Automatic SAT-Compilation of Protocol Insecurity Problems via Reduction to Planning*

Alessandro Armando[1] and Luca Compagna[1]

[1]DIST – Università degli Studi di Genova, Viale Causa 13 – 16145 Genova, Italy,
{armando,compa}@dist.unige.it

Abstract. We provide a fully automatic translation from security protocol specifications into propositional logic which can be effectively used to find attacks to protocols. Our approach results from the combination of a reduction of protocol insecurity problems to planning problems and well-known SAT-reduction techniques developed for planning. We also propose and discuss a set of transformations on protocol insecurity problems whose application has a dramatic effect on the size of the propositional encoding obtained with our SAT-compilation technique. We describe a model-checker for security protocols based on our ideas and show that attacks to a set of well-known authentication protocols are quickly found by state-of-the-art SAT solvers.

Keywords. Network security; Verification.

1 Introduction

Even under the assumption of perfect cryptography, the design of security protocols is notoriously error-prone. As a consequence, a variety of different protocol analysis techniques has been put forward [3, 4, 8, 10, 12, 16, 19, 21, 22]. In this paper we address the problem of translating protocol insecurity problems into propositional logic in a fully automatic way with the ultimate goal to build an automatic model-checker for security protocols based on state-of-the-art SAT solvers. Our approach combines a reduction of protocol insecurity problems to planning problems[1] with well-known SAT-reduction techniques developed for planning. At the core of our technique is a set of transformations whose application to the input protocol insecurity problem has a dramatic effect on the size of the propositional formulae obtained. We present a model-checker for security protocols based on our ideas and show that—using our tool—attacks to a set of well-known authentication protocols are quickly found by state-of-the-art SAT solvers.

* This work has been supported by the Information Society Technologies Programme, FET Open Assessment Project "AVISS" (Automated Verification of Infinite State Systems), IST-2000-26410.

[1] The idea of regarding security protocol analysis as a planning problem is not new. To our knowledge it is also been proposed in [1].

2 Security Protocols and Protocol Insecurity Problems

In this paper we concentrate our attention on error detection of authentication protocols (see [7] for a survey). As a simple example consider the following one-way authentication protocol:

$$(1) \quad A \rightarrow B : \{N_a\}_{K_{ab}}$$
$$(2) \quad B \rightarrow A : \{f(N_a)\}_{K_{ab}}$$

where N_a is a nonce[2] generated by Alice, K_{ab} is a symmetric key, f is a function known to Alice and Bob, and $\{x\}_k$ denotes the result of encrypting text x with key k. Successful execution of the protocol should convince Alice that she has been talking with Bob, since only Bob could have formed the appropriate response to the message issued in (*1*). In fact, Ivory can deceit Alice into believing that she is talking with Bob whereas she is talking with her. This is achieved by executing concurrently two sessions of the protocol and using messages from one session to form messages in the other as illustrated by the following protocol trace:

$$(1.1) \quad A \rightarrow I(B) : \{N_a\}_{K_{ab}}$$
$$(2.1) \quad I(B) \rightarrow A : \{N_a\}_{K_{ab}}$$
$$(2.2) \quad A \rightarrow I(B) : \{f(N_a)\}_{K_{ab}}$$
$$(1.2) \quad I(B) \rightarrow A : \{f(N_a)\}_{K_{ab}}$$

Alice starts the protocol with message (*1.1*). Ivory intercepts the message and (pretending to be Bob) starts a second session with Alice by replaying the received message—cf. step (*2.1*). Alice replies to this message with message (*2.2*). But this is exactly the message Alice is waiting to receive in the first protocol session. This allows Ivory to finish the first session by using it—cf. (*1.2*). At the end of the above steps Alice believes she has been talking with Bob, but this is obviously not the case.

A problem with the above rule-based notation to specify security protocols is that it leaves implicit many important details such as the shared information and how the principals should react to messages of an unexpected form. This kind of description is therefore usually supplemented with explanations in natural language which in our case explain that N_a is a nonce generated by Alice, that f is a function known to the honest participants, and that K_{ab} is a shared key.

To cope with the above difficulties and pave the way to the formal analysis of security protocols a set of models and specification formalisms as well as translators from high-level languages (similar to the one we used above to introduce our example) into these formalisms have been put forward. For instance, Casper [18] compiles high-level specifications into CSP, whereas CAPSL [5] and the AVISS tool [2] compile high-level specifications into formalisms based on multiset rewriting inspired by [6].

[2] *Nonces* are numbers generated by principals that are intended to be used *only once*.

2.1 The Model

We model the concurrent execution of a protocol by means of a state transition system. Following [16], we represent states by sets of atomic formulae called *facts* and transitions by means of rewrite rules over sets of facts. For the simple protocol above, facts are built out of a first-order sorted signature with sorts `user`, `number`, `key`, `func`, `text` (super-sort of all the previous sorts), `int`, `session`, `nonceid`, and `list_of text`. The constants 0, 1, and 2 (of sort `int`) denote protocols steps, $\underline{1}$ and $\underline{2}$ (of sort `session`) denote session instances, a and b (of sort `user`) denote honest participants, k_{ab} (of sort `key`) denotes a symmetric key and na (of sort `nonceid`) is a nonce identifier. The function symbol $\{\_\}\_ : \texttt{text} \times \texttt{key} \to \texttt{text}$ denotes the encryption function, $f : \texttt{number} \to \texttt{func}$ denotes the function known to the honest participants, $nc : \texttt{nonceid} \times \texttt{session} \to \texttt{number}$, and $s : \texttt{session} \to \texttt{session}$ are nonce and session constructors respectively. The predicate symbols are i of arity `text`, `fresh` of arity `number`, m of arity $\texttt{int} \times \texttt{user} \times \texttt{user} \times \texttt{text}$, and w of arity $\texttt{int} \times \texttt{user} \times \texttt{user} \times \texttt{list\_of text} \times \texttt{list\_of text} \times \texttt{session}$:

