History-Dependent Scheduling for Cryptographic Processes

Vincent Vanackère

Laboratoire d'Informatique Fondamentale de Marseille
Université de Provence,
39 rue Joliot-Curie, 13453 Marseille, France
vanackere@cmi.univ-mrs.fr

Abstract. This paper presents *history-dependent scheduling*, a new technique for reducing the search space in the verification of cryptographic protocols. This technique allows the detection of some "non-minimal" interleavings during a depth-first search, and was implemented in TRUST, our cryptographic protocol verifier. We give some experimental results showing that our method can greatly increase the efficiency of the verification procedure.

1 Introduction

In recent years, several symbolic reduction systems have been introduced, that aim at the analysis of security protocols. Some examples of this approach include [7, 1, 4]. These symbolic methods have an advantage over more traditional model-checking approaches in that they allow for the exploration and verification of an otherwise infinite-branching system, making it possible to perform an exact analysis for systems with a finite number of cryptographic processes.

However, the same problem remains as in most other model-checking techniques: as the number of parallel processes goes up, the number of possible interleavings makes the verification task harder – if not impossible in practice – because of the state-space explosion problem.

In this paper we present *history-dependent scheduling*, a new reduction technique that has been developed to be used in TRUST [10, 11], our cryptographic protocol verifier. This technique allows, in a depth-first search setting, the detection of some redundant – "non-minimal" – interleavings, and is also well-suited to symbolic transition systems, where few transitions actually commute and the usual reduction methods such as those of [6] do not apply.

The paper is organised as follows. After the presentation of our formal model, we will give an intuitive overview of our reduction procedure. This procedure will then be formally described and proved in the next two sections. The paper will end with some experimental results and concluding remarks.

2 Model

Our full model is presented in details in [2], and we will therefore only give a short version here.

We work under the usual Dolev-Yao model [5], where the network is under full control of an adversary that can analyse all messages exchanged and synthetize new ones. In our setting, messages can be viewed as terms in a free algebra – we work under the perfect encryption assumption – and we distinguish between basic names (agent's names, nonces, keys, ...) and composed messages (pairs $< \_, \_ >$ and encrypted terms $E(\_, \_)$), with the restriction that only basic names may be used as encryption keys. The set of names is denoted by $\mathcal{N}$ and the full set of messages by $\mathcal{M}$.

2.1 The Intruder's Knowledge

Given a set of terms T, we will denote the set of terms that the intruder may derive by $\mu(T)$. We assume a (computable) relation $\mathcal{D} \subseteq \mathcal{N} \times \mathcal{N}$ with the following interpretation:

$$(C, C') \in \mathcal{D} \text{ iff messages encrypted with } C \text{ can be decrypted with } C'.$$

We define $Inv(C) = \{C' \mid (C, C') \in \mathcal{D}\}$. Further hypotheses on the properties of $\mathcal{D}$ allow to model hashing, symmetric, and public keys. In particular: (i) for a *hashing* key C, $Inv(C) = \emptyset$, (ii) for a *symmetric* key C, $Inv(C) = \{C\}$, and (iii) for a *public* key C there is another key C' such that $Inv(C) = \{C'\}$ and $Inv(C') = \{C\}$.

Given a set of terms T we define the S (synthesis) and A (analysis) operators as follows:

- $S(T)$ is the least set that contains T and such that:

$$\begin{array}{ll} t_1, t_2 \in S(T) & \Rightarrow \langle t_1, t_2 \rangle \in S(T) \\ t_1 \in S(T), t_2 \in T \cap \mathcal{N} & \Rightarrow E(t_1, t_2) \in S(T) \ . \end{array}$$

- $A(T)$ is the least set that contains T and such that:

$$\begin{array}{ll} \langle t_1, t_2 \rangle \in A(T) & \Rightarrow t_i \in A(T),\ i = 1, 2 \\ E(t_1, t_2) \in A(T), A(T) \cap Inv(t_2) \neq \emptyset & \Rightarrow t_1 \in A(T) \ . \end{array}$$

As an example, if $T = \left\{E(\langle A, B\rangle, K), K^{-1}\right\}$ then $A(T) = T \cup \{A, B, \langle A, B\rangle\}$ and we have $E(A, K^{-1}) \in S(A(T))$.

With the above definitions, the knowledge that the intruder may derive from T is $\mu(T) = S(A(T))$. It should be noticed that the knowledge $\mu(T)$ is infinite whenever T is non-empty, and that it increases monotonically with T.

2.2 Processes and Configurations: Semantics

In our framework, a protocol is modelled as a finite number of processes interacting with an environment. Our process syntax includes the parallel composition – commutative and associative – of two processes, and thus we define a configuration k as a couple (P, T) where P is a process and T is an environment; the environment is a set of terms, composed of the initial knowledge augmented with all the messages emitted by the participants of the protocol so far. In the following, the adversary knowledge in a configuration $k \equiv (P, T)$ will be denoted by $\mu(k) = \mu(T)$.

