and send them to other participants with a false identity. We assume here without loss of generality that the intruder systematically diverts messages, possibly modifies them and forwards them to the receiver under the identity of the official sender. In other words all communications are mediated by a hostile environment represented by the intruder. The intruder actions for modifying the messages are simulated by rewrite rules on sets of messages.

The set of messages S_0 represents the initial knowledge of the intruder. We assume that at least the name of the intruder $Charlie$ belongs to this set.

Intruder rules are divided in several groups, for composing or decomposing messages. These rules, which are described in Table 1, are the only one we consider in this chapter and any mentions of "rules" refer to *these* rules. In Table 1 and in the remaining of the chapter, a, b and c represent any message and K represents any element of Key. For instance, the rule with label $L_c(\langle a,b\rangle)$ replaces a set of messages a,b by the following set of messages $a,b,\langle a,b\rangle$.

The rewrite relation is defined by $E \to E'$ if there exists one rule $l \to r$ (from Table 1) such that l is a subset of E and E' is obtained by replacing l by r in E. We write $\to^*$ for the reflexive and transitive closure of $\to$. We denote the application of a rule R to a set E of messages with result E' by $E \to_R E'$. We write $L_c = \{L_c(a) \mid \text{ for all messages } a\}$, and L_d in the same way. We call *derivation* a sequence of rule applications $E_0 \to_{R_1} E_1 \to_{R_2} .. \to_{R_n} E_n$. The rules R_i for $i = 1..n$ are called the rules of this derivation D. We write $R \in D$ (abusively) to denote that R is one of the rules R_i, for $i = 1..n$, that has been used in the derivation D. We omit the subscripts R_i in the derivation D when they are not relevant to the discussion. We write $E \to^* E'$ if $E = E'$ or if there exists a derivation $E \to E_1 \to .. \to E'$.

One can remark that if the intruder was allowed to generate new data he will not get more power. He is already able to create infinitely many data only known to him with simple encryptions. For instance he can construct an infinite sequence of terms only known to him, like e.g. $\{\!|N|\!\}^{\mathsf{s}}_N$, $\{\!|\{\!|N|\!\}^{\mathsf{s}}_N|\!\}^{\mathsf{s}}_N$, ... assuming that N is only known by the intruder. For the class of protocols that we will consider honest principals receiving these terms for the first time cannot distinguish them from nonces. Alternatively, the result in this chapter can also be easily extended to allow nonces for the intruder, simply by processing them like any composed term only checkable by the intruder.

2.3. Protocols

We shall specify protocols by a list of actions for each principal. In order to describe the protocol steps we introduce message terms (or terms for short). We assume that we have a set of variables Var. Then the set of terms is generated by the following tree grammar:

$$term ::= Var \mid Atoms \mid \langle term, term \rangle \mid \{\!|term|\!\}^{\mathsf{a}}_{Keys} \mid \{\!|term|\!\}^{\mathsf{s}}_{term}$$

As for messages, a signature usually denoted as $[M]_{k'}$ with a private key $k' \in Keys$ is here represented as $\{\!|M|\!\}^{\mathsf{a}}_{k'}$. Let $Var(t)$ be the set of variables that occur in a term t. A *substitution* assigns terms to variables. A *ground substitution* assigns messages to variables. The application of a substitution σ to a term t is written $t\sigma$. We also write $[x \leftarrow u]$ the substitution σ defined by $\sigma(x) = u$ and $\sigma(y) = y$ for $y \neq x$. The set of subterms of t is denoted by $Sub(t)$. These notations are extended to sets of terms E in a standard way. For instance, $E\sigma = \{t\sigma \mid t \in E\}$.

A principal (except the initiator) reply after receiving a message matching a specified term associated to its current state. Then from the previously received messages (and initial knowledge) he builds the next message he will send. This concrete message is obtained by instantiating the variables in the message pattern associated to the current step in the protocol specification.

A protocol is given with a finite set of principal names $Names \subseteq Atoms$, and a partially ordered list of steps for each principal name. This partial order aims at ensuring that the actions of each principal are performed in the right order. More formally we associate to each principal A a partially ordered finite set $(W_A, <_{W_A})$. Each protocol step is specified by a pair of terms denoted $R \Rightarrow S$ and is intended to represent some message R expected by a principal A and his reply S to this message. Hence a protocol specification P is given by a set of *steps*:

$$\{(\iota, R_\iota \Rightarrow S_\iota) \mid \iota \in \mathcal{I}\}$$

where $\mathcal{I} = \{(A, i) \mid A \in Names \text{ and } i \in W_A\}$ and R_ι and S_ι are terms. Given a protocol, we will assume that for all $(A, i) \in \mathcal{I}$, for all $x \in Var(S_{(A,i)})$, there exists $j \leq_{W_A} i$ (i.e. $j <_{W_A} i$ or $j = i$) such that $x \in Var(R_{(A,j)})$, meaning that any variable must be received before it is sent.

We write $|\mathcal{I}|$ for the size of $\mathcal{I}$. $Init$ and End are fixed messages used to initiate and close a protocol session. An *environment* for a protocol is a set of messages. A *correct execution order* π is a one-to-one mapping $\pi : \mathcal{I}' \to \{1, .., |\mathcal{I}'|\}$ such that 1) $\mathcal{I}' \subseteq \mathcal{I}$; 2) and for all $A \in Names$ and $i <_{W_A} j$ with $(A, j) \in \mathcal{I}'$, we have $(A, i) \in \mathcal{I}'$ and $\pi(A, i) < \pi(A, j)$. In other words π defines an execution order for the protocol steps. This order is compatible with the partial order of each principal. It is not supposed to be complete, but it must be acceptable (i.e. runnable) for principals. We write $|\mathcal{I}'|$ for the size of $\mathcal{I}'$ in a correct execution order. For the readability of examples, we allow writing protocol steps as $r_1, .., r_p \Rightarrow s_1, .., s_q$ instead of $\langle r_1, \langle .., r_p \rangle \rangle \Rightarrow \langle s_1, \langle .., s_q \rangle \rangle$. A *protocol execution* is given by a ground substitution σ, a correct execution order π and a sequence of environments $E_0, .., E_{|\mathcal{I}'|}$ verifying: $Init \in E_0$, $End \in E_{|\mathcal{I}'|}$, and for all $1 \leq k \leq |\mathcal{I}'|$, $R_{\pi^{-1}(k)}\sigma \in E_{k-1}$ and $S_{\pi^{-1}(k)}\sigma \in E_k$.

Each step ι of the protocol extends the current environment by adding the corresponding message $S_\iota\sigma$ when $R_\iota\sigma$ is present. One can remark that principals are not allowed to generate any new data such as nonces. But this is not a problem when the number of sessions is finite: in this setting from the operational point it is equivalent to assume that the new data generated by a principal during a protocol execution is part of his initial knowledge.

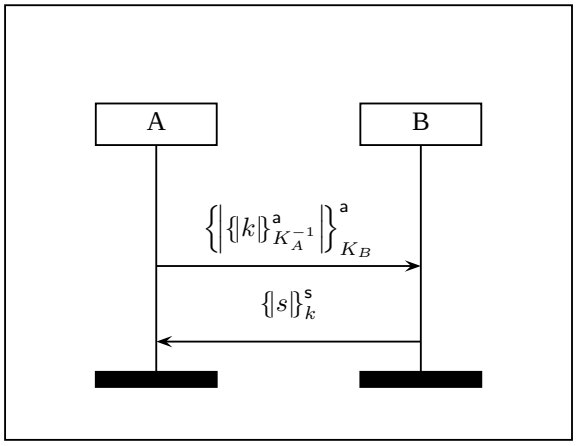

Figure 1. Handshake Protocol

Example: Handshake protocol

We recall this simple protocol (from Chapter *"Introduction"*) and its presentation by a message sequence chart. k is a symmetric key, K_B a public key, K_A^{-1} a private key associated to the public key K_A and s is a basic message.

Now we can express this protocol in our formal syntax. The orderings on steps is trivial since $W_A = W_B = \{1\}$.

$$\begin{array}{lll} ((\text{A},1), & Init & \Rightarrow \left\{\left|\{|k|\}^{\mathsf{a}}_{K_A^{-1}}\right|\right\}^{\mathsf{a}}_{K_B} \quad) \\ ((\text{B},1), & \left\{\left|\{|x|\}^{\mathsf{a}}_{K_A^{-1}}\right|\right\}^{\mathsf{a}}_{K_B} & \Rightarrow \{|s|\}^{\mathsf{s}}_{x} \quad) \end{array}$$

2.4. Attacks: Passive and active cases

Considering a protocol specification and a special term $Secret$ (called secret term), we say that there is an attack in N protocol sessions if the intruder can obtain the secret term in its knowledge set after completing at most N sessions.

2.4.1. Passive case

We consider first the case when the intruder has only the capability of eavesdropping messages between honest participants. Hence the intruder cannot intercept and modify the transmitted messages. If he can derive the secret with these limited means then we say that the protocol is subject to a *passive attack.*

We introduce a predicate $forge$ for checking whether a message can be constructed by the intruder from some known messages.