- $i(t)$ means that the intruder knows t.
- $\mathit{fresh}(n)$ means that n has not been used yet.
- $m(j, s, r, t)$ means that principal s has (supposedly)[3] sent message t to principal r at protocol step j.
- $w(j, s, r, ak, ik, c)$ represents the state of execution of principal r at step j of session c; in particular it means that r knows the terms stored in the lists ak (*acquired knowledge*) and ik (*initial knowledge*) at step j of session c, and—if $j \neq 0$—also that a message from s to r is awaited for step j to be executed.

Initial States. The initial state of the system is:[4]

$$w(0, a, a, [\,], [a, b, k_{ab}], \underline{1}) \,.\, w(1, a, b, [\,], [b, a, k_{ab}], \underline{1}) \tag{1}$$

$$.\, w(0, b, b, [\,], [b, a, k_{ab}], \underline{2}) \,.\, w(1, b, a, [\,], [a, b, k_{ab}], \underline{2}) \tag{2}$$

$$.\, \mathit{fresh}(nc(na, \underline{1})) \,.\, \mathit{fresh}(nc(na, s(\underline{1}))) \tag{3}$$

$$.\, \mathit{fresh}(nc(na, \underline{2})) \,.\, \mathit{fresh}(nc(na, s(\underline{2}))) \tag{4}$$

$$.\, i(a) \,.\, i(b) \tag{5}$$

Facts (1) represent the initial state of principals a and b (as initiator and responder, resp.) in session $\underline{1}$. Dually, facts (2) represent the initial state of principals b and a (as responder and initiator, resp.) in session $\underline{2}$. Facts (3) and (4) state the initial freshness of the nonces. Facts (5) represent the information initially known by the intruder.

Rewrite rules over sets of facts are used to specify the transition system evolves.

[3] As we will see, since the intruder may fake other principals' identity, the message might have been sent by the intruder.

[4] To improve readability we use the "." operator as set constructor. For instance, we write "$x \,.\, y \,.\, z$" to denote the set $\{x, y, z\}$.

Protocol Rules. The following rewrite rule models the activity of sending the first message:

$$w(0, A, A, [\,], [A, B, K_{ab}], C) \,.\, fresh(nc(na, C)) \xrightarrow{step_1(A,B,C,K_{ab})} m(1, A, B, \{nc(na, C)\}_{K_{ab}}) \,.\, w(2, B, A, [nc(na, C)], [A, B, K_{ab}], C) \quad (6)$$

Notice that nonce $nc(na, C)$ is added to the acquired knowledge of A for subsequent use. The receipt of the message and the reply of the responder is modeled by:

$$m(1, A, B, \{nc(ID, C1)\}_{K_{ab}}) \,.\, w(1, A, B, [\,], [B, A, K_{ab}], C) \xrightarrow{step_2(A,B,C,C1,K_{ab},ID)} m(2, B, A, \{f(nc(ID, C1))\}_{K_{ab}}) \,.\, w(1, A, B, [\,], [B, A, K_{ab}], s(C)) \quad (7)$$

The final step of the protocol is modeled by:

$$m(2, B, A, \{f(nc(ID, C1))\}_{K_{ab}}) \,.\, w(2, B, A, [nc(ID, C1)], [A, B, K_{ab}], C) \xrightarrow{step_3(A,B,C,C1,K_{ab},ID)} \,.\, w(0, A, A, [\,], [A, B, K_{ab}], s(C)) \quad (8)$$

Intruder Rules. There are also rules specifying the behavior of the intruder. In particular the intruder is based on the model of Dolev and Yao [11]. For instance, the following rule models the ability of the intruder of diverting the information exchanged by the honest participants:

$$m(J, S, R, T) \xrightarrow{divert(J,R,S,T)} i(S) \,.\, i(R) \,.\, i(T) \quad (9)$$

The ability of encrypting and decrypting messages is modeled by:

$$i(T) \,.\, i(K) \xrightarrow{encrypt(K,T)} i(T) \,.\, i(K) \,.\, i(\{T\}_K) \quad (10)$$

$$i(\{T\}_K) \,.\, i(K) \xrightarrow{decrypt(K,T)} i(K) \,.\, i(T) \quad (11)$$

Finally, the intruder can send arbitrary messages possibly faking somebody else's identity in doing so:

$$i(T) \,.\, i(S) \,.\, i(R) \xrightarrow{fake_1(R,S,T)} i(T) \,.\, i(S) \,.\, i(R) \,.\, m(1, S, R, T) \quad (12)$$

$$i(T) \,.\, i(S) \,.\, i(R) \xrightarrow{fake_2(R,S,T)} i(T) \,.\, i(S) \,.\, i(R) \,.\, m(2, S, R, T) \quad (13)$$

Bad States. A security protocol is intended to enjoy a a specific security property. In our example this property is the ability of authenticating Bob to Alice. A security property can be specified by providing a set of "bad" states, i.e. states

whose reachability implies a violation of the property. For instance, it is easy to see that any state containing both $w(0, a, a, [\,], [a, b, k_{ab}], s(\underline{1}))$ (i.e. Alice has finished the first run of session $\underline{1}$) and $w(1, a, b, [\,], [b, a, k_{ab}], \underline{1})$ (i.e. Bob is still at the beginning of session $\underline{1}$) witnesses a violation of the expected authentication property of our simple protocol and therefore it should be considered as a bad state.