Figure 1 gives the semantic rules as a reduction system on configurations. Informally, a process can either:

- (!) Write a message: the term is added to the environment knowledge.
- (?) Read some message from the environment: this can be any message the adversary is able to build from its current knowledge.
- (d) Decrypt some (encrypted) term with a corresponding inverse key.
- (pl) Perform some unpairing (the symmetric rule (pr) is not written).
- $(\mathsf{m}_1, \mathsf{m}_2)$ Test for the equality/inequality of two messages.
- (a) Check if some assertion φ holds in the current configuration.

$$
\begin{array}{lll}
(!) & (!t.P \mid P', T) & \rightarrow (P \mid P', T \cup \{t\}) \text{ if } t \in \mathcal{M} \\
(?) & (?x.P \mid P', T) & \rightarrow ([t/x]P \mid P', T) \text{ if } t \in \mu(T) \\
(\mathsf{d}) & (x \leftarrow \mathsf{dec}(E(t, C), C').P \mid P', T) & \rightarrow ([t/x]P \mid P', T) \text{ if } C' \in Inv(C), t \in \mathcal{M} \\
(\mathsf{pl}) & (x \leftarrow \mathsf{proj}_l(\langle t, t' \rangle).P \mid P', T) & \rightarrow ([t/x]P \mid P', T) \text{ if } t, t' \in \mathcal{M} \\
(\mathsf{a}) & (\mathsf{assert}(\varphi).P \mid P', T) & \rightarrow \begin{cases} (P \mid P', T) & \\ \mathsf{err} & \text{if } \not\models_T \varphi \end{cases} \\
(\mathsf{m}_1) & ([t = t]P_1, P_2 \mid P', T) & \rightarrow (P_1 \mid P', T) \text{ if } t \in \mathcal{M} \\
(\mathsf{m}_2) & ([t = t']P_1, P_2 \mid P', T) & \rightarrow (P_2 \mid P', T) \text{ if } t \neq t', t, t' \in \mathcal{M}
\end{array}
$$

Fig. 1. Reduction on configurations

Missing from the figure is the terminated process, denoted by 0, as well as the syntax of the assertion language, that will be presented in the next section. err denotes a special configuration that can only be reached from a false assertion.

In our model, *a correct protocol is a protocol that cannot reach the* err *configuration* – or, put in other words, a protocol such that all assertions reachable from the initial configuration of the system hold.

This reachability problem was shown to be NP-complete [2, 9].

2.3 The Assertion Language

The full assertion language we consider is the following:

$$\varphi ::= \mathit{true} \mid \mathit{false} \mid t = t' \mid t \neq t' \mid \mathsf{known}(t) \mid \mathsf{secret}(t) \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2$$

This is equivalent to saying that we consider arbitrary boolean combinations of atomic formulas checking the equality of two messages ($t = t'$) and the secrecy of a message ($\mathsf{secret}(t)$) with respect to the current knowledge of the adversary. As shown in [2], this language allows to easily express authentication properties such as aliveness and agreement [8]. An actual example of a specification using this assertion language is given in Appendix A.

The valuation of an assertion formula in an environment T is the "intuitive" one: for example, we have $\models_T \mathsf{secret}(t)$ iff $t \notin \mu(T)$. The only property on assertions that is used in this paper is the fact that *the truth value of an assertion in an environment T depends only on the knowledge $\mu(T)$.*

2.4 Symbolic Reduction and Challenges for Practical Verification

The main difficulty in the verification task is the fact that the input rule is infinitely branching as soon as the environment is not empty. In [1, 2] it was shown that it is possible to solve this problem by using a symbolic reduction system that stores the constraints in a symbolic shape during the execution. As an example, the input rule $(?x.P, T) \rightarrow ([t/x]P, T), t \in \mu(T)$ becomes $(?x.P, T, E) \rightarrow (P, T, (E\,;\, x : T))$. The complete description of the symbolic reduction system can be found in [2]. The main property we rely on is the fact that the symbolic reduction system is in lockstep with the ground one and provides a – sound and complete – decision procedure for processes specified using the full assertion language described in Sect. 2.3.

Although the symbolic reduction system is satisfying from a theoretical point of view, an inherent limitation is that it does not handle iterated processes (as the general case for iterated processes is undecidable). Thus, in order to verify a protocol against replay attacks and/or parallel sessions attacks, it is important to handle cases where there is a finite – even if small – number of participants playing each role.

Of course, as the number of parallel threads goes up, the number of possible interleavings make the verification task harder – if not impossible in practice – because we face the classical state explosion problem. Usual model-checking techniques to handle this problem involve the use of partial-order reduction [6], but unfortunately the symbolic reduction system doesn't lend itself very well to this approach, as different sequences of symbolic transitions will usually not lead to the same symbolic state.

We will tackle this problem by using a different approach: the basic idea is that by monitoring the inputs/outputs of the processes, we will be able to explore only some "complete" set of "interesting" interleavings. Due to the relative complexity of our reduction procedure, we will start by giving an intuitive overview in the next section.

3 A Scheduler for Cryptographic Processes

This section provides the intuition behind our reduction procedure. First, we recall the most important facts about our system:

– We consider a parallel composition of n processes.
– Processes can only interact through the environment.
– Each reduction step occurs only on one process, and its effect only depends on the environment knowledge.
– The environment knowledge can only increase along the reduction.

We will tackle our verification problem by using the following point of view:

– To each process of the parallel composition is associated a unique "nice level" (in the following, we will, without loss of generality, assume that the nice level of a process is its number in the parallel composition).
– The verification procedure is performed by a scheduler which is in charge, at each step of the reduction, of choosing the next process to be reduced, based on information on the past of the reduction. The scheduler may also choose to stop the exploration of the current branch.

