

SAT-based Model-Checking of Security Protocols using Planning Graph Analysis^{*}

Alessandro Armando, Luca Compagna, and Pierre Ganty

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

Abstract. In previous work we showed that automatic SAT-based model-checking techniques based on a reduction of protocol insecurity problems to satisfiability problems in propositional logic (SAT) can be used effectively to find attacks on security protocols. The approach results from the combination of a reduction of protocol insecurity problems to planning problems and well-known SAT-reduction techniques, called linear encodings, developed for planning. Experimental results confirmed the effectiveness of the approach but also showed that the time spent to generate the SAT formula largely dominates the time spent by the SAT solver to check its satisfiability. Moreover, the SAT instances generated by the tool get of unmanageable size on the most complex protocols. In this paper we explore the application of the Graphplan-based encoding technique to the analysis of security protocols and present experimental data showing that Graphplan-based encodings are considerably (i.e. up to 2 orders of magnitude) smaller than linear encodings. These results confirm the effectiveness of the SAT-based approach to the analysis of security protocols and pave the way to its application to large protocols arising in practical applications.

Keywords: bounded model-checking, security protocols, SAT-solvers, SAT encodings.

Experience Paper

1 Introduction

Security (or cryptographic) protocols are communication protocols that aim at providing security guarantees (such as authentication of principals or secrecy of some piece of information) through the application of cryptographic primitives. Since these protocols are at the core of security-sensitive applications in a variety

^{*} This work was partially funded by the IHP-RTN EC project CALCULEMUS (HPRN-CT-2000-00102), by the FET Open EC Project AVISPA (IST-2001-39252), and by the project “Convenzione per lo svolgimento di tesi di dottorato in una Network di istituzioni europee e mutuo riconoscimento del titolo di dottore di ricerca. (Dottorato in Ingegneria Elettronica e Informatica)” of MIUR.

of domains (e.g. health-care, e-commerce, and e-government), their proper functioning is crucial as a failure may undermine the customer and, more in general, the public trust in these applications.

The problem is that—in spite of their apparent simplicity—security protocols are notoriously error-prone. Many published protocols have been implemented and deployed in real applications only to be found flawed years later. For instance, the Needham-Schroeder authentication protocol [23] was found vulnerable to a serious attack 17 years after its publication [20]. Quite interestingly, many attacks can be carried out without breaking cryptography. These attacks exploit weaknesses in the protocol that are due to the complex and unexpected interleavings of different protocol sessions as well as to the possible interference of malicious agents. Since these weaknesses are very difficult to spot by simple inspection of the protocol specification, security protocols have received growing attention by the formal methods community as a new, very promising and challenging application domain.

In previous work [3] we showed that automatic SAT-based model-checking techniques based on a reduction of protocol insecurity problems to satisfiability problems in propositional logic (SAT) can be used effectively to find attacks on security protocols. The approach results from the combination of a reduction of protocol insecurity problems to planning problems and well-known SAT-reduction techniques, called *linear encodings*, developed for planning (see [18] for a survey on the topic). A model-checker, SATMC, based on our ideas has been developed and experimental results obtained by running SATMC against security protocols drawn from the Clark-Jacob’s library [10] confirm the effectiveness of the approach but also show that the time spent to generate the SAT formula largely dominates the time spent by the SAT solver to check its satisfiability. Moreover, the SAT instances generated by the tool get of unmanageable size on the most complex protocols. To cope with the problem in [4] we propose a new model-checking procedure based on an abstraction/refinement loop which interleaves the encoding and the solving phases. In this paper we follow a different route and explore the application of a sophisticated SAT-reduction technique, *Graphplan-based encoding* [18], which has been used with success in AI planning.

Even though linear and Graphplan-based encoding techniques have the same worst case (time and space) complexity, experimental data obtained by running SATMC on protocols drawn from the Clark-Jacob’s library clearly indicate that Graphplan-based encodings are considerably (i.e. up to 2 orders of magnitude) smaller than linear encodings. These results confirm the effectiveness of the SAT-based approach to the analysis of security protocols and pave the way to its application to large protocols arising in practical applications. To the best of our knowledge our work is the first (successful) application of Graphplan-based encodings in bounded model-checking [6].

Structure of the paper. We start in Section 2 by introducing security protocol via a very simple (flawed) authentication protocol. In Section 3 we define the notion of protocol insecurity problem and show that it can be seen as a planning problem. Section 4 is devoted to the formal description of the linear and the

Graphplan-based encodings together with the presentation of the experimental results. The related work is discussed in Section 5. We conclude in Section 6 with some final remarks and a discussion of the future work.

2 A Simple Example

As mentioned in Section 1 even small and convincing protocols are often wrong. To illustrate, consider the following one-way authentication protocol:

- (1) $A \rightarrow B : \{N\}_K$
- (2) $B \rightarrow A : \{f(N)\}_K$

where N is a nonce¹ generated by Alice, K 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 deceive 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) $A \rightarrow I(B) : \{N\}_K$
- (2.1) $I(B) \rightarrow A : \{N\}_K$
- (2.2) $A \rightarrow I(B) : \{f(N)\}_K$
- (1.2) $I(B) \rightarrow A : \{f(N)\}_K$

Alice starts the protocol with message (1.1).² 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.