2.2 Protocol Insecurity Problems

The above concepts can be recast into the concept of protocol insecurity problem. A *protocol insecurity problem* is a tuple $\Xi = \langle \mathcal{S}, \mathcal{L}, \mathcal{R}, \mathcal{I}, \mathcal{B} \rangle$ where $\mathcal{S}$ is a set of atomic formulae of a sorted first-order language called *facts*, $\mathcal{L}$ is a set of function symbols called *rule labels*, and $\mathcal{R}$ is a set of rewrite rules of the form $L \xrightarrow{\ell} R$, where L and R are finite subsets of $\mathcal{S}$ such that the variables occurring in R occur also in L, and ℓ is an expression of the form $l(\boldsymbol{x})$ where $l \in \mathcal{L}$ and $\boldsymbol{x}$ is the vector of variables obtained by ordering lexicographically the variables occurring in L. Let S be a state and $(L \xrightarrow{\ell} R) \in \mathcal{R}$, if σ is a substitution such that $L\sigma \subseteq S$, then one possible next state of S is $S' = (S \setminus L\sigma) \cup R\sigma$ and we indicate this with $S \xrightarrow{\ell\sigma} S'$. We assume the rewrite rules are *deterministic* i.e. if $S \xrightarrow{\ell\sigma} S'$ and $S \xrightarrow{\ell\sigma} S''$, then $S' \equiv S''$. The components $\mathcal{I}$ and $\mathcal{B}$ of a protocol insecurity problem are the initial state and a sets of states whose elements represent the bad states of the protocol respectively. A *solution to a protocol insecurity problem* Ξ (i.e. an attack to the protocol) is a sequence of states $S_1, \ldots, S_n$ such that $S_i \xrightarrow{\ell_i\sigma_i} S_{i+1}$ for $i = 1, \ldots, n$ and $\mathcal{I} \equiv S_1$, and there exists $S_{\mathcal{B}} \in \mathcal{B}$ such that $S_{\mathcal{B}} \subseteq S_n$.

3 Automatic SAT-Compilation of Protocol Insecurity Problems

Our proposed reduction of protocol insecurity problems to propositional logic is carried out in two steps. Protocol insecurity problems are first translated into planning problems which are in turn encoded into propositional formulae.

A *planning problem* is a tuple $\Pi = \langle \mathcal{F}, \mathcal{A}, Ops, I, G \rangle$, where $\mathcal{F}$ and $\mathcal{A}$ are disjoint sets of variable-free atomic formulae of a sorted first-order language called *fluents* and *actions* respectively; *Ops* is a set of expressions of the form

$$op(Act, Pre, Add, Del)$$

where $Act \in \mathcal{A}$ and Pre, Add, and Del are finite sets of fluents such that $Add \cap Del = \emptyset$; I and G are boolean combinations of fluents representing the initial state and the final states respectively. A state is represented by a set of fluents. An action is applicable in a state S iff the action preconditions occur in S and the application of the action leads to a new state obtained from S by removing the fluents in Del and adding those in Add. A *solution to a planning*

problem Π is a sequence of actions whose execution leads from the initial state to a final state and the precondition of each action appears in the state to which it applies.

3.1 Encoding Planning Problems into SAT

Let $\Pi = \langle \mathcal{F}, \mathcal{A}, Ops, I, G \rangle$ be a planning problem with finite $\mathcal{F}$ and $\mathcal{A}$ and let n be a positive integer, then it is possible to build a set of propositional formulae Φ^n_Π such that any model of Φ^n_Π corresponds to a partial-order plan of length n which can be linearized into a solution of Π. The encoding of a planning problem into a set of SAT formulae can be done in a variety of ways (see [17, 13] for a survey). The basic idea is to add an additional time-index to the actions and fluents to indicate the state at which the action begins or the fluent holds. Fluents are thus indexed by 0 through n and actions by 0 through $n-1$. If p is a fluent or an action and i is an index in the appropriate range, then $i{:}p$ is the corresponding time-indexed propositional variable.

The set of formulae Φ^n_Π is the smallest set (intended conjunctively) such that:

- **Initial State Axioms:** $0{:}I \in \Phi^n_\Pi$;
- **Goal State Axioms:** $n{:}G \in \Phi^n_\Pi$;
- **Universal Axioms:** for each $op(\alpha, Pre, Add, Del) \in Ops$ and $i = 0, \ldots, n-1$:

$$\begin{aligned}(i{:}\alpha \supset \textstyle\bigwedge\{i{:}p \mid p \in Pre\}) &\in \Phi^n_\Pi \\ (i{:}\alpha \supset \textstyle\bigwedge\{(i+1){:}p \mid p \in Add\}) &\in \Phi^n_\Pi \\ (i{:}\alpha \supset \textstyle\bigwedge\{\neg(i+1){:}p \mid p \in Del\}) &\in \Phi^n_\Pi\end{aligned}$$

- **Explanatory Frame Axioms:** for all fluents f and $i = 0, \ldots, n-1$:

$$(i{:}f \wedge \neg(i+1){:}f) \supset \bigvee\{i{:}\alpha \mid op(\alpha, Pre, Add, Del) \in Ops, f \in Del\} \in \Phi^n_\Pi$$

$$(\neg i{:}f \wedge (i+1){:}f) \supset \bigvee\{i{:}\alpha \mid op(\alpha, Pre, Add, Del) \in Ops, f \in Add\} \in \Phi^n_\Pi$$

- **Conflict Exclusion Axioms:** for $i = 0, \ldots, n-1$:

$$\neg(i{:}\alpha_1 \wedge i{:}\alpha_2) \in \Phi^n_\Pi$$

  for all $\alpha_1 \neq \alpha_2$ such that $op(\alpha_1, Pre_1, Add_1, Del_1) \in Ops$, $op(\alpha_2, Pre_2, Add_2, Del_2) \in Ops$, and $Pre_1 \cap Del_2 \neq \emptyset$ or $Pre_2 \cap Del_1 \neq \emptyset$.