Definition 1 ($forge$**)** *Let E be a set of terms and let t be a term such that there is E' with $E \rightarrow^* E'$ and $t \in E'$. Then we say that t is* forged *from E and we denote it by $t \in forge(E)$.*

The existence of a passive attack can be reduced to a deduction problem:

Definition 2 (passive attack) *If E contains the initial intruder knowledge and the set of messages eavesdropped from the protocol sessions, there is a passive attack if $Secret \in forge(E)$.*

2.4.2. Active case

Let us consider now the general case where the intruder has full power. Note that received messages are matched by principals with the left-hand sides of protocol steps, meaning that some substitution unifies the messages sent by the intruder and the ones expected by the principals. Hence the existence of an attack can be expressed as a constraint solving problem: is there a way for the intruder to build from its initial knowledge and already sent messages a new message (defined by a substitution for the variables of protocol steps) that will be accepted by the recipient, and so on, until the end of the session, and such that at the end the secret term is known by the intruder.

In other words an active attack is a protocol execution where the intruder can modify each intermediate environment and where the message $Secret$ belongs to the final environment. In an active attack the intruder is able to forge any message by using its initial knowledge and already sent messages (spied in the environments). We note by k the cardinality of $\mathcal{I}'$ in an (active) attack. Hence we can formulate the definition of an attack using the predicate $forge$:

Definition 3 (active attack) *Given a protocol $P = \{R'_\iota \Rightarrow S'_\iota \mid \iota \in \mathcal{I}\}$, a secret message $Secret$ and assuming the intruder has initial knowledge S_0, an* active attack *(or an attack in short) is described by a ground substitution σ and a correct execution order $\pi : \mathcal{I}' \longrightarrow 1, \ldots, k$ such that for all $i = 1, \ldots, k$, we have $R_i\sigma \in forge(S_0, S_1\sigma, .., S_{i-1}\sigma)$ and $Secret \in forge(S_0, S_1\sigma, .., S_k\sigma)$ where $R_i = R'_{\pi^{-1}(i)}$ and $S_i = S'_{\pi^{-1}(i)}$.*

The definition of an active attack we give here is slightly more general than the one in [20] and was first proposed in [9] since it does not require the protocol to be completely executed. Before proceeding let us give some detailed examples.

Example: Attack on Handshake Protocol

Let us assume that Agent A initiates Handshake Protocol (see Chapter *"Introdution"*) with Intruder Charlie (C in short). Then using the first message from A, C can impersonate A and initiate a session with B. This allows C to get the message $s_{A,C}$ that should have been known only by A and B.

In order to detect the potential attack displayed in Figure 2, we set up a system built from two handshake protocol sessions running in parallel. This system can be considered as a single session for a protocol admitting 4 steps specified below:

$$
\begin{array}{lll}
((\text{A},1), & Init & \Rightarrow \left\{\left|\{\!|k_{A,C}|\!\}^{\mathsf{a}}_{K_A^{-1}}\right|\right\}^{\mathsf{a}}_{K_C} \quad) \\
((\text{C},1), & \left\{\left|\{\!|x|\!\}^{\mathsf{a}}_{K_A^{-1}}\right|\right\}^{\mathsf{a}}_{K_C} & \Rightarrow \{\!|s_{A,C}|\!\}^{\mathsf{s}}_{x} \quad) \\
((\text{A},1'), & Init & \Rightarrow \left\{\left|\{\!|k_{A,B}|\!\}^{\mathsf{a}}_{K_A^{-1}}\right|\right\}^{\mathsf{a}}_{K_B} \quad) \\
((\text{B},1'), & \left\{\left|\{\!|x'|\!\}^{\mathsf{a}}_{K_A^{-1}}\right|\right\}^{\mathsf{a}}_{K_B} & \Rightarrow \{\!|s_{A,B}|\!\}^{\mathsf{s}}_{x'} \quad)
\end{array}
$$

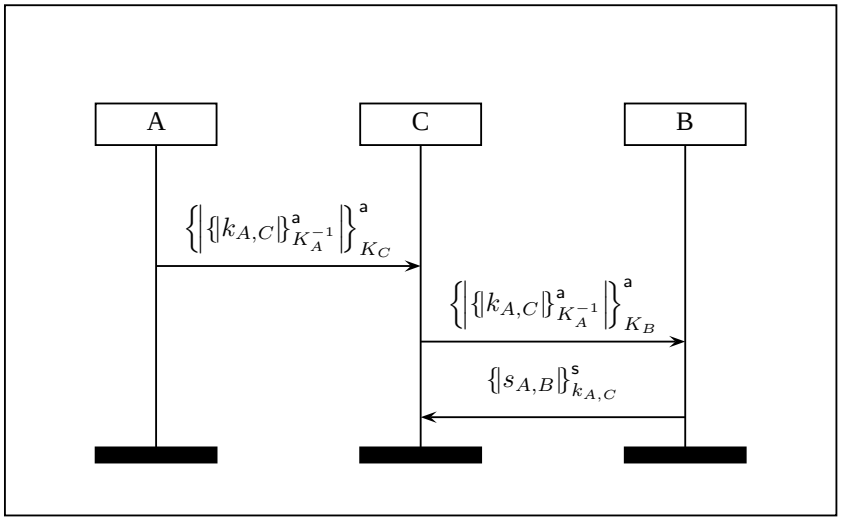

Figure 2. Attack on Simple Protocol

We assume that the initial intruder knowledge is $S_0 = \{K_B, K_A, K_C, K_C^{-1}, Init\}$. Then an attack is obtained by taking the following ground substitution and execution order:

Substitution	Protocol steps
$\sigma = [x \leftarrow k_{A,C}, x' \leftarrow k_{A,C}]$	$\pi(A,1) = 1,\ \pi(A,1') = 2,\ \pi(B,1') = 3$

This represents an attack since the following relations can be checked:

$$\begin{cases} R_{\pi^{-1}(1)}\sigma \in \mathit{forge}(S_0) \\ R_{\pi^{-1}(2)}\sigma \in \mathit{forge}(S_0, S_{\pi^{-1}(1)}\sigma) \\ R_{\pi^{-1}(3)}\sigma \in \mathit{forge}(S_0, S_{\pi^{-1}(1)}\sigma, S_{\pi^{-1}(2)}\sigma) \\ \quad s_{A,B} \in \mathit{forge}(S_0, S_{\pi^{-1}(1)}\sigma, S_{\pi^{-1}(2)}\sigma, S_{\pi^{-1}(3)}\sigma) \end{cases}$$

This can be verified easily. For instance, the last relation is valid since

$$K_A, K_C^{-1} \in S_0 \quad S_{\pi^{-1}(1)}\sigma = \left\{\left|\{\!|k_{A,C}|\!\}^{\mathsf{a}}_{K_A^{-1}}\right|\right\}^{\mathsf{a}}_{K_C} \quad S_{\pi^{-1}(3)}\sigma = \{\!|s_{A,B}|\!\}^{\mathsf{a}}_{k_{A,C}} \quad \text{and}$$

$$\begin{aligned} & K_A, K_C^{-1}, \left\{\left|\{\!|k_{A,C}|\!\}^{\mathsf{a}}_{K_A^{-1}}\right|\right\}^{\mathsf{a}}_{K_C}, \{\!|s_{A,B}|\!\}^{\mathsf{s}}_{k_{A,C}} \\ \rightarrow\ & K_A, K_C^{-1}, \left\{\left|\{\!|k_{A,C}|\!\}^{\mathsf{a}}_{K_A^{-1}}\right|\right\}^{\mathsf{a}}_{K_C}, \{\!|s_{A,B}|\!\}^{\mathsf{s}}_{k_{A,C}}, \{\!|k_{A,C}|\!\}^{\mathsf{a}}_{K_A^{-1}} \\ \rightarrow\ & K_A, K_C^{-1}, \left\{\left|\{\!|k_{A,C}|\!\}^{\mathsf{a}}_{K_A^{-1}}\right|\right\}^{\mathsf{a}}_{K_C}, \{\!|s_{A,B}|\!\}^{\mathsf{s}}_{k_{A,C}}, \{\!|k_{A,C}|\!\}^{\mathsf{a}}_{K_A^{-1}}, k_{A,C} \\ \rightarrow\ & K_A, K_C^{-1}, \left\{\left|\{\!|k_{A,C}|\!\}^{\mathsf{a}}_{K_A^{-1}}\right|\right\}^{\mathsf{a}}_{K_C}, \{\!|s_{A,B}|\!\}^{\mathsf{s}}_{k_{A,C}}, \{\!|k_{A,C}|\!\}^{\mathsf{a}}_{K_A^{-1}}, k_{A,C}, s_{A,B} \end{aligned}$$