3 Protocol Insecurity Problems and Planning Problems

We model the concurrent execution of a protocol by means of a state transition system. Following [8, 17], we represent states by sets of variables-free atomic formulae and transitions by means of rewrite rules over sets of facts.

¹ *Nonces* are numbers generated by principals that are intended to be used *only once*.

² Notice that with $(i.j)$ we indicate that the message has been sent at protocol step j of session i .

3.1 Protocol Insecurity Problems

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 $l(\mathbf{x})$ where $l \in \mathcal{L}$ and \mathbf{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 $S' = S''$. The components \mathcal{I} and \mathcal{B} of a protocol insecurity problem are the initial state and a set of 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 rewrite rules $l_1\sigma_1, \dots, l_n\sigma_n$ such that $S_i \xrightarrow{\ell_i\sigma_i} S_{i+1}$ for $i = 1, \dots, n$ with $S_1 = \mathcal{I}$ and $S_n \in \mathcal{B}$.

A protocol insecurity problem specifies the runs allowed by the protocol when embedded in a hostile environment together with an initial state and a set of bad states (i.e. states whose reachability implies the violation of the desired security properties). The states of the transition system model the state of the honest principals, the knowledge of the intruder, as well as the messages sent over the channel but not yet processed by the intended recipients (or diverted by the intruder). Rewrite rules model the legal transitions that can be performed by honest participants as well as the abilities of the intruder. For the simple protocol presented in Section 2, facts are of the form:

- $i(t)$, meaning that the intruder knows the term t ;
- $c(t)$, meaning that the fresh terms counter is equal to t ;
- $m(j, s, r, t)$, meaning that a message t has been sent (supposedly) from principal s to principal r at protocol step j , and
- $w(j, s, r, [t_1, \dots, t_k])$, representing the state of execution of principal r at step j ; it means that r knows the terms t_1, \dots, t_k at step j , and (if $j = 1, 2$) that r is waiting for a message from s for step j to be executed.

The initial state of the system is:³

$$c(0) \cdot w(0, a, a, []) \cdot w(1, a, b, []) \cdot w(0, b, b, []) \cdot w(1, b, a, []) \cdot i(a) \cdot i(b)$$

Facts $w(0, a, a, [])$, $w(1, a, b, [])$, $w(0, b, b, [])$, and $w(1, b, a, [])$ state that principals a and b are ready to play both the role of the initiator and of the responder. Fact $c(0)$ states that the fresh terms counter is initialized with the value 0. Finally $i(a)$ and $i(b)$ state that the identities of a and b are known to the intruder.

³ To improve readability we use the “.” operator as set constructor. For instance, we write “ $x \cdot y \cdot z$ ” to denote the set $\{x, y, z\}$.

The behavior of the honest principals and of the intruder is specified by means of rewrite rules. The activity of sending the first message is modeled by:⁴

$$c(T) \cdot w(0, A, A, []) \xrightarrow{step_1(A, B, T)} c(s(T)) \cdot m(1, A, B, \{n(T)\}_k) \\ \cdot w(2, B, A, [f(n(T))])$$

Notice that in the above rule a nonce is generated thus the counter of fresh terms is incremented. Notice also that term $f(n(T))$ 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:

$$m(1, A, B, \{n(T)\}_k) \cdot w(1, A, B, []) \xrightarrow{step_2(A, B, T)} m(2, B, A, \{f(n(T))\}_k) \\ \cdot w(3, B, B, [])$$

The final step of the protocol is modeled by:

$$m(2, B, A, \{f(n(T))\}_k) \cdot w(2, B, A, [f(n(T))]) \xrightarrow{step_3(A, B, T)} w(4, A, A, [])$$

where steps 3 and 4 occurring as first parameter in w -fact are used to denote the final state of the responder and of the initiator, respectively.

The following rule models the ability of the intruder of diverting the information exchanged by the honest participants:

$$m(J, S, R, T) \xrightarrow{divert(J, R, S, T)} i(R) \cdot i(S) \cdot i(T) \quad (1)$$

The ability of encrypting and decrypting messages is modeled by:

$$i(T) \cdot i(K) \xrightarrow{encrypt(K, T)} i(\{T\}_K) \quad (2)$$

$$i(\{T\}_K) \cdot i(K) \xrightarrow{decrypt(K, T)} i(T) \quad (3)$$

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

$$i(T) \cdot i(S) \cdot i(R) \xrightarrow{fake(J, R, S, T)} m(J, S, R, T)$$

Notice that with the above rules we represent the most general intruder based on the Dolev-Yao model [12]. In this model the intruder has the abilities to eavesdrop, divert and memorize messages as well as to compose, decompose, encrypt and decrypt—when he has the decryption key i.e. *perfect cryptography*—messages. Finally, he can send those messages to other participants with a false identity. It is worth pointing out that the rewrite rule formalism allows us to represent others intruders models. For instance, suppose honest agents belong to a local network, while the intruder does not. In this case the intruder cannot

⁴ Here and in sequel we use capital letters to denote variables.

overhear and/or divert messages exchanged between honest agents. This can be simply modelled by removing the rewrite rule (1).