It is immediate to see that the number of literals in Φ^n_Π is in $O(n|\mathcal{F}| + n|\mathcal{A}|)$. Moreover the number of Universal Axioms is in $O(nP_0|\mathcal{A}|)$ where P_0 is the maximal number of fluents mentioned in an operator (usually a small number); the number of Explanatory Frame Axioms is in $O(n|\mathcal{F}|)$; finally, the number of Conflict Exclusion Axioms is in $O(n|\mathcal{A}|^2)$.

3.2 Protocol Insecurity Problems as Planning Problems

Given a protocol insecurity problem $\Xi = \langle \mathcal{S}, \mathcal{L}, \mathcal{R}, \mathcal{I}, \mathcal{B} \rangle$, it is possible to build a planning problem $\Pi_\Xi = \langle \mathcal{F}_\Xi, \mathcal{A}_\Xi, Ops_\Xi, I_\Xi, G_\Xi \rangle$ such that each solution to Π_Ξ can be translated back to a solution to Ξ: $\mathcal{F}_\Xi$ is the set of facts $\mathcal{S}$; $\mathcal{A}_\Xi$ and Ops_Ξ are the smallest sets such that $\ell\sigma \in \mathcal{A}_\Xi$ and $op(\ell\sigma, L\sigma, R\sigma \setminus L\sigma, L\sigma \setminus R\sigma) \in Ops$ for all $(L \xrightarrow{\ell} R) \in \mathcal{R}$ and all ground substitutions σ; finally $I_\Xi = \bigwedge\{f \mid f \in \mathcal{I}\} \bigwedge\{\neg f \mid f \in \mathcal{S}, f \notin \mathcal{I}\}$ and $G_\Xi = \bigvee_{S_\mathcal{B} \in \mathcal{B}} \bigwedge\{f \mid f \in S_\mathcal{B}\}$. For instance, the actions associated to (6) are of the form:

$$
\begin{array}{l}
op(step_1(A, B, C, K_{ab}), \\
\quad [w(0, A, A, [\,], [A, B, K_{ab}], C), \\
\quad\ \mathit{fresh}(nc(na, C))], \\
\quad [m(1, A, B, \{nc(C)\}_{K_{ab}}), \\
\quad\ w(2, B, A, [nc(na, C)], [A, B, K_{ab}], C)], \\
\quad [w(0, A, A, [\,], [A, B, K_{ab}], C), \\
\quad\ \mathit{fresh}(nc(na, C))])
\end{array}
$$

The reduction of protocol insecurity problems to planning problems paves the way to an automatic SAT-compilation of protocol insecurity problems. However a direct application of the approach (namely the reduction of a protocol insecurity problem Ξ to a planning problem Π_Ξ followed by a SAT-compilation of Π_Ξ) is not immediately applicable. We therefore devised a set of optimizations and constraints whose combined effects often succeed in drastically reducing the size of the SAT instances.

3.3 Optimizations

Language specialization. We recall that for the reduction to propositional logic described in Section 3.1 to be applicable the set of fluents and actions must be finite. Unfortunately, the protocol insecurity problems introduced in Section 2 have an infinite number of facts and rule instances, and therefore the corresponding planning problems have an infinite number of fluents and actions. However the language can be restricted to a finite one since the set of states reachable in n steps is obviously finite (as long as the initial states comprise a finite number of facts). To determine a finite language capable to express the reachable states, it suffices to carry out a static analysis of the protocol insecurity problem.

To illustrate, let us consider again the simple protocol insecurity problem presented above and let $n = 7$, then $\|\texttt{int}\| = \{0, 1, 2\}$, $\|\texttt{user}\| = \{a, b\}$, $\|\texttt{iuser}\| = \{a, b, \texttt{intruder}\}$, $\|\texttt{key}\| = \{kab\}$, $\|\texttt{nonceid}\| = \{na\}$, $\|\texttt{session}\| = \bigcup_{i=0}^{\lfloor n/(k+1) \rfloor} s^i(\underline{1}) \bigcup_{i=0}^{\lfloor n/(k+1) \rfloor} s^i(\underline{2})$, where k is the number of protocol steps in a session run (in

this case $k = 2$),[5] $\|\texttt{number}\| = nc(\texttt{nonceid}, \texttt{session})$, $\|\texttt{func}\| = \bigcup_{i=0}^{n-1} f^i(\texttt{number})$, $\|\texttt{text}\| = \|\texttt{iuser}\| \cup \|\texttt{key}\| \cup \|\texttt{number}\| \cup \|\texttt{func}\| \cup \{\texttt{func}\}\texttt{key}$.[6]

Moreover, we can safely replace `list_of text` with [text, text, text]. The set of facts is then equal to $i(\texttt{text}) \cup \mathit{fresh}(\texttt{number}) \cup m(\texttt{int}, \texttt{iuser}, \texttt{iuser}, \texttt{text}) \cup w(\texttt{int}, \texttt{iuser}, \texttt{user}, \texttt{list\_of text}, \texttt{list\_of text}, \texttt{session})$ which consists of 10^{12} facts. This language is finite, but definitely too big for the practical applicability of the SAT encoding.