This scheduler is similar to the one in a usual operating system, in the sense that it makes use of information on the past to make his decisions; the analogy stops here because our verification scheduler does not aim at achieving fairness or latency, but only at completeness. Moreover, it is also non-deterministic and has the possibility to sequentially explore several branches starting from the same configuration. Obviously, if we take as a scheduler the one exploring all possible branches at each step of the reduction, our description is just another name for a depth-first search (but, of course, we will try to be more clever...).

Our reduction procedure can now be (informally) illustrated as a scheduler that does the following:

Eager reduction: If the scheduler has selected some process for reduction, then this process may as well remain selected until it modifies the environment knowledge, else the reduction done on this process would not be able to affect the possible reductions of the other processes. From now on, we will call "eager step of reduction" any succession of reductions on the same process where the last step strictly increases the environment knowledge, and we will assume that the scheduler only performs eager steps of reduction.

Minimal traces: Whenever the scheduler stops reducing some process and then, later, decides to resume the reduction of this process, it will expect that the new eager step of reduction:

1. could not have happened at the time the reduction on this process was stopped ("no late work" clause);
2. could not have happened before the reduction of another process with a higher nice level ("priority" clause).

If any of those 2 conditions does not hold, the scheduler will simply stop the exploration of the current reduction.

We will now focus at formally defining and proving this reduction procedure. Section 4 will focus on the *eager reduction* procedure (this notion was already briefly described in [10]). Then, in Sect. 5 we will introduce a notion of *minimal traces* and give some criterions allowing the detection of non-minimal traces.

4 Eager Reduction

This section focuses on the description of the eager reduction procedure, that will be proven correct and complete w.r.t. the original reduction system.

In the following, we study the reduction of a configuration $(P_1 \mid \ldots \mid P_m, T)$, denoted by $(\Pi P_i, T)$. We will not allow the rewrite of $P \mid Q$ as $Q \mid P$, and therefore we can define the relation $\rightarrow_x$ as a reduction on the x-th process of the parallel composition. The variables $k, k', \ldots$ will be used for configurations: we recall that if $k \equiv (\Pi P_i, T)$, then $\mu(k)$ is defined as the environment knowledge in that configuration ($\mu(k) = \mu(T) = S(A(T))$), and if $k = \mathsf{err}$ we set $\mu(\mathsf{err}) = \emptyset$.

Reduction steps that do not modify the environment knowledge will play an important role in the following, and will be denoted as "silent":

Definition 4.1 (Silent reduction).
We will note $k \xrightarrow{\tau} k'$ *iff* $k \rightarrow k'$ *and* $\mu(k') = \mu(k)$.

The following commutation lemma for silent reductions, whose proof can be found in Appendix B, plays a crucial role in our reduction procedure:

Lemma 4.1 (Commutation lemma).
If $i \neq j$,
(1) $k \xrightarrow{\tau}_i \cdot \rightarrow_j \mathsf{err} \Rightarrow k \rightarrow_j \mathsf{err}$
(2) $k \xrightarrow{\tau}_i \cdot \rightarrow_j k' \neq \mathsf{err} \Rightarrow k \rightarrow_j \cdot \xrightarrow{\tau}_i k'$

A corollary of this lemma is the fact that if we consider a sequence of silent reductions $k \xrightarrow{\tau} {}^* k'$, then we can choose – in the path from k to k' – any order to reduce the processes of the parallel composition.

We will now annotate sequences of reductions to include the order in which the different processes modify the environment knowledge. In the following, we will write $\twoheadrightarrow$ as a shortcut for $\rightarrow^*$. In a similar way, we will write $\twoheadrightarrow_x \equiv (\rightarrow_x)^*$ for a sequence of reductions occurring on the process x.

Definition 4.2. *We write:*

$$
\begin{array}{lll}
(1)\ k \xrightarrow{\tau}\!\!\!\!\twoheadrightarrow k' & \textit{iff}\ \ k \twoheadrightarrow k'\ \textit{and}\ \mu(k') = \mu(k) \\
(2)\ k \xrightarrow{x}\!\!\!\!\twoheadrightarrow k' & \textit{iff}\ \ k \xrightarrow{\tau}\!\!\!\!\twoheadrightarrow \cdot \rightarrow_x \cdot \xrightarrow{\tau}\!\!\!\!\twoheadrightarrow k'\ \textit{and}\ \mu(k') \neq \mu(k) \\
(3)\ k \xrightarrow{x_1,\ldots,x_n}\!\!\!\!\twoheadrightarrow k' & \textit{iff}\ \ k \xrightarrow{x_1}\!\!\!\!\twoheadrightarrow \cdot \xrightarrow{x_2}\!\!\!\!\twoheadrightarrow \cdots \xrightarrow{x_n}\!\!\!\!\twoheadrightarrow k'
\end{array}
$$

The relation $\xrightarrow{\tau}\!\!\!\!\twoheadrightarrow$ stands for any sequence of silent reductions ; $\xrightarrow{x}\!\!\!\!\twoheadrightarrow$ means that the environment knowledge is modified (increased) exactly once, by the process x, during the reduction. Notation $\xrightarrow{x_1,\ldots,x_n}\!\!\!\!\twoheadrightarrow$ is only syntactic sugar: intuitively, the sequence $\{x_1, \ldots, x_n\}$ represents the order in which the processes bring new information to the environment.