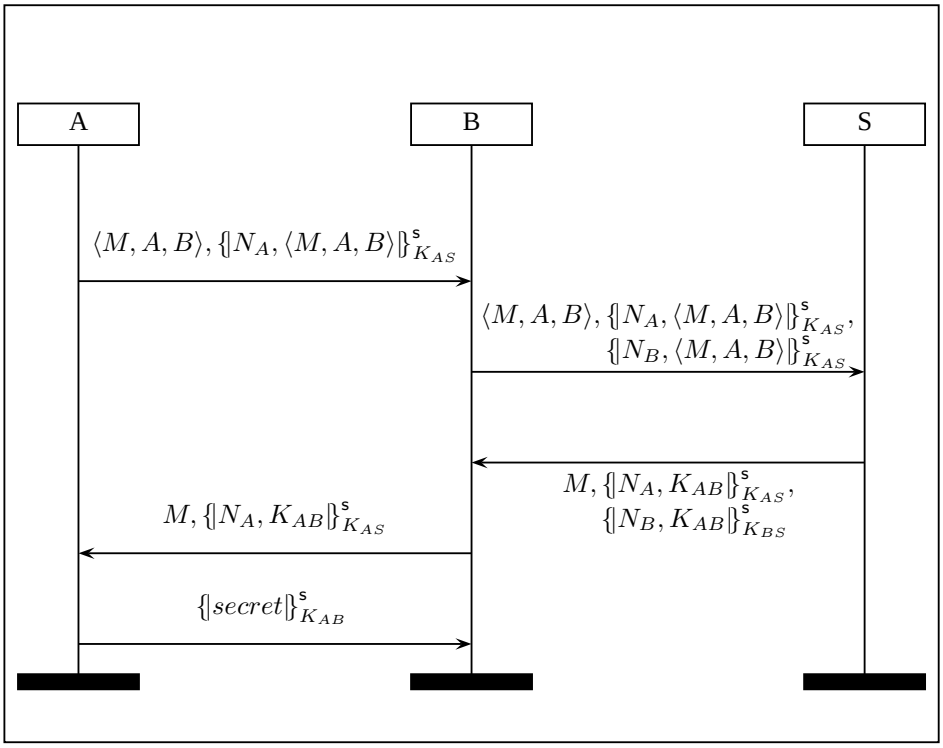

Figure 3. Otway-Rees Protocol

Example: Attack on Otway-Rees Protocol

The participants of the protocol are A, B and the server S. The symmetric keys Kas, Kbs will be respectively shared by the participants (A, S) and (B, S). The identifiers M, Na, Nb represents nonces. In Step 3, the server S creates the new secret symmetric key Kab to be used by A and B for further safe communications. We have extended the protocol with an extra step where A uses the key K_{AB} to send a secret message to B. In the attack, A will be fooled into believing that the term $\langle M, A, B\rangle$ (shorthand for $\langle\langle M, A\rangle, B\rangle$) is in fact the new key. The sequence of messages defining Otway-Rees is depicted in Figure 3. Let us write now this protocol specification with our notation, with m_{AB} as a short-hand for $\langle x_7, x_A, x_B\rangle$.

$$
\begin{array}{lll}
((A,1), & Init & \Rightarrow \langle M, A, B\rangle, \{\!|N_A, \langle M, A, B\rangle|\!\}^{\mathsf{s}}_{K_{AS}} \quad) \\
((B,1), & \langle x_2, x_3, B\rangle, x_4 & \Rightarrow x_2, x_3, B, x_4, \{\!|N_B, x_2, x_3, B|\!\}^{\mathsf{s}}_{K_{BS}} \quad) \\
((S,1), & m_{AB}, \{\!|x_8, m_{AB}|\!\}^{\mathsf{s}}_{K_{AS}}, \{\!|x_9, m_{AB}|\!\}^{\mathsf{s}}_{K_{BS}} & \Rightarrow x_7, \{\!|x_8, K_{ab}|\!\}^{\mathsf{s}}_{K_{AS}}, \{\!|x_9, K_{ab}|\!\}^{\mathsf{s}}_{K_{BS}} \quad) \\
 & m_{AB}, \{\!|x_8, m_{AB}|\!\}^{\mathsf{s}}_{K_{AS}}, \{\!|x_9, m_{AB}|\!\}^{\mathsf{s}}_{K_{BS}} & \Rightarrow x_7, \{\!|x_8, K_{ab}|\!\}^{\mathsf{s}}_{K_{AS}}, \{\!|x_9, K_{ab}|\!\}^{\mathsf{s}}_{K_{BS}} \quad) \\
((B,2), & x_2, x_5, \{\!|N_B, x_6|\!\}^{\mathsf{s}}_{K_{BS}} & \Rightarrow x_2, x_5 \quad) \\
((A,2), & M, \{\!|N_A, x_1|\!\}^{\mathsf{s}}_{K_{AS}} & \Rightarrow \{\!|Secret|\!\}^{\mathsf{s}}_{x_1} \quad) \\
((B,3), & \{\!|Secret|\!\}^{\mathsf{s}}_{x_6} & \Rightarrow end \quad)
\end{array}
$$

A protocol execution can be obtained by taking the protocol steps in the given order and by applying the following substitution:

$x_1 = K_{ab}$	$x_2 = M$	$x_3 = A$	$x_5 = \{\!\|x_8, K_{ab}\|\!\}^{\mathsf{s}}_{K_{AS}}$	$x_6 = K_{ab}$	$x_A = A$
$x_4 = \{\!\|\langle N_A, \langle M, A, B\rangle\rangle\|\!\}^{\mathsf{s}}_{K_{AS}}$		$x_7 = M$	$x_8 = N_A$	$x_9 = N_B$	$x_B = B$

An attack can be performed on this protocol with initial intruder knowledge $S_0 = \{Charlie, Init\}$, using:

Substitution	Protocol steps
$\sigma = [x_1 \leftarrow \langle M, A, B\rangle]$	$\pi(A,1) = 1, \pi(A,2) = 2$

since we have:

$$\begin{cases} R_{\pi^{-1}(1)}\sigma \in \mathit{forge}(S_0) \\ R_{\pi^{-1}(2)}\sigma \in \mathit{forge}(S_0, S_{\pi^{-1}(1)}\sigma) \\ \quad Secret \in \mathit{forge}(S_0, S_{\pi^{-1}(1)}\sigma, S_{\pi^{-1}(2)}\sigma) \end{cases}$$

3. Passive case

In this section we show how to decide efficiently the existence of a passive attack on a protocol. We first show some basic facts on the DAG-representation of message terms. Then we shall show how to obtain from any derivation a more compact one. We will then be able to prove that a normal attack has a polynomial size w.r.t. the size of the protocol and intruder knowledge, when using DAG representations.

3.1. DAG representation

The **DAG-representation** of a set E of message terms is the graph $(\mathcal{V}, \mathcal{E})$ with labeled edges, where:

- the set of vertices $\mathcal{V} = Sub(E)$, the set of subterms of E. The label of a vertice is the root symbol of the corresponding subterm ($\langle \_, \_ \rangle$, $\{\!|\_|\!\}^{\mathsf{s}}_{\_}$,$\{\!|\_|\!\}^{\mathsf{a}}_{\_}$ or an element from $Atoms$).
- the set of edges $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2$:
  $\mathcal{E}_1 = \{v_s \xrightarrow{down} v_e \mid \exists b,\ v_s = \{\!|v_e|\!\}^{\mathsf{s}}_b$ or $v_s = \{\!|v_e|\!\}^{\mathsf{a}}_b$ or $v_s = \langle v_e, b\rangle\}$
  $\mathcal{E}_2 = \{v_s \xrightarrow{up} v_e \mid \exists b,\ v_s = \{\!|b|\!\}^{\mathsf{s}}_{v_e}$ or $v_s = \{\!|b|\!\}^{\mathsf{a}}_{v_e}$ or $v_s = \langle b, v_e\rangle\}$

Let us note that the DAG representation is unique. Moreover if n is the number of elements in $Sub(t)$, one can remark that $(\mathcal{V}, \mathcal{E})$ has at most n nodes and $2.n$ edges. Hence its size is linear in n, and for convenience we shall define the DAG-size of E, denoted by $|E|_{DAG}$, to be the number of distinct subterms of E, i.e the number of elements in $Sub(E)$. For a term t, we simply write $|t|_{DAG}$ for $|\{t\}|_{DAG}$. We note that a standard DAG representation of t would use $O(n.log(n))$ space with $n = |t|_{DAG}$, but this difference with our DAG-size has no impact on the polynomial-time complexity of algorithms presented below. An example of DAG representation can be found in Figure 3.1.