A security protocol is intended to enjoy 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, any state containing a subset of facts of the form $w(4, A, A, []) \cdot w(1, A, B, [])$ (i.e. A has finished a run of the protocol as initiator and B is still at the beginning of the protocol run as responder) witnesses a violation of the expected authentication property and therefore it should be considered as a bad state. It is easy to build a propositional formula G such that each model of G represents a bad state. For the above example $G \equiv (w(4, a, a, []) \wedge w(1, a, b, [])) \vee (w(4, b, b, []) \wedge w(1, b, a, []))$.

3.2 Planning Problem

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 $(Pre \xrightarrow{Act} Add ; Del)$ where $Act \in \mathcal{A}$ and $Pre, Add,$ and Del are finite sets of fluents such that $Add \cap Del = \emptyset$; I is a set of fluents representing the initial state and G is a boolean combination of fluents representing the final states. A state is represented by a set S of fluents meaning that all the fluents in S hold in the state, while all the fluents in $\mathcal{F} \setminus S$ do not hold in the state (close-world-assumption). An action is applicable in a state S iff the action preconditions (fluents in Pre) 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 , called *plan*, is a sequence of actions whose execution leads from the initial state to a final state and the preconditions of each action appears in the state to which it applies. The *length* of a plan is the number of actions occurring in it. Plans can be represented in a compact way by means of a partial-order plan. A *partial-order plan* is a pair $\langle A, \leq \rangle$ where A is a set of pairs $\langle \alpha, i \rangle$ such that $\alpha \in \mathcal{A}$ and $i \in \{0, 1, \dots\}$, and \leq is a partial order⁵ on $\{0, 1, \dots\}$. A plan \mathcal{P} is in the set of the plans denoted by the partial-order plan $\langle A, \leq \rangle$ iff (i) there exists a bijection between \mathcal{P} and A and (ii) for each $\langle \alpha, i \rangle, \langle \beta, j \rangle \in A$ such that $j \not\leq i$ there is a subsequence of \mathcal{P} in which α precedes β . The *length* of the partial-order plan $\langle A, \leq \rangle$ is the cardinality of the set $\{i \mid \langle \alpha, i \rangle \in A\}$. For instance, the partial-order plan $\langle \{\langle a, 0 \rangle, \langle b, 0 \rangle, \langle c, 3 \rangle, \langle d, 5 \rangle, \langle a, 5 \rangle\}, \{0 \leq 0, 0 \leq 3, 0 \leq 5, 3 \leq 3, 3 \leq 5, 5 \leq 5\} \rangle$ has length 3 and represents the set of plans $\{\langle a, b, c, d, a \rangle, \langle b, a, c, d, a \rangle, \langle a, b, c, a, d \rangle, \langle b, a, c, a, d \rangle\}$.

3.3 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

⁵ A reflexive, antisymmetric, and transitive binary relation.

Π_{Ξ} can be translated back to a solution to Ξ : \mathcal{F}_{Ξ} is the set of facts \mathcal{S} ; \mathcal{A}_{Ξ} and Ops_{Ξ} are the smallest sets such that $\ell\sigma \in \mathcal{A}_{\Xi}$ and $L\sigma \xrightarrow{\ell\sigma} R\sigma \setminus L\sigma$; $L\sigma \setminus R\sigma \in Ops_{\Xi}$ for all $(L \xrightarrow{\ell} R) \in \mathcal{R}$ and all ground substitutions σ ; finally $I_{\Xi} \equiv \mathcal{I}$ and $G_{\Xi} = \bigvee_{S_{\mathcal{B}} \in \mathcal{B}} (\bigwedge S_{\mathcal{B}} \wedge \mathcal{S} \setminus S_{\mathcal{B}})$.

4 Automatic SAT-Compilation of Planning Problems

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 propositional formula Φ_{Π}^n such that any model of Φ_{Π}^n corresponds to a partial-order plan of length n representing solutions of Π . The encoding of a planning problem into a set of SAT formulae can be done in a variety of ways (see [18, 14] 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 p^i is the corresponding time-indexed propositional variable.