A closer look to the protocol reveals that the above language still contains many spurious facts. In particular the $m(\ldots)$, $w(\ldots)$, and $i(\cdot)$ can be specialized (e.g. by using specialized sorts to restrict the message terms to those messages which are allowed by the protocol). By analyzing carefully the facts of the form $m(\ldots)$ and $w(\ldots)$ occurring in the protocol rules of our example we can restrict the sort `func` in such a way that $\|\texttt{func}\| = \{f(\texttt{number})\}$ and replace `list_of text` with $[\texttt{iuser}, \texttt{iuser}, \texttt{key}] \cup [\texttt{number}]$. Thanks to this optimization, the number of facts drops to $12,620$.

An other important language optimization borrowed from [15] splits message terms containing pairs of messages such as $m(j, s, r, \langle msg_1, msg_2 \rangle)$ (where $\langle \_, \_ \rangle$ is the pairing operator) into two message terms $m(j, s, r, msg_1, 1)$ and $m(j, s, r, msg_2, 2)$. (Due to the simplicity of the shown protocol, splitting messages has no impact on its language size.)

Fluent splitting. The second family of optimizations is based on the observation that in $w(j, s, r, ak, ik, c)$, the union of the first three arguments with the sixth form a key (in the data base theory sense) for the relation. This allows us to modify the language by replacing $w(j, s, r, ak, ik, c)$ with the conjunction of two new predicates, namely $wk(j, s, r, ak, c)$ and $inknw(j, s, r, ik, c)$. Similar considerations (based on the observation that the initial knowledge of a principal r does not depend on the protocol step j nor on principal s) allow us to simplify $inknw(j, s, r, ik, c)$ to $inknw(r, ik, c)$. Another effective improvement stems from the observation that ak and ik are lists. By using the set of new facts $wk(j, s, r, ak_1, 1, c), \ldots, wk(j, s, r, ak_l, l, c)$ in place of $wk(j, s, r, [ak_1, \ldots, ak_l], c)$ the number of wk terms drops from $O(|\texttt{text}|^l)$ to $O(l|\texttt{text}|)$.[7] In the usual simple example the application of fluent splitting reduces the number of facts to $1,988$.

Exploiting static fluents. The previous optimization enables a new one. Since the initial knowledge of the honest principal does not change as the protocol execution makes progress, facts of the form $inknw(r, ik, c)$ occurring in the initial state are preserved in all the reachable states and those not occurring in the initial state will not be introduced. In the corresponding planning problem, this

[5] The bound on the number of steps implies a bound on the maximum number of possible session repetitions.

[6] If $S_1, \ldots, S_m$ and S' are sorts and f is a function symbol of arity $S_1, \ldots, S_m \to S'$, then $\|S_i\|$ is the set of terms of sort S_i and $f(S_1, \ldots, S_m)$ denotes $\{f(t_1, \ldots, t_m) \mid t_i \in \|S_i\|, i = 1, \ldots, m\}$.

[7] If S is a sort, then $|S|$ is the cardinality of $\|S\|$.

means that all the atoms $i : inknw(r, ik, c)$ can be replaced by $inknw(r, ik, c)$ for $i = 0, \ldots, n-1$ thereby reducing the number of propositional letters in the encoding. Moreover, since the initial state is unique, this transformation enables an off-line partial instantiation of the actions and therefore a simplification of the propositional formula.

Reducing the number of Conflict Exclusion Axioms. A critical issue in the propositional encoding technique described in Section 3.1 is the quadratic growth of the number of Conflict Exclusion Axioms in the number of actions. This fact often confines the applicability of the method to problems with a small number of actions. A way to lessen this difficulty is to reduce the number of conflicting axioms by considering the intruder knowledge as *monotonic*. Let f be a fact, S and S' be states, then we say that f is monotonic iff for all S if $f \in S$ and $S \rightarrow S'$, then $f \in S'$. Since a monotonic fluent never appears in the delete list of some action, then it cannot be a cause of a conflict. The idea here is to transform the rules so to make the facts of the form $i(\cdot)$ monotonic. The transformation on the rules is very simple as it amounts to adding the monotonic facts occurring in the left hand side of the rule to its right hand side. A consequence is that a monotonic fact simplifies the Explanatory Frame Axioms relative to it. The nice effect of this transformation is that the number of Conflict Exclusion Axioms generated by the associated planning problems drops dramatically.

Impersonate. The observation that most of the messages generated by the intruder by means of (12) and (13) are rejected by the receiver as non-expected or ill-formed suggests to restrict these rules so that the intruder sends only messages matching the patterns expected by the receiver. For each protocol rule of the form:

$$\ldots m(j,s,r,t) \;.\; w(j,s,r,ak,ik,c) \;.\; \ldots \xrightarrow{step_i(\ldots)} \ldots$$

we use a new rule of the form:

$$\ldots . w(j,s,r,ak,ik,c) . i(s) . i(r) . i(t') . \ldots \xrightarrow{impersonate_i(\ldots)} \ldots . m(j,s,r,t') . w(j,s,r,ak,ik,c) . i(s) . i(r) . i(t') . \ldots$$

This rule states that if agent r is waiting for a message t from s and the intruder knows a term t' matching t, then the intruder can impersonate s and send t'. This optimization (borrowed from [16]) often reduces the number of rule instances in a dramatic way. In our example, this optimization step allows us to trade all the 1152 instances of (12) and (13) with 120 new rules.