Now, why do we define the DAG size for terms ? First, because it is a way to represent messages and thus protocols in a compact manner that strengthens the complexity result; And second, because the subterm sharing works quite well with the replacement of subterms by others, which will be the essence of the proof of our main property. Consequently, we uses DAG size also to measure the size of attacks, and we concentrate on the properties of smallest attacks called *normal attack* w.r.t the following definition :

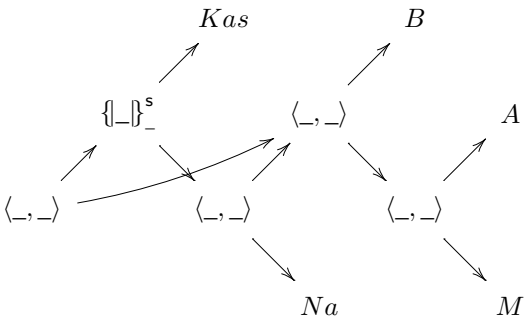

Figure 4. DAG Representation of the First Message of Otway-Rees Protocol

Definition 4 *Let $\mathcal{P}$ be a protocol and (σ, π) an attack on $\mathcal{P}$. Then (σ, π) is a normal attack iff it is minimal among all attacks on $\mathcal{P}$ w.r.t the following measure :*

$$l(\sigma, \pi) = |\{Charlie\} \cup \{\sigma(x)\}_{x \in Var}|_{DAG}$$

This is as expected except for the presence of the special atom $Charlie$. It ensures that a normal attack uses preferably $Charlie$ instead of any other atom or term, when it is possible, thus allowing us to express properties on normal attacks which we prove by replacing a subterm of σ by $Charlie$.

3.2. Simple lemmas on intruder derivations

In this section, we will give some useful definitions and properties of derivations. We shall introduce a notion of normal derivation, denoted by $Deriv_t(E)$. A related notion of normal derivation has been studied in [7]. Rather than a natural deduction presentation in [7] we use here term rewriting.

Definition 5 *Given a derivation $D = E_0 \rightarrow_{R_1} E_1 \rightarrow_{R_2} .. \rightarrow_{R_n} E_n$, a term t is a goal of D if $t \in E_n$ and $t \notin E_{n-1}$.*

For instance if $t \in forge(E)$ there exists a derivation with goal t: we take a derivation $D = E \rightarrow_{R_1} \ldots \rightarrow_{R_n} E'$ with $t \in E'$ and then we take the smallest prefix of D containing t. We will consider particular derivations minimal in length :

Definition 6 *We denote $Deriv_t(E)$ a derivation of minimal length among the derivations from E with goal t (chosen arbitrarily among the possible ones).*

In order to bound the length of such derivations, we can prove the two following lemmas: every intermediate term in $Deriv_t(E)$ is a subterm of E or t. Lemma 1 means that if a term is decomposed in a minimal derivation from E then this term has not been composed before and therefore belongs to E:

Lemma 1 *If there exists t' such that $L_d(t') \in Deriv_t(E)$ then t' is a subterm of E*

Proof:
Let $D = Deriv_t(E) = E_0 \rightarrow_{R_1} E_1 ... \rightarrow_{R_n} E_n$ and t' be (one of) the first occuring term not validating the lemma for D, assuming there exists one (or more). That is, there exists p such that $R_p = L_d(t')$, t' is not a subterm of E, $D_1 = E_0 \rightarrow_{R_1} ...E_{p-1}$, and for all $L_d(v) \in D_1$, v is a subterm of E. (Note: $p > 1$ since $t' \notin E$) Moreover, by minimality of D we have $L_c(t') \notin D$. Therefore, t' has necessarily be obtained by a decomposition rule in D_1, i.e. there exists a term u with t' subterm of u such that $L_d(u) \in D_1$. However, thanks to the minimality of t', this means that u is a subterm of E, and thus t' too. So the assumption is contradicted, and consequently and the lemma is proved. □

Lemma 2 means that in a minimal derivation we only need to compose subterms of the final goal or keys occurring in the initial set E:

Lemma 2 *If there exists t' such that $L_c(t') \in Deriv_t(E)$ then t' is a subterm of $\{t\} \cup E$*

Proof:
Let $D = Deriv_t(E) = E_0 \rightarrow_{R_1} E_1 ... \rightarrow_{R_n} E_n$ and t' be (one of) the last occuring term not validating the lemma for D, assuming there exists one (or more). That is, there exists p such that $R_p = L_c(t')$, t' is not a subterm of $\{t\} \cup E$, $D_2 = E_p \rightarrow_{R_{p+1}} ...E_n$, and for all $L_c(v) \in D_2$, v is a subterm of $\{t\} \cup E$. (Note: $p < n$ since $t' \neq t$) Moreover, by minimality of D, we have $L_d(t') \notin D$. Similarly, by minimality of D and since $t' \neq t$, there exists a rule in D_2 using t': otherwise $L_c(t')$ would be useless and D would not be minimal. Let us consider all possible rules using t' in D_2. Decomposition of t' is not allowed as pointed above; Decomposition of $\{|u|\}^{\mathsf{a}}_{t'^{-1}}$ is not allowed because t' is not an atom (since it is composed), and thus not in $Keys$; Decomposition of $\{|u|\}^{\mathsf{s}}_{t'}$ means that there exists $L_d(w) \in D_2$ such that t' is a subterm of w, and thus thanks to Lemma 1, t' is a subterm of E; And finally, composition using t' means that there exists $L_c(u) \in D_2$ with t' subterm of u, and thus by hypothesis both u and t' are subterms of $\{t\} \cup E$. Therefore, in every case the assumption is contradicted, and consequently the lemma is proved. □

We show in the next proposition that there always exist derivations of a term t from a set E with a number of rules bounded by the DAG-size of initial and final terms t, E. This will be very useful to bound the length of the derivations involved in a normal attack.

Proposition 1 *For any set of terms E and for any term t, if $Deriv_t(E) = E \rightarrow_{L_1} E_1 .. \rightarrow_{L_n} E_n$ then for all $1 \leq i \leq n$, $E_i \subseteq Sub(E_i) \subseteq Sub(t, E)$.*

Proof:
Let $Deriv_t(E) = E \rightarrow_{L_1} ... \rightarrow_{L_n} E_n$. By definition of $Deriv$, for any term t' subterm of $\cup_{i=1..n} E_i \setminus E$ there exists u with t' subterm of u such that either $L_d(u) \in D$ or $L_c(u) \in D$. Therefore, thanks to Lemmas 1 and 2 above, u (and thus t') is a subterm of $\{t\} \cup E$. It follows that for $i = 0..n$, $E_i \subseteq Sub(t, E)$, which proves the proposition. □

3.3. Decision Algorithm

We can now give a simple algorithm to decide whether $t \in forge(E)$. It is depicted in Algorithm 1, and assumes a representation of $Sub(E, t)$ as a DAG: anyway the DAG can be computed in polynomial time from inputs given in standard tree representations.

INPUTS: E, t represented by a DAG for $Sub(E,t)$
$K_1 \leftarrow E$
$K_0 \leftarrow Sub(E,t) \setminus E$
repeat
  $C \leftarrow \emptyset$
  for all $e \in Sub(E,t)$ **do**
    if $e \in K_0$, and $l, r \in K_1$, with $e = \{\!|l|\!\}^{\mathsf{s}}_r$ or $e = \{\!|l|\!\}^{\mathsf{a}}_r$ or $e = \langle l, r\rangle$ **then**
      $K_0 \leftarrow K_0 \setminus \{e\}$
      $K_1 \leftarrow K_1 \cup \{e\}$
      $C \leftarrow C \cup \{e\}$
    end if
    if $e \in K_1$, $l \in K_0$, $r \in K_1$, with $e = \{\!|l|\!\}^{\mathsf{s}}_r$ or $e = \{\!|l|\!\}^{\mathsf{a}}_r$ **then**
      $K_0 \leftarrow K_0 \setminus \{l\}$
      $K_1 \leftarrow K_1 \cup \{l\}$
      $C \leftarrow C \cup \{l\}$
    end if
    if $e \in K_1$ with $e = \langle l, r\rangle$ **then**
      $K_1 \leftarrow K_1 \cup \{l, r\}$
    end if
  end for
until $C = \emptyset$ or $t \in K_1$
if $t \in K_1$ **then**
  True
else
  False
end if

Algorithm 1. Passive Attack Decision

The correctness of Algorithm 1 is a consequence of the previous lemmas. We only need to compute the set K_1 of subterms of E, t that can be deduced from E and check whether t belongs to this set. The algorithm computes iteratively K_1 by checking for each term whether it can be derived from the current K_1 or can be decrypted (and if so its content is added to K_1).

The REPEAT loop can be entered at most $|Sub(E,t)|_{DAG}$ times. The FORALL loop is executed $|Sub(E,t)|_{DAG}$ times. The work done in this innermost loop is also bounded by a constant. Hence the algorithm takes $O(n^2)$ time where $n = |Sub(E,t)|_{DAG}$.