Remark 4.1 (An output can hide behind another...).

If $k \twoheadrightarrow k'$ then there exists a sequence $\{x_1, \ldots, x_n\}$ such that $k \xrightarrow{x_1,\ldots,x_n}\!\!\!\!\twoheadrightarrow k'$. It should be noted that the set of sequences s satisfying $k \xrightarrow{s}\!\!\!\!\twoheadrightarrow k'$ is not related to

Mazurkiewicz traces[1]; as an example, if $k = (!A.P_1 \mid !\langle A,B\rangle.P_2,\ \{B\})$ and $k' = (P_1 \mid P_2,\ \{A, B, \langle A,B\rangle\})$, both the sequences $\{1\}$ and $\{2\}$ satisfy $k \overset{s}{\twoheadrightarrow} k'$ (after the output from either one of the processes, an output from the other one will not bring any new information to the environment).

Definition 4.3 (Eager reduction).

We define $\hookrightarrow_x$, a step of eager reduction on the process x, as:

$$k \hookrightarrow_x k' \text{ iff } k \overset{\tau}{\twoheadrightarrow}_x \cdot \rightarrow_x k' \text{ and } \mu(k') \neq \mu(k) \ .$$

The eager reduction relation $\hookrightarrow$ is then defined by $\hookrightarrow = \bigcup_x \hookrightarrow_x$.

By extension, we will write $k \hookrightarrow_{x_1,\ldots,x_n} k'$ whenever $k \hookrightarrow_{x_1} \cdot \hookrightarrow_{x_2} \cdots \hookrightarrow_{x_n} k'$.

Thus, during a step of eager reduction, we reduce only some process x of the configuration until it increases the environment knowledge or reaches error: this formal definition is the one corresponding to rule (1) of our scheduler.

By using Lemma 4.1, we can establish the following theorem on eager reduction (the proof can be found in Appendix C):

Theorem 4.1.

1. $k \xrightarrow{x_1,\ldots,x_n}\!\!\!\!\twoheadrightarrow k' \neq \mathsf{err} \Rightarrow k \hookrightarrow_{x_1,\ldots,x_n} \cdot \overset{\tau}{\twoheadrightarrow} k'$
2. $k \xrightarrow{x_1,\ldots,x_n}\!\!\!\!\twoheadrightarrow \mathsf{err} \Rightarrow k \hookrightarrow_{x_1,\ldots,x_n} \mathsf{err}$

We can now state the soundness and completeness of the eager reduction:

Theorem 4.2.

$$k \rightarrow^* \mathsf{err} \text{ iff } k \hookrightarrow^* \mathsf{err}$$

Proof. Completeness follows from Theorem 4.1(2): if $k \twoheadrightarrow \mathsf{err}$, then there exists $\{x_1, \ldots, x_n\}$ such that $k \xrightarrow{x_1,\ldots,x_n}\!\!\!\!\twoheadrightarrow \mathsf{err}$, and thus $k \hookrightarrow_{x_1,\ldots,x_n} \mathsf{err}$, which means $k \hookrightarrow^* \mathsf{err}$. Soundness comes trivially from $\hookrightarrow \subseteq \rightarrow^*$. □

5 Characteristics and Minimal Traces

In this section we will show how it is possible to avoid the full exploration of all eager traces by exploiting some information on the past of a trace. It is namely possible, by monitoring the values taken in input, to detect, at the end of its execution, that some step of reduction could have occurred earlier in the trace; by using a suitable order on traces (Definition 5.2), we will show that we can cut this branch of the search space whenever we detect that the current trace cannot be a minimal one.

In all this section, we consider an initial configuration k such that $k \rightarrow^* \mathsf{err}$ and we will show the existence of a reduction sequence from k to err verifying some particular properties.

[1] We recall that two words/traces are Mazurkiewicz-equivalent iff they can be obtained by permutations of independent letters/transitions.

5.1 Definitions

Definition 5.1 (Traces and characteristics).
An eager trace *from k to k' is a sequence $k \hookrightarrow k_1 \hookrightarrow \cdots \hookrightarrow k'$.*
We will denote $k \hookrightarrow_{p^m} k'$ whenever $k \hookrightarrow_p \cdots \hookrightarrow_p k'$ with m eager steps.
Any eager trace can be uniquely written as:

$$k \hookrightarrow_{p_1{}^{m_1},\ldots,p_n{}^{m_n}} k' \text{ with } \forall i \;\; p_i \neq p_{i+1} \text{ and } m_i > 0 \;\; .$$

The characteristic of this trace is then defined by the tuple $(p_1{}^{m_1}, \ldots, p_n{}^{m_n})$.

In the following, we will often write "trace" as a shortcut for "eager trace". We now introduce a way to compare the traces characteristics:

Definition 5.2 (Partial order on characteristics).
The relation $\prec$ is defined as:

$$(p_1{}^{m_1}, \ldots, p_n{}^{m_n}) \prec (p'_1{}^{m'_1}, \ldots, p'_{n'}{}^{m'_{n'}})$$

$$\textit{iff} \quad \exists i \in \{1, \ldots, \min(n, n')\} \begin{cases} \forall j < i \;\; (p_j, m_j) = (p'_j, m'_j) \\ p_i < p'_i \vee (p_i = p'_i \wedge m_i > m'_i) \end{cases}$$

Example 5.1. If $a < b < c$, then $(a^3, b^1, c^2) \prec (a^3, c^2) \prec (a^3, c^1, b^7)$.