It is easy to see that this transformation is correct as it preserves the existing attacks and does not introduce new ones.

Step compression. A very effective optimization, called *step compression* has been proposed in [9]. It consists of the idea of merging intruder with protocol rules. In particular, an impersonate rule:

$$w(i,x_1,x_2,x_3,x_4,x_5) \cdot i(x_1) \cdot i(x_2) \cdot i(x_6) \xrightarrow{impersonate_i(\ldots)} m(i,x_1,x_2,x_6) \cdot w(i,x_1,x_2,x_3,x_4,x_5) \cdot i(x_1) \cdot i(x_2) \cdot i(x_6) \quad (14)$$

a generic protocol step rule:

$$w(i,y_1,y_2,y_3,y_4,y_5) \cdot m(i,y_1,y_2,y_6) \xrightarrow{step_i(\ldots)} w(j,y_1,y_2,y_7,y_4,y_5) \cdot m(i+1,y_2,y_1,y_8) \quad (15)$$

and a divert rule:

$$m(i+1,z_1,z_2,z_3) \xrightarrow{divert_{i+1}(\ldots)} i(z_1) \cdot i(z_2) \cdot i(z_3) \quad (16)$$

can be replaced by the following rule:

$$w(i,x_1,x_2,x_3,x_4,x_5)\sigma \cdot i(x_1)\sigma \cdot i(x_2)\sigma \cdot i(x_6)\sigma \xrightarrow{step\_comp_i(\ldots)\sigma} w(j,y_1,y_2,y_7,y_4,y_5)\sigma \cdot i(z_1)\sigma \cdot i(z_2)\sigma \cdot i(z_3)\sigma$$

where $\sigma = \sigma_1 \circ \sigma_2$ with $\sigma_1 = mgu(\{w(i,x_1,x_2,x_3,x_4,x_5) = w(i,y_1,y_2,y_3,y_4,y_5), m(i,x_1,x_2,x_6) = m(i,y_1,y_2,y_6,)$ and $\sigma_2 = mgu(\{m(i+1,y_2,y_1,y_8) = m(i+1,z_1,z_2,z_3)\})$.

The rationale of this optimization is that we can safely restrict our attention to computation paths where (14), (15), and (16) are executed in this sequence without any interleaved action in between.

By applying this optimization we reduce both the number of facts (note that the facts of the form $m(\ldots)$ are no longer needed) and the number of rules as well as the number of steps necessary to find the attacks. For instance, by using this optimization the partial-order plan corresponding to the attack to the NSPK protocol has length 7 whereas if this optimization is disabled the length is 10, the numbers of facts decreases from 820 to 505, and the number of rules from 604 to 313.

3.4 Bounds and Constraints

In some cases in order to get encodings of reasonable size, we must supplement the above attack-preserving optimizations with the following bounding techniques and constraints. Even if by applying them we may loose some attacks, in our experience (cf. Section 4) this rarely occurs in practice.

Bounding the number of session runs. Let n and k be the bounds in the number of operation applications and in the number of protocol steps characterizing a protocol session respectively. Then the maximum number of times a session can be repeated is $\lfloor n/(k+1) \rfloor$. Our experience indicates that attacks usually require a number of session repetitions that is less than $\lfloor n/(k+1) \rfloor$. As a matter of fact two session repetitions are sufficient to find attacks to all the protocols we have analyzed so far. By using this optimization we can reduce the cardinality of the sort `session` (in the case of the NSPK protocol, we reduce it by a factor 1.5) and therefore the number of facts that depend on it.

Multiplicity of fresh terms. The number of fresh terms needed to find an attack is usually less than the number of fresh terms available. As a consequence, a lot of fresh terms allowed by the language associated with the protocol are not used, and many facts depending on them are allowed, but also not used. Often, one single fresh term for each fresh term identifier is sufficient for finding the attack. For instance the simple example shown above has the only fresh term identifier na and to use the only nonce $nc(na, \underline{1})$ is enough to detect the attack. Therefore, the basic idea of this constraint is to restrict the number of fresh terms available, thereby reducing the size of the language. For example, application of this constraint to the analysis of the NSPK protocol protocol preserves the detection of the attack and reduces the numbers of facts and rules from 313 to 87 and from 604 to 54 respectively. Notice that for some protocols such as the Andrew protocol the multiplicity of fresh terms is necessary to detect the attack.

Constraining the rule variables. This constraint is best illustrated by considering the Kao-Chow protocol (see e.g. [7]):

$$
\begin{array}{ll}
(1) & A \rightarrow S : A, B, N_a \\
(2) & S \rightarrow B : \{A, B, N_a, K_{ab}\}K_{as}, \{A, B, N_a, K_{ab}\}K_{bs} \\
(3) & B \rightarrow A : \{A, B, N_a, K_{ab}\}K_{as}, \{N_a\}K_{ab}, N_b \\
(4) & A \rightarrow B : \{N_b\}K_{ab}
\end{array}
$$

During the step (2) S sends B a pair of messages of which only the second component is accessible to B. Since B does not know K_{ab}, then B cannot check that the occurrence of A in the first component is equal to that inside the second. As a matter of fact, we might have different terms at those positions. The constraint amounts to imposing that the occurrences of A (as well as of B, N_a, and K_{ab}) in the first and in the second part of the message must coincide. Thus, messages of the form $\{a, b, nc(na, s(\underline{1}))\}kas, \{a, b, nc(nb, s(\underline{1}))\}kbs$ would be ruled out by the constraint. The application of this constraint allows us to get a feasible encoding of the Kao-Chow protocols in reasonable time. For instance, with this constraint disabled the encoding of the Kao Chow Repeated Authentication 1 requires more than 1 hour, otherwise it requires 16.34 seconds.