We now suggest a more efficient solution by reduction to Horn-Sat:

Proposition 2 *Given a set of terms E and a term t the existence of a passive attack can be decided in linear time.*

Proof:
Given $Sub(E,t)$, we consider each subterm $s \in Sub(E,t)$ as a propositional variable and we generate the following set of Horn clauses:

- for $s = \{|e|\}_b^{\mathsf{s}}$ or $s = \{|e|\}_b^{\mathsf{a}}$ we generate the clauses: $s, b \Rightarrow e$ and $e, b \Rightarrow s$;
- for $s = \langle l, r \rangle$ we generate the clauses $l, r \Rightarrow s$, $s \Rightarrow l$, and $s \Rightarrow r$;
- for $s \in E$ we generate s;

The union of all generated clauses and $\neg t$ is a set called $\mathcal{H}$. Now obviously the set of clauses $\mathcal{H}$ is satisfiable iff t cannot be derived by intruder rules from E. There exists a linear algorithm by Dowling and Gallier [11] to test the satisfiability of a set of Horn clauses, hence we can conclude. □

4. Active Case

4.1. Polynomial bound on attacks

We shall prove that when there exists an attack then a normal attack (see Definition 4) can always be constructed from subterms that are already occurring in the problem specification. This will allow to give bounds on the message sizes and on the number of rewriting rules involved in such an attack.

Let us consider a protocol $P = \{R'_\iota \Rightarrow S'_\iota \mid \iota \in \mathcal{I}\}$, a secret message $Secret$ and a set of messages S_0 as the initial intruder knowledge. We assume that there exists an attack described by a ground substitution σ and a correct execution order $\pi : \mathcal{I}' \longrightarrow 1, \ldots, k$ (where k is the cardinality of $\mathcal{I}'$). We define $R_i = R'_{\pi^{-1}(i)}$ and $S_i = S'_{\pi^{-1}(i)}$ for $i = 1, \ldots, k$. Note that R', S' now replace R, S for readability, unlike previous sections.

We also define: $\mathcal{SP}$ as the set of subterms of the terms in the set $\mathcal{P} = \{R_j \mid j = 1, \ldots, k\} \cup \{S_j \mid j = 0, \ldots, k\}$, and $\mathcal{SP}_{\leq i}$ the set of subterms of the terms in $\{R_j \mid j = 1, \ldots, i\} \cup \{S_j \mid j = 0, \ldots, i\}$.

The following Proposition 3 is a key property of this chapter. It shows that every substitution σ in a normal attack is only built with parts of the protocol specification. Using this, we will be able to prove that every substitution σ, and every message, in a normal attack have a DAG-size bounded by a polynomial in the protocol's DAG-size.

However, this relies on the technical Lemma 3. Informally, it states that if a term γ can be forged from a set of message E by composing two messages γ_1, γ_2 both derived from E then it is always possible to avoid decomposing γ in a derivation from E (with any goal t). Indeed, such a decomposition would generate messages γ_1, γ_2 that can be derived directly from E in another way.

Lemma 3 *Let $t \in forge(E)$ and $\gamma \in forge(E)$ such that $Deriv_\gamma(E)$ ends with an application of a rule in L_c. Then there is a derivation D with goal t starting from E, and verifying $L_d(\gamma) \notin D$.*

Proof:
Let $t \in forge(E)$ and $\gamma \in forge(E)$ be given with $Deriv_\gamma(E)$ ending with an application of a rule in L_c. Let D be $Deriv_\gamma(E)$ without its last rule, i.e. $Deriv_\gamma(E)$ is D followed by L_c. Let D' be the derivation obtained from $Deriv_t(E)$ by replacing every decomposition L_d of γ by D. Then D' is a correct derivation, since D generates α and β which are the two direct subterms of γ (γ is obtained by a composition). D does not contain a decomposition L_d of γ by definition of $Deriv_\gamma(E)$, since it already contains

a composition of γ. Hence D' build t from E, satisfies $L_d(\gamma) \notin D'$, and the lemma follows. □

We now state and prove our main property:

Proposition 3 *Given a normal attack (σ, π), for any variable x and any s subterm of $\sigma(x)$, we have $s \in \mathcal{SP}\sigma$.*

Proof:
We prove this proposition by contradiction. That is, let (σ, π) be a normal attack with $s \in Sub(Var(\mathcal{P})\sigma)$, and assume (*) $s \notin \mathcal{SP}\sigma$.

Let N be *minimal* such that $s \in Sub(R_N\sigma, S_N\sigma)$. It exists by definition of s, since there exists at least one protocol step containing the variable which value contains s. Consequently, there exists $y \in Var(R_N, S_N)$ such that s is a (strict) subterm of $\sigma(y)$, since otherwise $\exists t \in Sub(R_N, S_N)$ such that $t\sigma = s$, and (*) would be contradicted. Moreover, by definition of $\mathcal{P}$ any variable (including y) must be received before it is sent, which means here that $s \in Sub(R_N\sigma)$ since N is minimal.

It follows that $s \notin Sub(S_0\sigma, .., S_{N-1}\sigma)$ but $s \in Sub(R_N\sigma)$. However, we also know that $R_N\sigma \in \mathit{forge}(S_0\sigma, .., S_{N-1}\sigma)$, meaning that necessarily $L_c(s) \in D$ with $D = Deriv_{R_N\sigma}(S_0, .., S_{N-1}\sigma)$: only L_c rules can create new subterms along a derivation. We note that $s \notin Atoms$ (by definition of composition rules). By truncating D up to $L_c(s)$, it follows that :

$$s \in \mathit{forge}(S_0\sigma, .., S_{N-1}\sigma)$$

Second, we use the fact that s can be built from $S_0\sigma, .., S_{N-1}\sigma$ to create a new and smaller attack from σ. Let $D = Deriv_s(S_0\sigma, .., S_{N-1}\sigma)$. Thanks to the minimality of N, s is not a subterm of $S_0\sigma, .., S_{N-1}\sigma$, and thus, thanks to lemma 1 we know that D ends with a composition rule. We denote by $t\delta$ the replacement of every occurrence of s in a term t by $Charlie$. The replacement δ is extended to sets of terms, substitutions and rules in the natural way. The new attack we are interested in will be defined by $\sigma' = \sigma\delta$ and the same execution order π as for σ, at the condition that every messages $R_j\sigma'$ can still be build by the intruder from $S_0\sigma', .., S_{j-1}\sigma'$, for any j. This is what we prove now:

- Thanks to (*), $\forall t \in \mathcal{SP}$ including R_j and S_j for any j, we have $t\sigma' = t(\sigma\delta) = (t\sigma)\delta$. Thus, this is simply denoted as $t\sigma\delta$.
- If $j < N$, then $s \notin Sub(R_j\sigma, E_0)$ with $E_0 = S_0\sigma, .., S_{j-1}\sigma$, and thus, $R_j\sigma' \in \mathit{forge}(E_0\delta)$.
- If $j \geq N$, then thanks to Lemma 3 applied on D, there exists a derivation

  $$E_0 \rightarrow_{L_1} E_1 \rightarrow_{L_2} ... \rightarrow_{L_p} E_p$$

  with goal $R_j\sigma$, starting from $E_0 = S_0\sigma, .., S_{j-1}\sigma$, and such that $\forall i$, $L_i \neq L_d(s)$, i.e. where s is never decomposed. We show that for any $i = 1..p$, $E_i\delta \subseteq \mathit{forge}(E_{i-1}\delta)$:

  * If $L_i = L_c(t)$ with $t = \langle \alpha, \beta \rangle$ or $t = \{\!|\alpha|\!\}^{\mathsf{a}}_{\beta}$ or $t = \{\!|\alpha|\!\}^{\mathsf{s}}_{\beta}$, then either $s \neq t$ i.e. $E_i\delta \subseteq \mathit{forge}(E_{i-1}\delta)$ since $t\delta$ can be build from $\alpha\delta$ and $\beta\delta$, or $s = t$ i.e. $E_i\delta \subseteq \mathit{forge}(E_{i-1}\delta)$ since $Charlie \in S_0 \subseteq E_{i-1}\delta$.

* If $L_i = L_d(t)$ with $t = \langle \alpha, \beta \rangle$ or $t = \{|\alpha|\}^{\mathsf{a}}_{\beta}$ or $t = \{|\alpha|\}^{\mathsf{s}}_{\beta}$, then $s \neq t$ by construction of the derivation. Therefore, $t\delta$ is non-atomic and can be decomposed i.e. $E_{i-1}\delta \rightarrow_{L_d(t\delta)} E_i\delta$, meaning that $E_i\delta \subseteq forge(E_{i-1}\delta)$.