The introduction of this particular relation could appear counter-intuitive, due to condition $m_i > m'_i \ldots$ The informal explanation is the following: the purpose of this order is to set as minimal the traces for which *stopping the reduction on some process implies that the next reduction on this process must necessarily depend on what happened "in-between"*. Then, if $k \hookrightarrow_a \cdot \hookrightarrow_a \cdot \hookrightarrow_b k'$ and $k \hookrightarrow_a \cdot \hookrightarrow_b \cdot \hookrightarrow_a k'$, the first sequence will have a smaller characteristic than the second one (and should ideally be favoured by our scheduler).

It should be noted that our relation makes use of an arbitrary order on the processes – their number – that corresponds in fact to the "nice levels" of our informal scheduler description.

Lemma 5.1. *$\prec$ is a partial order.*

Remark 5.1. In fact, two characteristics are always comparable unless one is the prefix of the other.

The following lemma will allow us to establish the existence of minimal traces leading to error:

Lemma 5.2. *The set of all characteristics associated to some non-empty set of traces from the same initial configuration admits some minimal elements.*

Proof. Although $\prec$ is not well founded this property simply holds because there is only a finite number of possible characteristics for all the traces starting from a given initial configuration k (namely if W is the total number of output instructions contained in k, any eager trace will be at most of length $W + 1$). □

Definition 5.3 (Minimal traces).
We consider the set of all characteristics associated to all traces from k to err*: this set admits minimal elements, and any trace from k to* err *whose characteristic is minimal will be called minimal.*

As a consequence, finding criteria for non-minimal traces will allow us to avoid the full exploration of all traces by the scheduler.

5.2 Criteria for Non-Minimal Traces

In this section, we will develop a characterisation of some non-minimal traces, based on the observation of the values taken by the input variables of the processes. This will require a new notation:

Definition 5.4. *Let X be a set of terms. We denote:*

$$(?x.P, T) \overset{?_X}{\to} ([t/x]P, T) \quad \textit{iff} \quad t \in S(A(T)) \cap X$$

By extension, we will also denote $k \overset{?_X}{\hookrightarrow}_p k'$ when all the input values in the reduction from k to k' are included in the set X.

The following – crucial – lemma is proven in Appendix D:

Lemma 5.3 (Eager commutation).

If $i \neq j$, $X \subseteq \mu(k)$ and $k \hookrightarrow_i \cdot \overset{?_X}{\hookrightarrow}_j \cdot \hookrightarrow^+$ err, then $k \overset{?_X}{\hookrightarrow}_j \cdot \hookrightarrow^+$ err.

We can now state our fundamental theorem, that gives two sufficient conditions for the non-minimality of a trace:

Theorem 5.1 (Non-minimality). *Any trace $k_1 \hookrightarrow_{{p_1}^{m_1}} \cdots \hookrightarrow_{{p_n}^{m_n}}$ err containing a sub-trace of one of the following forms is not minimal (we denote $X_i \equiv \mu(k_i)$):*

$$\begin{array}{l}
(1)\ k_i \hookrightarrow_{{p_i}^{m_i}} k_{i+1} \cdots \hookrightarrow_{{p_j}^{m_j}} \cdot \overset{?_{X_{i+1}}}{\hookrightarrow}_p k' \neq \mathsf{err} \\
\quad p = p_i,\ p \notin \{p_{i+1}, \ldots, p_j\} \\
(2)\ k_i \hookrightarrow_{{p_i}^{m_i}} \cdots \hookrightarrow_{{p_j}^{m_j}} \cdot \overset{?_{X_i}}{\hookrightarrow}_p k' \neq \mathsf{err} \\
\quad p < p_i,\ p \notin \{p_i, \ldots, p_j\}
\end{array}$$

Proof.

(1) By iterating Lemma 5.3, we show $k_i \hookrightarrow_{{p_i}^{m_i+1}} \cdot \hookrightarrow^+$ err. Therefore there exists some trace of characteristic $({p_1}^{m_1}, \ldots, {p_i}^{m_i+q}, \ldots)$ leading to error, with $q \geq 1$. However, by definition of $\prec$ we have $({p_1}^{m_1}, \ldots, {p_i}^{m_i+q}, \ldots) \prec ({p_1}^{m_1}, \ldots, {p_i}^{m_i}, \ldots)$, thus the non-minimality of our initial trace.

(2) By iterating Lemma 5.3, we get $k_i \hookrightarrow_p \cdot \hookrightarrow^+$ err. If $i > 1$ and $p = p_{i-1}$, we are back to the previous case, else there exists some trace with characteristic $({p_1}^{m_1}, \ldots, {p_{i-1}}^{m_{i-1}}, p^q, \ldots)$ leading to error, with $q \geq 1$. As $p < p_i$, we have $({p_1}^{m_1}, \ldots, p^q, \ldots) \prec ({p_1}^{m_1}, \ldots, {p_i}^{m_i}, \ldots)$, and our initial trace is not minimal.