In the rest of this section we will formally describe the linear and the Graphplan-based encodings. These encoding techniques have been implemented in SATMC [3]. In order to compare them in the domain of security protocols, we have run SATMC against a selection of (flawed) security protocols drawn from [10]. For each protocol we have built a corresponding protocol insecurity problem modeling a scenario with a bounded number of principals which exchange messages on a channel controlled by the most general intruder based on the Dolev-Yao model. Moreover, we assume perfect cryptography (see Section 2) and that all atoms are typed i.e. we do not allow for type confusion (*strong typing assumption*).⁶

It is worth pointing out that SATMC is one of the back-ends of the AVISS tool [2]. Using the tool, the user can specify a protocol and the security properties to be checked using a high-level specification language similar to the Alice&Bob notation we used in Section 2 to present our simple authentication protocol. The AVISS tool translates the specification into a rewrite-based declarative Intermediate Format (IF) based on multiset rewriting which is amenable to formal analysis. SATMC can optionally accept protocol specifications in the IF language which are then automatically translated into equivalent planning problems.

4.1 The Linear Encoding

By using linear encoding techniques, Φ_{Π}^n is defined by

$$\Phi_{\Pi}^n = \iota(\mathbf{f}^0) \wedge \bigwedge_{i=0}^{n-1} \tau(\mathbf{f}^i, \boldsymbol{\alpha}^i, \mathbf{f}^{i+1}) \wedge \gamma(\mathbf{f}^n) \quad (4)$$

⁶ As pointed out in [16] type-flaw attacks can be prevented by tagging the fields of a message with information indicating its intended type.

where \mathbf{f} and $\boldsymbol{\alpha}$ are vectors of the fluents and actions in \mathcal{F} and \mathcal{A} respectively and

- $\iota(\mathbf{f}^0)$ is a formula encoding the initial state and is a conjunction of the formulae f^0 if $f \in I$ and $\neg f^0$ if $f \notin I$;
- $\gamma(\mathbf{f}^n)$ is a formula encoding the final states and is obtained from G by replacing each fluent f with f^n ;
- $\tau(\mathbf{f}^i, \boldsymbol{\alpha}^i, \mathbf{f}^{i+1})$ is a formula encoding the transition relation and is a conjunction of the *Universal Axioms*:

$$\begin{aligned}\alpha^i &\supset \bigwedge \{f^i \mid f \in Pre\} \\ \alpha^i &\supset \bigwedge \{f^{i+1} \mid f \in Add\} \\ \alpha^i &\supset \bigwedge \{\neg f^{i+1} \mid f \in Del\}\end{aligned}$$

for each $(Pre \xrightarrow{\alpha} Add ; Del) \in Ops$, the *Explanatory Frame Axioms*:

$$\begin{aligned}(f^i \wedge \neg f^{i+1}) &\supset \bigvee \left\{ \alpha^i \mid (Pre \xrightarrow{\alpha} Add ; Del) \in Ops, f \in Del \right\} \\ (\neg f^i \wedge f^{i+1}) &\supset \bigvee \left\{ \alpha^i \mid (Pre \xrightarrow{\alpha} Add ; Del) \in Ops, f \in Add \right\}\end{aligned}$$

for all fluents f , and the *Conflict Exclusion Axioms (CEA)*: $\neg(\alpha_1^i \wedge \alpha_2^i)$ for all $\alpha_1 \neq \alpha_2$ such that $(Pre_1 \xrightarrow{\alpha_1} Add_1 ; Del_1) \in Ops$, $(Pre_2 \xrightarrow{\alpha_2} Add_2 ; Del_2) \in Ops$, 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_{Π}^n is in $O(n|\mathcal{F}| + n|\mathcal{A}|)$. Moreover the number of clauses generated by the 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 clauses generated by the Explanatory Frame Axioms is in $O(n|\mathcal{F}|)$; finally, the number of clauses generated by the CEA is in $O(n|\mathcal{A}|^2)$.

Computer experiments obtained by using linear encodings with increasing values of n and feeding the propositional formula generated at each step to a state-of-the-art SAT solver⁷ soon showed that solving time is largely dominated by encoding time and that the latter is strictly related to the size of the SAT instances generated. We thus found it convenient to apply an *Abstraction Refinement Loop* [4] based on the idea of disabling the generation of the CEA and checking if the “pseudo” partial-order plan⁸ found can be linearized (and hence executed). SATMC therefore looks for conflicting actions in the pseudo partial-order plan found and extends the previously generated formula with clauses negating the conflicts (if any). The resulting formula is then fed back to the SAT-solver and the whole procedure is iterated until a solution without conflicts is met or the formula becomes unsatisfiable. The results of our experiments are reported in Table 1 with the generation of the CEA enabled (**CEA=on**) and

⁷ Currently Chaff [22], SIM [15], and SATO [26] are supported.

⁸ A “pseudo” partial-order plan corresponds to a set of sequences of actions such that each sequence in the set is not guaranteed to be executable.

disabled (**CEA=off**).⁹ For each protocol we give the smallest value of n at which the attack is found (**N**), the number of propositional variables (**A**) and clauses (**CL**) in the SAT formula (in thousands), the time spent to generate the SAT formula (**EncT**), the time spent by Chaff to solve the last SAT formula (**Last**), and the total time spent by Chaff to solve all the SAT formulae generated for that protocol (**Tot**).¹⁰ If the generation of the CEA is disabled, then the number of iterations of the Abstraction Refinement Loop is also given (**#**).