By iteration on i from 1 to p, it follows that $E_p\delta \subseteq forge(E_0\delta)$, and thus :

$$R_j\sigma' \in forge(S_0\sigma', .., S_{j-1}\sigma')$$

Consequently, (σ', π) defines an attack. However, this violates the minimality of (σ, π) as a normal attack (see Definition 4), since every occurrences of the non-atomic term s in σ were replaced by $Charlie$. Please note that $Charlie$ is counted in the size of an attack, even if it is not a subterm of σ. Also, s appears in σ at least once. Therefore, the hypothesis (*) is contradicted, which proves the proposition. □

We can now use this property to bound the DAG-size of every $\sigma(x)$, and even all the messages exchanged during a normal attack. This is shown in the following theorem:

Theorem 1 *If σ is the substitution in a normal attack, then for every $E \subseteq \mathcal{SP}$ we have $|E\sigma|_{DAG} \leq |\mathcal{P}|_{DAG}$. In particular, this holds for $E = Var$, or $E = \{R_i\} \cup S_0 \cup .. \cup S_{i-1}$ with $i = 1..k$, or $E = \{Secret\} \cup S_0 \cup .. \cup S_k$.*

Proof:
Let σ and E be as above. That is, $|E\sigma|_{DAG} = |Sub(E\sigma)| \leq |Sub(\mathcal{SP}\sigma)|$ with $|F|$ the number of elements in F. Let s be in $Sub(\mathcal{SP}\sigma)$. By construction, there exists $u \in \mathcal{SP}$ such that $s \in Sub(u\sigma)$. Therefore, we have two cases depending on where s occurs in $u\sigma$: either $s = u'\sigma$ with $u' \in Sub(u)$, i.e. $s \in \mathcal{SP}\sigma$; or $s \in Sub(\sigma(x))$ with $x \in Var(u)$, and thus thanks to Proposition 3, $s \in \mathcal{SP}\sigma$. Since in both cases we have $s \in \mathcal{SP}\sigma$, it follows that $Sub(\mathcal{SP}\sigma) \subseteq \mathcal{SP}\sigma$, and thus :

$$|E\sigma|_{DAG} \leq |\mathcal{SP}\sigma| \leq |\mathcal{SP}| = |\mathcal{P}|_{DAG}$$

since applying σ to a set preserves or reduces its cardinality. □

4.2. Decision Algorithm for Active Attacks

We are now going to present a NP decision procedure for finding an attack. The procedure amounts to guess a correct execution order π and a possible ground substitution σ with a polynomially bounded DAG-size, and finally to check using the algorithm presented before for passive attacks that at each protocol step, the intruder can produce the mesasge expected by honest participants.

We assume that we are given a protocol specification $\{(\iota, R'_\iota \Rightarrow S'_\iota) \mid \iota \in \mathcal{I}\}$. Let $P = \{R'_\iota, S'_\iota \mid \iota \in \mathcal{I}\}$, a secret message $Secret$ and a finite set of messages S_0 for initial intruder knowledge. If P, S_0 are not given in DAG-representation, they are first converted to this format (in polynomial time). We assume that the DAG-size of $P, S_0, Secret$ is n, and the finite set of variables occuring in P is V.

The procedure for checking the existence of an active attack is written in Algorithm 2. We discuss its correctness and complexity below, first for a single session then for the several ones.

1: INPUTS: Protocol specification P, set of terms S_0, and term $Secret$.
2: Guess a correct execution order $\pi : \mathcal{I}' \longrightarrow \{1, .., k\}$ with $k = |\mathcal{I}'|$
3: Guess a ground substitution σ such that $|V\sigma|_{DAG} \leq |P|_{DAG}$
4: Let $R_i = R'_{\pi^{-1}(i)}$ and $S_i = S'_{\pi^{-1}(i)}$ for $i \in \{1..k\}$
5: Let R_{k+1} be $Secret$.
6: For each $i \in \{1..k+1\}$, check that $R_i\sigma \in \mathit{forge}(\{S_j\sigma \mid j < i\} \cup \{S_0\})$ using Algorithm 1
7: If each check is successful then answer YES, otherwise answer NO.

Algorithm 2. NP Decision Procedure for the Insecurity Problem

4.2.1. Single Session Case

Let us first remark that the procedure in Algorithm 2 is NP. A correct execution order is a permutation of a subset of $\mathcal{I}$, and can be guessed in polynomial time. A ground substitution σ such that $|V\sigma|_{DAG} \leq n$ can be guessed in polynomial time : first, guess an ordered list of l nodes, $l \leq n$, and equip each of them with a label (either a binary operator from the term algebra, an atom in P or $Charlie$), plus two edges pointing to higher ordered nodes if the label was a binary operator, and none if it was an atom in P or $Charlie$; by construction this defines an acyclic graph of size at most polynomial in n, representing a set of terms T. This representation is not minimal as a DAG, but we reduce it to a DAG representation by eliminating duplicate nodes (i.e. with same label and same childs) in polynomial time. Now, we guess σ by choosing a term in T as the value of x for each variable x in V. Finally, we know from Section 3 that the passive attack decision algorithm is polynomial, and here it is used at most $n + 1$ times.

We can now see that this procedure is correct and complete since it answers YES if and only if the protocol has an *attack*. If an attack exists, then one of the smallest attacks on this protocol is a *normal attack*, defining a correct execution order and a ground substitution which are possible guesses for the algorithm since the passive attack decision algorithm is complete. On the other hand if the procedure answers YES, the verification performed on the guessed substitution proves that the protocol has an attack, since the passive attack decision algorithm is correct.

4.2.2. Multiple Sessions Case

We simulate the execution of several sessions of a protocol P by the execution of a single session for a more complex protocol P' of size polynomial in $|P| \times m$ where m is the number of sessions. Therefore this will reduce immediately the security problem for several sessions to the security problem for one session and will show that the insecurity problem is in NP for multiple sessions too. Note that the principals may have some common initial knowledge in different sessions. Hence the sessions are not necessarily disjoint.

We assume given a protocol specification P with its associated partial order $<$ on a set of steps W. Let m be the number of sessions of this protocol we want to study, let Var be the set of variables in P and let $Nonces$ be the set of nonces (a subset of $Atoms$) in P. The nonces are given fresh values at each new session by definition. Also variables from different sessions should be different. This is because we consider that

in this model messages are not memorized from one session to another (except maybe by the intruder). Therefore we shall define m renaming functions ν_i, for $i = 1..m$, as bijections from $W \cup Nonces \cup Var$ to m new sets (mutually disjoint and disjoint from $W \cup Nonces \cup Var$) such that:

$$\begin{array}{ll} \nu_i(w) = w_i & \text{for all } w \in W \\ \nu_i(N) = N_i & \text{for all } N \in Nonces \\ \nu_i(x) = x_i & \text{for all } x \in Var \end{array}$$

We assume that each set of steps W_i for $i = 1..m$, is provided with a partial order $<_i$ such that for all $w, w' \in W$ and for all $w_i, w'_i \in W_i$, $w < w'$ iff $w_i < w'_i$. Let P_i be the protocol obtained by applying the renaming ν_i to P. We have now m copy P_i, $i = 1..m$, of the protocol. We combine them now into a unique protocol denoted $m.P$ as follows. The set of steps is by definition the union $\bigcup_{i=1}^{m} W_i$ of the steps in all copies P_i, for $i = 1..m$. The partial order on $\bigcup_{i=1}^{m} W_i$ is defined as $\bigcup_{i=1}^{m} <_i$. It is easy to see that the execution of one session of the new protocol is equivalent to the execution of m interleaved sessions of the initial protocol.

Lemma 4 *Let S_0 be the initial intruder knowledge. The DAG-size of $(m.P, S_0)$ is $O(n \times m)$ where n is the DAG-size of P, S_0.*

Therefore a normal attack of $m.P$ can be bounded polynomially:

Corollary 1 *If σ is the substitution in a normal attack of $m.P$ assuming that the initial intruder knowledge is S_0 and the DAG-size of (P, S_0) is n, then σ can be represented in $O((n \times m)^2)$.*

Then applying the NP procedure for one session we derive immediately:

Theorem 2 *Protocol insecurity for a finite number of sessions is decidable and in NP.*

5. Complexity

We show now that the existence of an attack when the input are a protocol specification and initial knowledge of the intruder is NP-hard by reduction from 3-SAT. The proof is similar to the one given by [1] for their model, but does not need any conditional branching in the protocol specification. The propositional variables are $x_1, .., x_n = \overrightarrow{x}$, and an instance of 3-SAT is $f(\overrightarrow{x}) = \bigwedge_{i \in I}(x_{i,1}^{\varepsilon_{i,1}} \vee x_{i,2}^{\varepsilon_{i,2}} \vee x_{i,3}^{\varepsilon_{i,3}})$ where $\forall i, j \in I \times \{1..3\}$, $x_{i,j} \in \{x_1, .., x_n\}$, $\varepsilon_{i,j} \in \{0, 1\}$, and x^0 (resp.x^1) means x (resp.$\neg x$).

The idea of the reduction is to let the intruder generate a first message, $x_1, .., x_n$, representing a possible solution for this 3-SAT problem. From this initial message a principal A creates a term representing the instance of the formula f by this solution. Then the intruder will use the principals B, C, D as oracles for verifying that this instance can be evaluated to $\top$. In order to do it the intruder will have to *select* an adequate protocol execution where each principal checks the truth of a literal in a conjunct. For instance when the first literal of a conjunct is a propositional variable (resp. a negated variable), principal B checks whether this variable was assigned the value $\top$ (resp. $\bot$).

If the execution can be achieved then E gives the $Secret$ term to the intruder, and the protocol admits an attack.