5.3 Configurations with History

Theorem 5.1 shows that by having the relevant information on the past of the current trace, our scheduler could easily detect – and discard – non-minimal traces. We will therefore enrich configurations with a third component, the *history*, which will be a list of triple; for each reduction sequence on one process, we will indeed record the process number, the current environment just at the beginning of the reduction on this process, and the list of all the values that were read during the sequence. Formally, we write: $\mathcal{H} = (p_0, T_0, X_0) : \cdots : (p_n, T_n, X_n)$, where each p_i is a process number, T_i an environment and X_i a set of terms.

We define a function append_H for updating a history as follows:

Definition 5.5. *If $\mathcal{H} = (p_0, T_0, X_0) : \cdots : (p_n, T_n, X_n)$, then:*

- *if $p = p_n$, $\mathsf{append}_H(\mathcal{H}, p, T, X) = (p_0, T_0, X_0) : \cdots : (p_n, T_n, X_n \cup X)$*
- *if $p \neq p_n$, $\mathsf{append}_H(\mathcal{H}, p, T, X) = (p_0, T_0, X_0) : \cdots : (p_n, T_n, X_n) : (p, T, X)$.*

Reduction rules are then modified to integrate a history:

- the input rule becomes:

$$(?x.P \mid P', T, \mathcal{H}) \rightarrow_i ([t/x]P \mid P', T, \mathcal{H}') \text{ if } \begin{cases} t \in S(A(T)) \\ \mathcal{H}' = \mathsf{append}_H(\mathcal{H}, i, T, \{t\}) \end{cases}$$

- for all the other rules, if $(P, T) \rightarrow_i (P', T')$ then:

$$(P, T, \mathcal{H}) \rightarrow_i (P', T', \mathcal{H}') \text{ if } \mathcal{H}' = \mathsf{append}_H(\mathcal{H}, i, T, \emptyset) \ .$$

Put in other words, each time the scheduler will choose a new process for reduction, it will start a new section of the history and record the current environment; this section will then be updated with all the values that are read by this process during its execution/reduction.

We can now state our final theorem, giving the two conditions that will be checked, after each eager step, by the scheduler in order to discard non-minimal traces:

Theorem 5.2 (Non-minimal history). *We consider a trace $k \hookrightarrow^+ k' \neq \mathsf{err}$ such that the history $\mathcal{H}(k') = (p_0, T_0, X_0) : \cdots : (p_n, T_n, X_n)$ satisfies one of the following properties:*

$$(1)\ \exists i < n.\ \begin{cases} p_i = p_n \\ p_n \notin \{p_{i+1}, \ldots, p_{n-1}\} \\ X_n \subseteq \mu(T_{i+1}) \end{cases}$$

$$(2)\ \exists i < n.\ \begin{cases} p_i > p_n \\ p_n \notin \{p_{i+1}, \ldots, p_{n-1}\} \\ X_n \subseteq \mu(T_i) \end{cases}$$

Then this trace cannot be the prefix of a minimal one.

Proof. $\mathcal{H}(k')$ verifies either (1) or (2) and therefore the eager reduction to k' will verify the corresponding condition from Theorem 5.1. As a consequence, any trace having the one to k' as a prefix cannot be a minimal one.

6 Experimental Results

We have implemented our history-based reduction procedure in our verifier, TRUST. Figure 2 gives some times for the full analysis of some typical protocols: the measures were taken on an Athlon XP 1800, and all times are in seconds (the time spent by the verifier is roughly proportional to the number of basic reduction steps that are done). For each protocol, we detail the number of roles involved and give the times to do a full search depending on the number of parallel – interleaved – sessions[2]. T_0 is the time when the usual reduction method is applied (*i.e.* the scheduler reduces each process until it emits an output); T_{Eager} is the time for the eager reduction procedure alone; T_{Min} is the time without eager reduction, but using the history to detect non-minimal traces; then $T_{\text{Eager+min}}$ is the time for the full reduction procedure presented in this paper. As can be seen in the figure, our reduction method reveals itself quite effective in practice: sometimes a reduction factor of 60 is gained, and we did not encounter any case where the added checks (for non-minimality) made history-dependent scheduling slower.

Protocol	# roles	# sessions	T_0	T_{Eager}	T_{Min}	$T_{\text{Eager+Min}}$
Needham-Schroeder-Lowe	2	3	50	6.80	1.18	0.59
	2	4	?	2034	99	41
Needham-Schroeder-Lowe[3]	3	2	277	2.38	1.04	0.16
	3	3	?	963	163	8.46
Otway-Rees	3	2	0.75	0.32	0.28	0.14
	3	3	5879	722	497	98
Carlsen	3	2	6.15	4.79	1.29	0.91
	3	3	?	?	?	2272
Kerberos v5	4	2	5.40	4.11	2.31	1.93

Fig. 2. Times for the analysis of various protocols (in seconds).

7 Conclusion

In this paper we have presented history-dependent scheduling, a new reduction method for cryptographic protocols that is based on the monitoring of the inputs/outputs performed by the processes during the reduction. This method relies on the two following techniques:

[2] A question mark instead of the time means that we were not "patient" enough to wait for the end of the verification procedure.

[3] This is the seven-message version of the protocol, making use of 3 key servers in parallel.