4 Implementation and Computer Experiments

We have implemented the above ideas in SATMC, a SAT-based Model-Checker for security protocol analysis. Given a protocol insecurity problem Ξ, a bound on the length of partial-order plan n, and a set of parameters specifying which bounds and constraints must be enabled (cf. Section 3.4), SATMC first applies the optimizing transformations previously described to Ξ and obtains a new protocol insecurity problem Ξ', then Ξ' is translated into a corresponding planning problem $\Pi_{\Xi'}$ which is in turn compiled into SAT using the methodology outlined in Section 3.1. The propositional formula is then fed to a state-of-the-art SAT solver (currently Chaff [20], SIM [14], and SATO [23] are supported) and any model found by the solver is translated back into an attack which is reported to the user.

SATMC is one of the back-ends of the AVISS tool [2]. Using this tool, the user can specify a protocol and the security properties to be checked using a high-level specification language and the tool translates the specification in an Intermediate Format (IF) based on multiset rewriting. The notion of protocol insecurity problem given in this paper is inspired by the Intermediate Format. Some of the features supported by the IF (e.g. public and private keys, compound keys as well as other security properties such as authentication and secrecy) have been neglected in this paper for the lack of space. However, they are supported by SATMC.

We have run our tool against a selection of problems drawn from [7]. The results of our experiments are reported in Table 1 and they are obtained by applying all the previously described optimizations, by setting $n = 10$, by imposing two session runs for session, by allowing multiple fresh terms, and by constraining the rule variables. For each protocol we give the kind of the attack found (Attack), the number of propositional variables (Atoms) and clauses (Clauses), and the time spent to generate the SAT formula (EncT) as well as the time spent by Chaff to solve the corresponding SAT instance (SolveT). The label MO indicates a failure to analyze the protocol due to memory-out.[8] It is important to point out that for the experiments we found it convenient to disable the generation of Conflict Exclusion Axioms during the generation of the propositional encoding. Of course, by doing this, we are no longer guaranteed that the solutions found are linearizable and hence executable. SATMC therefore performs a check on any partial order plan found and if any conflict is detected, then clauses negating those conflicts are added to the propositional formula and the resulting formula is fed to again to the SAT-solver. This procedure is repeated until a solution without conflicts is found or no other models are found by the SAT solver. The experiments show that the SAT solving activity is carried out very quickly and that the overall time is dominated by the SAT encoding.

[8] Times have been obtained on a PC with a 1.4 GHz Processor and 512 MB of RAM. Due to a limitation of SICStus Prolog the SAT-based model-checker is bound to use 128 MB during the encoding generation.

Table 1. Performance of SATMC

Protocol	Attack	Atoms	Clauses	EncT	SolveT
ISO symmetric key 1-pass unilateral authentication	Replay	679	2,073	0.18	0.00
ISO symmetric key 2-pass mutual authentication	Replay	1,970	7,382	0.43	0.01
Andrew Secure RPC Protocol	Replay	161,615	2,506,889	80.57	2.65
ISO CCF 1-pass unilateral authentication	Replay	649	2,033	0.17	0.00
ISO CCF 2-pass mutual authentication	Replay	2,211	10,595	0.46	0.00
Needham-Schroeder Conventional Key	Replay STS	126,505	370,449	29.25	0.39
Woo-Lam II	Parallel-session	7,988	56,744	3.31	0.04
Woo-Lam Mutual Authentication	Parallel-session	771,934	4,133,390	1,024.00	7.95
Needham-Schroeder Signature protocol	MM	17,867	59,911	3.77	0.05
Neuman Stubblebine repeated part	Replay STS	39,579	312,107	15.17	0.21
Kehne Langendorfer Schoenwalder (repeated part)	Parallel-session	-	-	MO	-
Kao Chow Repeated Authentication, 1	Replay STS	50,703	185,317	16.34	0.17
Kao Chow Repeated Authentication, 2	Replay STS	586,033	1,999,959	339.70	2.11
Kao Chow Repeated Authentication, 3	Replay STS	1,100,428	6,367,574	1,288.00	MO
ISO public key 1-pass unilateral authentication	Replay	1,161	3,835	0.32	0.00
ISO public key 2-pass mutual authentication	Replay	4,165	23,883	1.18	0.01
Needham-Schroeder Public Key	MM	9,318	47,474	1.77	0.05
Needham-Schroeder Public Key with key server	MM	11,339	67,056	4.29	0.04
SPLICE/AS Authentication Protocol	Replay	15,622	69,226	5.48	0.05
Encrypted Key Exchange	Parallel-session	121,868	1,500,317	75.39	1.78
Davis Swick Private Key Certificates, protocol 1	Replay	8,036	25,372	1.37	0.02
Davis Swick Private Key Certificates, protocol 2	Replay	12,123	47,149	2.68	0.03
Davis Swick Private Key Certificates, protocol 3	Replay	10,606	27,680	1.50	0.02
Davis Swick Private Key Certificates, protocol 4	Replay	27,757	96,482	8.18	0.13

Legenda: MM: Man-in-the-middle attack
Replay STS: Replay attack based on a Short-Term Secret
MO: Memory Out

5 Conclusions and Perspectives

We have proposed an approach to the translation of protocol insecurity problems into propositional logic based on the combination of a reduction to planning and well-known SAT-reduction techniques developed for planning. Moreover, we have introduced a set of optimizing transformations whose application to the input protocol insecurity problem drastically reduces the size of the corresponding propositional encoding. We have presented SATMC, a model-checker based on our ideas, and shown that attacks to a set of well-known authentication protocols are quickly found by state-of-the-art SAT solvers.