Table 1. Experimental data using the linear encoding

Protocol	N	A	CEA = on				CEA = off				
			CL	EncT	SolvingT		CL	EncT	SolvingT		#
					Last	Tot			Last	Tot	
<i>Andrew</i>	9	145	-	-	-	-	2,256	111.4	2.0	12.1	1
<i>EKE</i>	5	62	13,949	7,100	7.6	19.7	783	74.1	0.7	3.7	2
<i>ISO-CCF-1 U</i>	4	< 1	< 1	0.1	0.0	0.0	< 1	0.1	0.0	0.0	0
<i>ISO-CCF-2 M</i>	4	< 1	6	0.3	0.0	0.1	4	0.2	0.0	0.1	0
<i>ISO-PK-1 U</i>	4	< 1	2	0.1	0.0	0.0	2	0.1	0.0	0.0	0
<i>ISO-PK-2 M</i>	4	2	17	0.9	0.0	0.0	10	0.6	0.1	0.1	0
<i>ISO-SK-1 U</i>	4	< 1	< 1	0.1	0.0	0.0	< 1	0.1	0.0	0.0	1
<i>ISO-SK-2 M</i>	4	< 1	3	0.4	0.0	0.0	3	0.4	0.0	0.0	0
<i>KaoChow 1</i>	7	36	355	18.4	0.1	0.7	131	8.0	0.1	0.7	4
<i>KaoChow 2</i>	9	586	35,178	1,494	-	-	1,804	140.4	1.6	15.6	5
<i>KaoChow 3</i>	9	995	-	-	-	-	5,737	585.5	7.5	41.6	1
<i>KLS rep.</i>	-	-	-	-	-	-	-	-	-	-	-
<i>NSCK</i>	9	115	787	41.3	0.4	1.6	334	17.1	0.3	1.3	0
<i>NSPK</i>	7	7	51	2.3	0.1	0.1	33	1.5	0.1	0.1	0
<i>NSPK-server</i>	8	9	212	8.0	0.1	0.2	54	2.8	0.1	0.1	0
<i>SPLICE</i>	9	14	91	4.6	0.1	0.2	62	3.6	0.1	0.2	0
<i>Swick 1</i>	5	4	17	1.0	0.1	0.1	13	0.8	0.0	0.1	1
<i>Swick 2</i>	6	8	59	3.2	0.1	0.1	29	1.7	0.1	0.1	0
<i>Swick 3</i>	4	5	12	0.8	0.1	0.1	11	0.7	0.0	0.1	1
<i>Swick 4</i>	5	15	64	11.0	0.1	0.2	57	10.2	0.1	0.3	1
<i>Stubblebine rep</i>	3	13	2,048	82.9	0.3	0.6	95	6.3	0.1	0.1	1
<i>Woo-Lam M</i>	6	481	-	-	-	-	2,498	304.4	1.8	7.7	1

- means that this data information is not available, because a memory out has been reached during the protocol analysis;

< 1 means that the number of atoms or clauses is less than 1 thousand.

As anticipated, the data show that solving time is largely dominated by encoding time. However the size of the SAT formulae and the time to generate

⁹ Experiments have been carried out on a PC with a 1.4 GHz Processor and 1 GB of RAM.

¹⁰ Times are measured in seconds.

them drop significantly if CEA are disabled and the Abstraction Refinement Loop is activated. Notice that by applying the Abstraction Refinement strategy we are able to discover attacks to security protocols such as *Andrew*, *KaoChow 3* and *Woo-Lam M* that could not be analyzed with CEA enabled.

4.2 The Graphplan-based Encoding¹¹

By using the linear encoding technique, the encoding of the transition relation — $\tau(\mathbf{f}^i, \boldsymbol{\alpha}^i, \mathbf{f}^{i+1})$ — is independent from the time step and this means that important simplifications are possible on the resulting formula. For instance, not all the actions are applicable at time step 0 but the formula

$$\tau(\mathbf{f}^0, \boldsymbol{\alpha}^0, \mathbf{f}^1) \quad (5)$$

encodes the effects of all possible actions. By looking at the initial state it is possible to build a simple but equivalent version of (5), say

$$\tau_0(\mathbf{f}^0, \boldsymbol{\alpha}^0, \mathbf{f}^1).$$

The same line of reasoning can be applied at the subsequent steps: by computing an over-approximation of the reachable steps at time step i we can then determine a simplified encoding of the transition relation at time step i , say $\tau_i(\mathbf{f}^i, \boldsymbol{\alpha}^i, \mathbf{f}^{i+1})$, for $i = 0, \dots, n - 1$.

Graphplan-based encoding¹² is based on this idea and preliminary to the generation of the encoding is the construction of a data structure (called planning graph) used to determine (among other things) an over-approximation of the reachable states at each time step i .