– Eager reduction, that was already presented in [10], is a natural big-step semantics based on the outputs of the different processes.
– Detection of non-minimal traces, through the use of a history of the current reduction sequence, allows to stop the exploration of some traces and is based on very simple criteria on the inputs of the processes.

As far as related work is concerned, our technique seems to share some common grounds with the one developed independently in [3], in that both methods somehow aim to reduce the search space by verifying/maintaining additional constraints on the input values; however, differences between the formalisms make the comparison a non-trivial task, that we have to leave as a future work.

It should be noted that we have only defined and proved here our method on the ground reduction system; this method can be adapted in a straightforward manner to the symbolic system, and the same non-minimality conditions are then checked at the symbolic level (although we should mention that the proof of completeness for the symbolic case becomes more complex, due to some slight differences between the symbolic eager reduction and the ground one).

As practical experiments show, history-dependent scheduling can be quite effective in practice – and in all our tests never induces any slowdown. We expect that the ideas behind this method are general enough to be applied to other verification systems, and also stress the fact that this technique is, by its nature, especially well suited for verification in a depth-first setting.

References

1. R. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In *Proc. CONCUR00, Springer LNCS 1877*, 2000. Also RR-INRIA 3915.
2. R. Amadio, D. Lugiez and V. Vanackère. On the symbolic reduction of processes with cryptographic functions. RR-INRIA 4147, March 2001. Revised version in *Theoretical Computer Science*, 290(1);695-740, 2003.
3. D. Basin, S. Mödersheim and L. Viganò. Constraint Differentiation: A New Reduction Technique for Constraint-Based Analysis of Security Protocols. Technical Report 405, Dep. of Computer Science, ETH Zurich, 2003.
4. M. Boreale. Symbolic Trace Analysis of Cryptographic Protocols. In *Proc. ICALP01, Springer LNCS, Berlin*, 2001.
5. D. Dolev and A. Yao. On the security of public key protocols. *IEEE Trans. on Information Theory*, 29(2):198–208, 1983.
6. P. Godefroid. Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem. In *Springer LNCS 1032*, 1996.
7. A. Huima. Efficient infinite-state analysis of security protocols. In *Proc. Formal methods and security protocols, FLOC Workshop, Trento*, 1999.
8. G. Lowe. A hierarchy of authentication specifications. In *Proc. 10th IEEE Computer Security Foundations Workshop*, 1997.
9. M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. RR INRIA 4134, March 2001.
10. V. Vanackère. The TRUST protocol analyser, automatic and efficient verification of cryptographic protocols. In *VERIFY'02 Workshop, Copenhagen*, 2002.
11. http://www.cmi.univ-mrs.fr/∼vvanacke/trust/

A Specifying Security Properties through Assertions

We take as an example the following 3 message version of the Needham-Schroeder Public Key protocol:

$$A \to B : \{na, A\}_{Pub(B)}$$
$$B \to A : \{na, nb\}_{Pub(A)}$$
$$A \to B : \{nb\}_{Pub(B)}$$

In our framework, the protocol can be modeled as follows:

$$\begin{aligned} Init(myid, resp) : & \text{ fresh } na. \\ & !E(\langle na, myid\rangle, Pub(resp)). \\ & ?e.\ \langle na', nb\rangle \leftarrow \mathsf{dec}(e, Priv(myid)).\ [na' = na]. \\ & !E(nb, Pub(resp)). \\ & \mathsf{assert}(\mathsf{secret}(nb) \wedge \mathsf{auth}(resp, myid, na, nb)).\ 0 \end{aligned}$$

$$\begin{aligned} Resp(myid, init) : & ?e.\ \langle na, a\rangle \leftarrow \mathsf{dec}(e, Priv(myid)).\ [a = init]. \\ & \text{fresh } nb. \\ & !_{\mathsf{auth}(myid,init,na,nb)}\ E(\langle na, nb\rangle, Pub(init)). \\ & ?e'.\ [e' = E(nb, Pub(myid))]. \\ & 0 \end{aligned}$$

The assertion "auth(msg)" is a shortcut for "$\mathsf{known}(E(msg, K_{\mathsf{auth}}))$", and instruction "$!_{\mathsf{auth}(msg)}t$" is some syntactic sugar for "$!\langle E(msg, K_{\mathsf{auth}}), t\rangle$". In this example, the initiator specifies that at the end of its run of the protocol the nonce nb must be secret, and expects an agreement with some responder on the nonces na and nb.

B Proof of Lemma 4.1

(1) We have $k \xrightarrow{\tau}_i k' \to_j \mathsf{err}$. The reduction on j is a false assertion, and as we have $\mu(k) = \mu(k')$ and as the truth value of an assertion only depends on the environment knowledge, the same assertion will also evaluate to false in the configuration k.