Since the time spent by SAT solver is largely dominated by the time needed to generate the propositional encoding, in the future we plan to keep working on ways to reduce the latter. A promising approach amounts to treating properties of cryptographic operations as invariants. Currently these properties are modeled as rewrite rules (cf. rule (10) in Section 2.1) and this has a bad impact on the size of the final encoding. A more natural way to deal with these properties amounts to building them into the encoding but this requires, among other things, a modification of the explanatory frame axioms and hence more work (both theoretical and implementational) is needed to exploit this very promising transformation.

Moreover, we would like to experiment SATMC against security problems with partially defined initial states. Problems of this kind occur when the initial knowledge of the principal in not completely defined or when the session instances are partially defined. We conjecture that neither the size of the SAT encoding nor the time spent by the SAT solver to check the SAT instances will be significantly affected by this generalization. But this requires some changes in the current implementation of SATMC and a thorough experimental analysis.

References

1. Luigia Carlucci Aiello and Fabio Massacci. Verifying security protocols as planning in logic programming. *ACM Transactions on Computational Logic*, 2(4):542–580, October 2001.
2. A. Armando, D. Basin, M. Bouallagui, Y. Chevalier, L. Compagna, S. Moedersheim, M. Rusinowitch, M. Turuani, L. Viganò, and L. Vigneron. The AVISS Security Protocols Analysis Tool. In *14th International Conference on Computer-Aided Verification (CAV'02)*. 2002.
3. David Basin and Grit Denker. Maude versus haskell: an experimental comparison in security protocol analysis. In Kokichi Futatsugi, editor, *Electronic Notes in Theoretical Computer Science*, volume 36. Elsevier Science Publishers, 2001.
4. D. Bolignano. Towards the formal verification of electronic commerce protocols. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 133–146. 1997.
5. Common Authentication Protocol Specification Language. URL `http://www.csl.sri.com/~millen/capsl/`.
6. Cervesato, Durgin, Mitchell, Lincoln, and Scedrov. Relating strands and multiset rewriting for security protocol analysis. In *PCSFW: Proceedings of The 13th Computer Security Foundations Workshop*. IEEE Computer Society Press, 2000.

7. John Clark and Jeremy Jacob. A Survey of Authentication Protocol Literature: Version 1.0, 17. Nov. 1997. URL `http://www.cs.york.ac.uk/~jac/papers/drareview.ps.gz`.
8. Ernie Cohen. TAPS: A first-order verifier for cryptographic protocols. In *Proceedings of The 13th Computer Security Foundations Workshop*. IEEE Computer Society Press, 2000.
9. Sebastian Moedersheim David Basin and Luca Viganò. An on-the-fly model-checker for security protocol analysis. forthcoming, 2002.
10. Grit Denker, Jonathan Millen, and Harald Rueß. The CAPSL Integrated Protocol Environment. Technical Report SRI-CSL-2000-02, SRI International, Menlo Park, CA, October 2000. Available at `http://www.csl.sri.com/~millen/capsl/`.
11. Danny Dolev and Andrew Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.
12. B. Donovan, P. Norris, and G. Lowe. Analyzing a library of security protocols using Casper and FDR. In *Proceedings of the Workshop on Formal Methods and Security Protocols*. 1999.
13. Michael D. Ernst, Todd D. Millstein, and Daniel S. Weld. Automatic SAT-compilation of planning problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1169–1177. Morgan Kaufmann Publishers, San Francisco, 1997.
14. Enrico Giunchiglia, Marco Maratea, Armando Tacchella, and Davide Zambonin. Evaluating search heuristics and optimization techniques in propositional satisfiability. In Rajeev Goré, Aleander Leitsch, and Tobias Nipkow, editors, *Proceedings of IJCAR'2001*, pages 347–363. Springer-Verlag, 2001.
15. Mei Lin Hui and Gavin Lowe. Fault-preserving simplifying transformations for security protocols. *Journal of Computer Security*, 9(1/2):3–46, 2001.
16. Florent Jacquemard, Michael Rusinowitch, and Laurent Vigneron. Compiling and Verifying Security Protocols. In M. Parigot and A. Voronkov, editors, *Proceedings of LPAR 2000*, LNCS 1955, pages 131–160. Springer-Verlag, Heidelberg, 2000.
17. Henry Kautz, David McAllester, and Bart Selman. Encoding plans in propositional logic. In Luigia Carlucci Aiello, Jon Doyle, and Stuart Shapiro, editors, *KR'96: Principles of Knowledge Representation and Reasoning*, pages 374–384. Morgan Kaufmann, San Francisco, California, 1996.
18. Gawin Lowe. Casper: a compiler for the analysis of security protocols. *Journal of Computer Security*, 6(1):53–84, 1998. See also `http://www.mcs.le.ac.uk/~gl7/Security/Casper/`.
19. Catherine Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996. See also `http://chacs.nrl.navy.mil/projects/crypto.html`.
20. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*. 2001.
21. L.C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1):85–128, 1998.
22. D. Song. Athena: A new efficient automatic checker for security protocol analysis. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop (CSFW '99)*, pages 192–202. IEEE Computer Society Press, 1999.
23. H. Zhang. SATO: An efficient propositional prover. In William McCune, editor, *Proceedings of CADE 14*, LNAI 1249, pages 272–275. Springer-Verlag, Heidelberg, 1997.