Let $k \geq 0$, then a k -*planning graph* for a planning problem $\langle \mathcal{F}, \mathcal{A}, Ops, I, G \rangle$ is a directed acyclic graph $G = \langle N_f, N_a, \xrightarrow{pre}, \xrightarrow{add}, \xrightarrow{del}, \oplus \rangle$ where N_f is a time-indexed family of sets of *fluent nodes*, i.e. $N_f = N_f^0 \cup \dots \cup N_f^k$ where N_f^i is the set of fluent nodes of *layer* i ; N_a is a time-indexed family of sets of *action nodes*, i.e. $N_a = N_a^0 \cup \dots \cup N_a^{k-1}$ where N_a^i is the set of action nodes of *layer* i ; \xrightarrow{pre} is a time-indexed relation between fluent nodes and action nodes, i.e. $\xrightarrow{pre}^i \subseteq N_a^i \times N_f^i$, whose instances are called *preconditions edges*; \xrightarrow{add} and \xrightarrow{del} are time-indexed binary relations between action nodes and fluent nodes, i.e. $\xrightarrow{add}^i \subseteq N_a^i \times N_f^{i+1}$ and $\xrightarrow{del}^i \subseteq N_a^i \times N_f^{i+1}$, whose instances are called *add edges* and *delete edges* respectively; finally $\oplus = \bigcup_{0 \leq i < k} \oplus_f^i \cup \bigcup_{0 \leq j < k-1} \oplus_a^j$ is a time-indexed (commutative) relation of mutual exclusion (mutex for short) between nodes, i.e. $\oplus_f^i \subseteq N_f^i \times N_f^i$ and $\oplus_a^j \subseteq N_a^j \times N_a^j$.

More in detail, the k -planning graph associated to a planning problem $\Pi = \langle \mathcal{F}, \mathcal{A}, Ops, I, G \rangle$ is inductively defined as follows:

¹¹ Graphplan was the first planner due to Blum and Furst that uses the planning graph data structure. So when we talk about Graphplan we mean the algorithm defined in [7] that works on the planning graph data structure following a paradigm called Planning Graph Analysis.

¹² See [25] for a survey.

1. $f \in N_f^0$ iff $f \in I$ and $f \in N_f^i$ iff there exists $a \in N_a^{i-1}$ such that $a \xrightarrow{add}^{i-1} f$ for $i = 1, \dots, k$;
2. $a \in N_a^i$ iff $(Pre \xrightarrow{a} Add ; Del) \in Ops$, $Pre \subseteq N_f^i$, and for all fluents $f, f' \in Pre$ not $f \oplus^i f'$ for $i = 0, \dots, k-1$;
3. $a \xrightarrow{pre}^i f$ iff $(Pre \cup \{f\} \xrightarrow{a} Add ; Del) \in Ops$ for $i = 0, \dots, k-1$;
4. $a \xrightarrow{add}^i f$ iff $(Pre \xrightarrow{a} Add \cup \{f\}; Del) \in Ops$ for $i = 0, \dots, k-1$;
5. $a \xrightarrow{del}^i f$ iff $(Pre \xrightarrow{a} Add ; Del \cup \{f\}) \in Ops$ for $i = 0, \dots, k-1$;
6. $a \oplus_a^i a'$ iff $a \neq a'$ and (i) either there exists $f \in \mathcal{F}$ s.t. $a \xrightarrow{del}^i f$, and $a' \xrightarrow{add}^i f$ or $a' \xrightarrow{pre}^i f$, (ii) or there exist $f, f' \in \mathcal{F}$ s.t. $a \xrightarrow{pre}^i f$, $a' \xrightarrow{pre}^i f'$, and $f \oplus_f^i f'$ for $i = 0, \dots, k-1$;
7. $f \oplus_f^i f'$ iff $f \neq f'$ and for all $a \in \mathcal{A}$ such that $a \xrightarrow{add}^{i-1} f$ and for all $a' \in \mathcal{A}$ such that $a' \xrightarrow{add}^{i-1} f'$ we have $a \oplus_a^{i-1} a'$ for $i = 1, \dots, k$.

Our Graphplan-based algorithm alternates two phases: *graph expansion* and *solution extraction* which are iterated until a necessary (but not sufficient) condition for plan existence is achieved: there exists a subset S of N_f^k that corresponds to a state represented by G and there are no pairs $f_1, f_2 \in S$ such that $f_1 \oplus_f^k f_2$. During the graph expansion phase, the planning graph is extended by appending one action layer, which contains all actions that are applicable, and a fluent layer which is the union of the effects of the newly added action layer and the last fluent layer. The algorithm also calculates the mutex constraints for each newly added *action* and *fluent* layers.

Notice that, we also introduces “nop” actions, which provide the links between the identical fluents appearing in consecutive layers. The resulting planning graph is a compact way to represent an over-approximation of the forward search tree (i.e. the tree representing the forward reachable state space starting from the initial state). This planning graph is built in polynomial time with respect to the size of the corresponding planning problem. (See [7] for details.)