Let us describe now the protocol. We introduce two atomic keys K and P, and an atomic term $\perp$ for representing the boolean value False. The encryption by K will encode negation, and thus, the boolean value True is represented by $\top = \{\!|\perp|\!\}^{\mathsf{s}}_K$. This coding of 3-SAT do not requires e.g. that $\{\!|\{\!|\perp|\!\}^{\mathsf{s}}_K|\!\}^{\mathsf{s}}_K$ reduce to $\perp$, so multiple encryption with K over $\perp$ do not represent any specific boolean value. The symmetric key P exists for storing data under process. Then we define :

- $g(0, x_{i,j}) = x_{i,j}$ and $g(1, x_{i,j}) = \{\!|x_{i,j}|\!\}^{\mathsf{s}}_K$.
- $f_i(\overrightarrow{x}) = \langle g(\varepsilon_{i,1}, x_{i,1}), \langle g(\varepsilon_{i,2}, x_{i,2}), g(\varepsilon_{i,3}, x_{i,3})\rangle\rangle$ for all $i \in I$

The protocol variables x, y, z occurring in the description of step (U, j) should be considered as indexed by (U, j); the index will be omitted for readability. The partial order on the protocol steps is the empty order. Hence the protocol steps can be executed in any order. Note also that the number of steps for each principal B, C, D is equal to the number of conjuncts in the 3-SAT instance.

Principal A: $(A, 1)$, $x_1, .., x_n \Rightarrow \{\!|\langle f_1(\overrightarrow{x}), \langle f_2(\overrightarrow{x}), \langle .., \langle f_n(\overrightarrow{x}), end\rangle\rangle\rangle\rangle|\!\}^{\mathsf{s}}_P$

Principal B: (B, i), $\{\!|\langle\langle\{\!|\perp|\!\}^{\mathsf{s}}_K, \langle x, y\rangle\rangle, z\rangle|\!\}^{\mathsf{s}}_P \Rightarrow \{\!|z|\!\}^{\mathsf{s}}_P$ for $i \in I$.

Principal C: (C, i), $\{\!|\langle\langle x, \langle\{\!|\perp|\!\}^{\mathsf{s}}_K, y\rangle\rangle, z\rangle|\!\}^{\mathsf{s}}_P \Rightarrow \{\!|z|\!\}^{\mathsf{s}}_P$ for $i \in I$.

Principal D: (D, i), $\{\!|\langle\langle x, \langle y, \{\!|\perp|\!\}^{\mathsf{s}}_K\rangle\rangle, z\rangle|\!\}^{\mathsf{s}}_P \Rightarrow \{\!|z|\!\}^{\mathsf{s}}_P$ for $i \in I$.

Principal E: $(E, 1)$, $\{\!|end|\!\}^{\mathsf{s}}_P \Rightarrow Secret$

We take $S_0 = \{\top, \perp\}$ as the initial intruder knowledge. Remember that $\top = \{\!|\perp|\!\}^{\mathsf{s}}_K$. By looking at this protocol, it is obvious that with such poor initial knowledge, there is an attack iff the message sent by principal A can be reduced to $\{\!|end|\!\}^{\mathsf{s}}_P$ i.e. for all $i \in I$, there exists $j \in \{1, 2, 3\}$ such that $g(\varepsilon_{i,j}, x_{i,j}) = \top$. But this means that the intruder has given to A a term representing a solution of 3-SAT, since $g(\varepsilon_{i,j}, x_{i,j})$ is ${x_{i,j}}^{\varepsilon_{i,j}}$. Hence the protocol admits an attack iff the corresponding 3-SAT problem has a solution. Moreover this reduction is obviously polynomial. Hence the problem of finding an attack with bounded sessions is NP-hard.

The example above shows that the insecurity problem is NP-hard for protocols with pairs, but without composed keys and without variables in key positions. But we can obtain hardness for a class of protocols with different restrictions. The next protocol shows that the insecurity problem remains NP-hard even without pairs, without composed keys and with a unique honest principal whose steps are linearly ordered. On the other hand we need to use variables at key positions.

Hence our next result will show that finding an attack to a single session of a sequential protocol is already an NP-hard problem. Therefore the non-determinism of the intruder is sufficient for the insecurity problem to be NP-hard.

Let $f(\overrightarrow{x}) = \bigwedge_{i=1}^{m} D_j$ be an instance of 3-SAT following the same definition as above (for the first protocol), and let n be the number of propositional variables of $\overrightarrow{x}$. In the following, x and y are protocol variables and we suppose that their occurrences represent different variables in different steps of the protocols i.e. they are implicitly indexed by the protocol steps. To each propositional variable x_j we associate an atom V_j, for $j = 1, \ldots, n$. The initial intruder knowledge includes only these terms:

1. $\{|\{|P|\}^{\mathsf{s}}_{\bot}|\}^{\mathsf{s}}_{K}$, $\{|\{|P|\}^{\mathsf{s}}_{\top}|\}^{\mathsf{s}}_{K}$, and P. The intruder will assign boolean values to $\overrightarrow{x}$ by using $\{|P|\}^{\mathsf{s}}_{\bot}$ or $\{|P|\}^{\mathsf{s}}_{\top}$.
2. $\{|\{|K|\}^{\mathsf{s}}_{\bot}|\}^{\mathsf{s}}_{V_j}$ and $\{|\{|K|\}^{\mathsf{s}}_{\top}|\}^{\mathsf{s}}_{V_j}$, for $j = 1..n$. These are faked values for $\overrightarrow{x}$ allowing the intruder to "skip" some protocol steps when needed. But doing that, he will not gain any useful knowledge.

We use only one honest principal A, and the protocol steps of A are linearly ordered by: $(A,(i,j)) < (A,(i',j'))$ iff $i < i'$ or $i = i'$ and $j < j'$, for $i = 0, \ldots, m+1$ and $j = 1, \ldots, n$:

$$(A,(0,j)): \quad \{|x|\}^{\mathsf{s}}_{K} \Rightarrow \{|x|\}^{\mathsf{s}}_{V_j}$$

In these steps, the intruder selects values for $\overrightarrow{x}$. Since there is one and only one step for each value V_j, the instantiation of $\overrightarrow{x}$ is complete and non redundant. Since the intruder does not know K, these values can only be $\{|P|\}^{\mathsf{s}}_{\bot}$ or $\{|P|\}^{\mathsf{s}}_{\top}$.

For each conjunct indices i , $1 \leq i \leq m$, and for each j , $1 \leq j \leq n$, such that x_j is a variable in the conjunct D_i, let us define the step $(A,(i,j))$ as :

$$(A,(i,j)): \quad \{|\{|y|\}^{\mathsf{s}}_{\top}|\}^{\mathsf{s}}_{V_j} \Rightarrow \{|Secret_i|\}^{\mathsf{s}}_{y} \quad \text{if } x_j \text{ occurs positively in } D_i \text{ or}$$
$$(A,(i,j)): \quad \{|\{|y|\}^{\mathsf{s}}_{\bot}|\}^{\mathsf{s}}_{V_j} \Rightarrow \{|Secret_i|\}^{\mathsf{s}}_{y} \quad \text{if } x_j \text{ occurs negatively in } D_i$$

The goal of the intruder is to know all terms $Secret_i$: this would prove that every conjunct D_i is evaluated to $\top$. To do this, he must use for y a value he knows in order to decrypt at least one message $\{|Secret_i|\}^{\mathsf{s}}_{y}$ for each i. However the intruder has only two possible actions: either he sends to A the message $\{|\{|K|\}^{\mathsf{s}}_{\top}|\}^{\mathsf{s}}_{V_j}$ or the message $\{|\{|K|\}^{\mathsf{s}}_{\bot}|\}^{\mathsf{s}}_{V_j}$ but then he will receive back $\{|Secret_i|\}^{\mathsf{s}}_{K}$ which is useless (this step can be considered as blank for the intruder), or he has assigned to V_j the correct value $\{|\{|P|\}^{\mathsf{s}}_{\top}|\}^{\mathsf{s}}_{V_j}$ or $\{|\{|P|\}^{\mathsf{s}}_{\bot}|\}^{\mathsf{s}}_{V_j}$, and by sending it to A at the right step he will get back $\{|Secret_i|\}^{\mathsf{s}}_{P}$ that he can decrypt with P to get $Secret_i$.

The last protocol step is to ensure that the intruder knows each $Secret_i$. For this purpose let us introduce an atom $BigSecret$ that will be revealed to the intruder iff he knows every atom $Secret_i$. The last protocol step is:

$$(A,(m+1,0)): \quad P \Rightarrow \{|..\{|BigSecret|\}^{\mathsf{s}}_{Secret_1}..|\}^{\mathsf{s}}_{Secret_m}.$$

Therefore, the intruder knows $BigSecret$ if and only if each conjunct D_j is evaluated to $\top$, and this protocol has an attack on $BigSecret$ if and only if the 3-SAT instance admits a solution. This shows the correctness of the reduction, which is obviously polynomial.

It is interesting to see that the class of protocols considered in the previous reduction is very close to the simple class of ping-pong protocols [10]: the only difference is the use of variables as keys (but these variables can take only atomic values).