(2) We have $k_1 \xrightarrow{\mathsf{r}_1}_i k_2 \xrightarrow{\mathsf{r}_2}_j k_3$ ($k_1 = k$ and $k_3 = k'$). The proof is done by basic case analysis on the rules r_1 and r_2. . All rules but (?), (a) and (!) do not depend at all from the environment nor modify, it and thus the result holds whenever $\to_i \notin \{(\mathsf{a}), (?), (!)\}$ or $\to_j \notin \{(\mathsf{a}), (?), (!)\}$. Commutation when $\mathsf{r}_1 = (\mathsf{a})$ or $\mathsf{r}_2 = (\mathsf{a})$ is always possible, given the fact that a process may choose not to evaluate an assertion. There remains only 4 cases:

1. $k_1 \xrightarrow{?}_i k_2 \xrightarrow{?}_j k_3$
2. $k_1 \xrightarrow{!}_i k_2 \xrightarrow{!}_j k_3$
3. $k_1 \xrightarrow{?}_i k_2 \xrightarrow{!}_j k_3$
4. $k_1 \xrightarrow{!}_i k_2 \xrightarrow{?}_j k_3$

Cases (1), (2) and (3) are straightforward[4]. Case (4) is the interesting one (and the one where our eager reduction procedure will take advantage): namely we can perform the input rule first and then reach k_3 after an output from the process number i, due to the fact that the input rule only depends on the knowledge $\mu(k_2)$, which is the same as $\mu(k_1)$ since $k_1 \xrightarrow{\tau}_i k_2$. □

C Proof of Theorem 4.1

We simultaneously prove (1) and (2) by induction on n.

Case $(n = 1)$**.** We have to establish:

(1) $k \overset{x}{\twoheadrightarrow} k' \neq \mathsf{err} \Rightarrow k \hookrightarrow_x \cdot \overset{\tau}{\twoheadrightarrow} k'$

(2) $k \overset{x}{\twoheadrightarrow} \mathsf{err} \Rightarrow k \hookrightarrow_x \mathsf{err}$

Both properties are easily shown by iterating Lemma 4.1 in order to move all reductions on the process x at the beginning of the reduction sequence.

Case $(n > 1)$. By induction:

(1) $k \overset{x_1,\ldots,x_n}{\twoheadrightarrow} k'$ implies $k \overset{x_1,\ldots,x_{n-1}}{\twoheadrightarrow} k'' \overset{x_n}{\twoheadrightarrow} k'$ and by induction hypothesis we know that:

$$\exists k_{n-1}.\ k \hookrightarrow_{x_1,\ldots,x_{n-1}} k_{n-1} \overset{\tau}{\twoheadrightarrow} k'' \ .$$

Then $k_{n-1} \overset{\tau}{\twoheadrightarrow} k'' \overset{x_n}{\twoheadrightarrow} k'$, which means $k_{n-1} \overset{x_n}{\twoheadrightarrow} k'$, and by the result for case $n = 1$ we can deduce $\exists k_n.\ k_{n-1} \hookrightarrow_{x_n} k_n \overset{\tau}{\twoheadrightarrow} k'$.

(2) We have $k \overset{x_1,\ldots,x_{n-1}}{\twoheadrightarrow} k' \overset{x_n}{\twoheadrightarrow} \mathsf{err}$ and by induction hypothesis:

$$\exists k''.\ k \hookrightarrow_{x_1,\ldots,x_{n-1}} k'' \overset{\tau}{\twoheadrightarrow} k' \overset{x_n}{\twoheadrightarrow} \mathsf{err} \ .$$

Then $k'' \overset{x_n}{\twoheadrightarrow} \mathsf{err}$ and thus, by the result for case $n = 1$, $k'' \hookrightarrow_{x_n} \mathsf{err}$, which ends the proof. □

D Proof of Lemma 5.3

First, we will extend Lemma 4.1 with the two following properties $(i \neq j)$:

(1) If $k \rightarrow_i \cdot \overset{?_X}{\rightarrow}_j k'$ and $X \subseteq \mu(k)$ then $k \overset{?_X}{\rightarrow}_j \cdot \rightarrow_i k'$

(2) If $k \rightarrow_i k_1 \overset{!}{\rightarrow}_j k'$ and $\mu(k_1) \neq \mu(k')$ then $\exists k_2.\, k \overset{!}{\rightarrow}_j k_2 \rightarrow_i k' \wedge \mu(k) \neq \mu(k_2)$

By iterated application of (1), (2) and Lemma 4.1, we can now move the reduction steps on the process j at the beginning of the sequence and get:

$$\exists (k', k'').\ \ k \overset{?_X}{\longrightarrow\!\!\!\!\!\rightarrow}_j k' \overset{!}{\rightarrow}_j k'' \rightarrow\!\!\!\!\!\rightarrow_i \cdot \hookrightarrow^+ \mathsf{err} \wedge \mu(k') \neq \mu(k'') \ .$$

As $\mu(k') \neq \mu(k'')$, $k \overset{?_X}{\longrightarrow\!\!\!\!\!\rightarrow}_j k' \overset{!}{\rightarrow}_j k''$ implies $\exists q \geq 1.\ k \overset{?_X}{\hookrightarrow}_{j^q} k''$. On the other hand, we know by the completeness of eager reduction that $k'' \hookrightarrow^+ \mathsf{err}$. Therefore $k \overset{?_X}{\hookrightarrow}_{j^q} \cdot \hookrightarrow^+ \mathsf{err}$, which implies $k \overset{?_X}{\hookrightarrow}_j \cdot \hookrightarrow^+ \mathsf{err}$. □

[4] It should be noted that (3) is "folklore" and used very broadly in the literature; we can summarize it as: "if we have in a parallel one process doing an input and another one an output, the output can always be done first without any loss".