Once the planning graph expansion phase is completed, the algorithm carries out the solution extraction phase. This phase produces the equivalent SAT encoding of planning graph. This translation from the planning graph to SAT is due to Kautz and Selman in [18]. By equivalent we mean that the propositional formula Φ_H^n is such that any model of Φ_H^n corresponds to a partial-order plan of length n .

The encoding of a planning problem into SAT via the planning graph analysis of the planning problem is of the form :

$$\Phi_H^n = \iota(\mathbf{f}^0) \wedge \bigwedge_{i=0}^{n-1} \tau_i(\mathbf{f}^i, \boldsymbol{\alpha}^i, \mathbf{f}^{i+1}) \wedge \gamma(\mathbf{f}^n)$$

where \mathbf{f} and $\boldsymbol{\alpha}$ are vectors of the fluents and actions in \mathcal{F} and \mathcal{A} respectively and

- $\iota(\mathbf{f}^0)$ is a formula encoding the initial state and is a conjunction of the formulae f^0 if $f \in N_f^0$;
- $\gamma(\mathbf{f}^n)$ is a formula encoding the final states built out of G . Without loss of generality we assume that the formula G is in disjunctive normal form. The formula $\gamma(\mathbf{f}^n)$ is built from G by removing the disjunctions containing fluents not occurring in N_f^n .
- $\tau_i(\mathbf{f}^i, \boldsymbol{\alpha}^i, \mathbf{f}^{i+1})$ is a formula encoding the transition relation at step i and is a conjunction of the *Action Preconditions Add-Effects Axioms*:

$$\alpha^i \supset \bigwedge \{f^i \mid \alpha \xrightarrow[pre]{i} f\} \quad \alpha^i \supset \bigwedge \{f^{i+1} \mid \alpha \xrightarrow[add]{i} f\}$$

for all α in N_a^i , the *Backward Chaining Axioms*:

$$f^{i+1} \supset \bigvee \{\alpha^i \mid \alpha \xrightarrow[add]{i} f\}$$

for all f in N_f^{i+1} , and the *Mutex Axioms*: $\neg(a^i \wedge b^i)$ for all a and b such that $a \oplus b$.

The results of our experiments are reported in Table 2. For each protocol we give the smallest value of n at which the attack is found (**N**), the number of propositional variables (**A**) and clauses (**CL**) in the SAT formula, the time spent to generate the SAT formulae (**EncT**) and the total time spent by the SAT solver to solve the SAT formulae (**SolT**).

The columns of the table headed by $\oplus = \oplus_a \cup \oplus_f$ contain the results of experiments that we carried out with our implementation of the Graphplan algorithm. Results show that the algorithm performs well on a large set of examples, however, for bigger ones (see, e.g., *KLS rep.* protocol) the time spent at generating the formula (**EncT**) gets high. This observation led us at investigating a new direction based on the computation of a weaker version of the mutex relation.

An interesting point shown in [11] is that, on average, the computation of the mutex relation between fluent nodes is time consuming. In our first implementation of the Graphplan algorithm mutexes were computed following clauses 6 and 7. We implemented a second version in which we restrict the computation on action nodes only (i.e. $\oplus = \oplus_a$). If we examine the inductive definition of \oplus_a (clause 6) we see that only case (i) is relevant since $\oplus_f = \emptyset$. We call this particular subset of the mutex relation between nodes *static mutexes* (as in [11]) noted $\hat{\oplus}$ and we call *dynamic mutexes* the set $\tilde{\oplus} = \oplus \setminus \hat{\oplus}$. This latter set is made of the mutexes between fluent node (clause 7) and mutexes between actions nodes defined by the case (ii) in clause 6.

Table 2 shows that when we restrict the computation of \oplus to $\hat{\oplus}$ (columns headed by $\oplus = \hat{\oplus}$) we have a gain up to two orders of magnitude in the time of the planning graph analysis. See for instance the *KLS rep.* protocol.

By considering a weaker version of the mutex relation, we obtain a rougher over-approximation of the forward search tree since the more mutexes we have, the more accurate is the over-approximation. On the other side, when we compute all mutexes, we get either a less or equal number of atoms and a greater