Decomposition rules	Composition rules
$L_s(t):\ t \to a, t$ with $t = \{\mid\{\mid a\mid\}_b^{\mathsf{s}}\mid\}_b^{\mathsf{s}}$	$L_r(t):\ a \to a, t$ with $t = \{\mid\{\mid a\mid\}_b^{\mathsf{s}}\mid\}_b^{\mathsf{s}}$
$L_s(t):\ t \to a, t$ with $t = \{\mid\{\mid a\mid\}_K^{\mathsf{a}}\mid\}_{K^{-1}}^{\mathsf{a}}$	$L_r(t):\ a \to a, t$ with $t = \{\mid\{\mid a\mid\}_K^{\mathsf{a}}\mid\}_{K^{-1}}^{\mathsf{a}}$

Table 3. Extension of the Intruder Model.

From the results above we finally conclude with the main result:

Theorem 3 *Finding an attack for a protocol with a fixed number of sessions is an NP-complete problem.*

6. Conclusion

By representing messages as DAGs we have been able to prove that when the number of sessions of a protocol is fixed, an intruder needs only to forge messages of linear size in order to find an attack. This result admits obvious practical implications since it gives an a priori bound on the space of messages to be explored for finding flaws in the protocol (with a model-checker, for instance). We have then derived an NP-procedure for finding an attack with a fixed number of sessions and composed keys. This result matches the lower bound of the problem. Several interesting variants of our model can be reduced to it. These variants are also easy to implement.

First we could consider that a principal is unable to recognize that a message supposed to be encrypted by some key K has really been constructed by an encryption with K. This can be exploited to derive new attacks: if we assume that the rules in Table 3 are added to the intruder model, then new attacks can now be performed as shown in the following example (the message end is omitted).

Protocol Rules	Attack
$((A,0),\quad init \Rightarrow \{\mid Secret\mid\}_P^{\mathsf{a}}$	
$((A,1),\ \{\mid x\mid\}_{K^{-1}}^{\mathsf{a}} \Rightarrow x)$	$\sigma(x) = \{\mid\{\mid Secret\mid\}_P^{\mathsf{a}}\mid\}_K^{\mathsf{a}}$
$((A,2),\ \{\mid\{\mid y\mid\}_P^{\mathsf{a}}\mid\}_K^{\mathsf{a}} \Rightarrow y)$	$\sigma(y) = Secret$

since $\{\mid Secret\mid\}_P^{\mathsf{a}} \to_{L_r} \{\mid Secret\mid\}_P^{\mathsf{a}}, \{\mid\{\mid\{\mid Secret\mid\}_P^{\mathsf{a}}\mid\}_K^{\mathsf{a}}\mid\}_{K^{-1}}^{\mathsf{a}}$
and $(A,1)$ produces $\{\mid\{\mid Secret\mid\}_P^{\mathsf{a}}\mid\}_K^{\mathsf{a}}$
and $(A,2)$ produces $Secret$

Obviously, such an attack cannot be performed if the L_r rules are not included in the intruder rewrite system. Since simple cryptographic systems verify the property that encryption is idempotent, it might be interesting to add these new L_r rules. Moreover, it is in fact quite easy to prove that the insecurity problem remains NP-complete when these L_r and L_s rules are included: they behave exactly in the same way as L_c and L_d, allowing us again to restrict ourselves to consider only some special derivations. See [20] for more informations on this.

We have only considered secrecy properties so far. However, correspondence properties like authentication, integrity, some kinds of fairness, etc.. can also be expressed using an appropriate execution order and a polynomial number of $forge$ constraints. Thus, they can also be detected in NP time.

	Without Nonces	With Nonces
No bounds [13]	Undecidable	Undecidable
Infinite number of sessions, and bounded messages [12]	DEXPTIME-complete	Undecidable
Finite number of sessions, and unbounded messages	NP-complete	NP-complete

Table 4. Known results

Moreover, protocols with more complex control can also be decided in NP, as long as executions can be described in polynomial space and checked on a polynomial number of protocol steps in polynomial time. In particular, branching or synchronizations are not a problem, thus allowing honest participants to make choices.

Finally, let us notice that our model remains valid when the intruder is allowed to generate any number of new data: we simply replace in an attack all data that is freshly generated by the intruder by its name $Charlie$. Since honest agents do not test inequalities, especially inequalities between data obtained from the intruder, all the constraints are satisfied and the attack still works. Moreover, even allowing inequalities is possible : since in each attack at most a polynomial number of inequalities can be performed, at most a polynomial number of nonces are required, and thus, they can be picked up from the (finite) set of Atoms. This implies that in the finite session case, the intruder does not gain any power by creating nonces. We can summarize the known results in the Table 4.

More extensions of the presented results, especially by adding algebraic properties to the protocol and intruder models, can be found e.g. in [5,6,16].

References

[1] R. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In C. Palamidessi, editor, *Proceedings of Concur*, LNCS 1877, pages 380–394. Springer-Verlag, 2000.

[2] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA Tool for the automated validation of internet security protocols and applications. In K. Etessami and S. Rajamani, editors, *17th International Conference on Computer Aided Verification, CAV'2005*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285, Edinburgh, Scotland, 2005. Springer.

[3] I. Cervesato, N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop: CSFW'99*, pages 55–69. IEEE Computer Society Press, 1999.

[4] Y. Chevalier. *Résolution de problèmes d'accessibilité pour la compilation et la validation de protocoles cryptographiques*. Phd thesis, Université Henri Poincaré, Nancy, deecember 2003.

[5] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP Decision Procedure for Protocol Insecurity with XOR. In *Proceedings of the Logic In Computer Science Conference, LICS'03*, pages 261–270, 2003. `http://www.avispa-project.org`.

[6] Y. Chevalier, R. Küsters, M. Rusinowitch, M. Turuani, and L. Vigneron. Deciding the Security of Protocols with Diffie-Hellman Exponentiation and Products in Exponents. In *Proceedings of FSTTCS'2003*, Lecture Notes in Computer Science. Springer-Verlag, 2003.

[7] E. M. Clarke, S. Jha, and W. R. Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *Programming Concepts and Methods, IFIP TC2/WG2.2,2.3 International Conference on Programming Concepts and Methods (PROCOMET '98) 8-12 June 1998, Shelter Island, New York, USA*, volume 125 of *IFIP Conference Proceedings*, pages 87–106. Chapman & Hall, 1998.

[8] H. Comon-Lundh, V. Cortier, and E. Zalinescu. Deciding security properties for cryptographic protocols. application to key cycles. *ACM Trans. Comput. Log.*, 11(2), 2010.

[9] R. Corin and S. Etalle. An Improved Constraint-Based System for the Verification of Security Protocols. In *Proceedings of SAS 2002*, LNCS 2477, pages 326–341. Springer-Verlag, 2002.

[10] D. Dolev and A. Yao. On the Security of Public-Key Protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.

[11] W. F. Dowling and J. H. Gallier. Continuation semantics for flowgraph equations. *Theor. Comput. Sci.*, 44:307–331, 1986.

[12] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proceedings of the Workshop on Formal Methods and Security Protocols*, 1999.

[13] S. Even and O. Goldreich. On the security of multi-party ping-pong protocols. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:34–39, 1983.

[14] A. Huima. Efficient infinite-state analysis of security protocols. In *Proceedings of the FLOC'99 Workshop on Formal Methods and Security Protocols (FMSP'99)*, 1999.

[15] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Verifying Security Protocols. In M. Parigot and A. Voronkov, editors, *Proceedings of LPAR 2000*, LNCS 1955, pages 131–160. Springer-Verlag, 2000.

[16] Z. Liang and R. M. Verma. Correcting and Improving the NP Proof for Cryptographic Protocol Insecurity. In *Information Systems Security, 5th International Conference, ICISS 2009, Kolkata, India, December 14-18, 2009, Proceedings*, volume 5905 of *Lecture Notes in Computer Science*, pages 101–116. Springer, 2009.

[17] F. Martinelli and T. C. N. R. Analysis of security protocols as open systems. *Theoretical Computer Science*, 290:1057–1106, 2003.

[18] J. K. Millen and G. Denker. Capsl and mucapsl. *Journal of Telecommunications and Information Technology*, 4:16–27, 2002.

[19] J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *ACM Conference on Computer and Communications Security*, pages 166–175, 2001.

[20] M. Rusinowitch and M. Turuani. Protocol Insecurity with Finite Number of Sessions is NP-Complete. In *14th Computer Security Foundations Workshop (CSFW-14 2001), June 11-13, 2001, Cape Breton, Nova Scotia, Canada*, pages 174–. IEEE Computer Society, 2001.

[21] M. Turuani. *Sécurité des Protocoles Cryptographiques: Décidabilité et Complexité*. Phd thesis, Université Henri Poincaré, Nancy, december 2003.

[22] M. Turuani. The CL-Atse Protocol Analyser. In *Term Rewriting and Applications - Proc. of RTA*, volume 4098 of *Lecture Notes in Computer Science*, pages 277–286, Seattle, WA, USA, 2006.