Table 2. Experimental data using the Graphplan-based encoding

Protocol	N	$\oplus = \oplus_a \cup \oplus_f$				$\oplus = \hat{\oplus}$			
		A	CL	EncT	SolT	A	CL	EncT	SolT
<i>Andrew</i>	9	699	4,459	17.0	0.0	701	1,811	1.5	0.0
<i>EKE</i>	5	545	3,584	10.9	0.0	553	1,648	1.5	0.0
<i>ISO-CCF-1 U</i>	4	154	621	0.2	0.0	154	398	0.1	0.0
<i>ISO-CCF-2 M</i>	4	169	600	0.3	0.0	169	402	0.2	0.0
<i>ISO-PK-1 U</i>	4	202	794	0.5	0.0	202	494	0.2	0.0
<i>ISO-PK-2 M</i>	4	196	707	0.6	0.0	196	468	0.2	0.0
<i>ISO-SK-1 U</i>	4	142	561	0.2	0.0	142	367	0.1	0.0
<i>ISO-SK-2 M</i>	4	234	747	0.5	0.0	234	511	0.2	0.0
<i>KaoChow 1</i>	7	526	4,055	14.6	0.0	583	2,230	1.5	0.01
<i>KaoChow 2</i>	9	1,020	10,325	108.1	0.02	1,136	4,507	4.2	0.03
<i>KaoChow 3</i>	9	1,229	18,532	419.1	0.01	1,467	7,927	8.8	0.05
<i>KLS rep.</i>	7	2,018	62,836	3,557.4	0.03	2,092	31,510	23.5	0.07
<i>NSCK</i>	9	622	3,823	9.5	0.01	627	1,710	1.3	0.01
<i>NSPK</i>	7	559	3,129	6.7	0.0	572	1,555	1.0	0.0
<i>NSPK-server</i>	8	1,077	6,043	24.5	0.0	1,079	2,951	3.9	0.0
<i>SPLICE</i>	9	972	7,545	52.2	0.01	1,008	2,953	3.8	0.0
<i>Swick 1</i>	5	329	1,300	1.1	0.0	329	783	0.4	0.01
<i>Swick 2</i>	6	438	1,937	2.6	0.0	438	1,143	0.1	0.0
<i>Swick 3</i>	4	231	889	0.7	0.0	231	594	0.3	0.0
<i>Swick 4</i>	5	405	1,469	2.7	0.0	405	939	0.6	0.0
<i>Stubblebine rep</i>	3	220	791	0.7	0.0	222	608	0.3	0.0
<i>Woo-Lam M</i>	6	665	3,806	17.3	0.0	687	1,813	2.9	0.0

or equal number of clauses. This means that the over-approximation is more accurate. However experimental results show that this gain in accuracy on the over-approximation does not pay off (see, e.g., the *KaoChow 3* protocol).

We have also considered a third implementation of the algorithm characterized by the fact that we do not build the mutex relation and hence we do not generate any “Mutex Axioms” during the solution extraction phase. Similarly to the linear encoding, we apply an *Abstraction Refinement Loop* that checks if the pseudo partial-order plan found can be linearized (and hence executed). SATMC therefore looks for conflicting actions¹³ in the pseudo partial-order plan found and extends the previously generated formula with clauses negating the conflicts (if any). The resulting formula is then fed back to the SAT-solver and the whole procedure is iterated until a solution without conflicts is met or the formula becomes unsatisfiable. However applying the Abstraction Refinement Loop to the Graphplan encoding, we do not have the same gain as in the linear

¹³ Differently from the linear encoding, two distinct actions α_1, α_2 are said to be in conflict iff $\alpha_1 \hat{\oplus} \alpha_2$

encoding case. Also, there are no significant improvement in performance when compared to the second version of the algorithm.

The comparison between the linear and the Graphplan-based encoding techniques is summarized in Figures 1 and 2. For each encoding techniques and for each protocol those figures show (in logarithmic scale) the size of the SAT formulae (number of atoms and clauses), and the encoding time, respectively. Notice that, concerning the linear encoding technique we plot the results obtained by applying the Abstraction Refinement Loop (see Section 4.1), while concerning the Graphplan-based encoding technique we report the results obtained by generating only the weaker version of the mutex relation. The comparison shows that, by using the Graphplan-based encoding technique, we obtain (i) SAT instances whose size is up to 2 orders of magnitude smaller and, in most of the cases, (ii) better encoding and solving times. This enables SATMC to analyze protocols (e.g., the *KLS rep.* protocol), that were not feasible using the linear approach. However, there are situations in which the use of the linear encoding technique is preferable to the use of the Graphplan-based encoding. For instance, in the domain of security protocols we are interested in analyzing scenarios in which a bounded number of concurrent sessions is specified without any constraint on which agent will play a particular role in a session. Therefore, instead of a single initial state, in these situations we have to deal with a set of possible initial states. By using the linear encoding it is sufficient to define $\iota(\mathbf{f}^0)$ to be a formula whose models are all the possible initial states. The application of the Graphplan-based encoding to the above scenario requires the analysis of a sequence of different problems (one for each possible initial state) and thus much of the efficiency of the approach is lost.

5 Related Work

The idea of regarding security protocols as planning problems is not new. In [1], it has been proposed an executable planning specification language \mathcal{AL}_{SP} for representing security protocols and checking the possibility of attacks via a model finder for logic programs with stable model semantics. Compared to this approach SATMC performs better (on the available results) and can readily exploit improvements of state-of-the-art SAT solvers.

Preliminary results obtained by running the SAT technology planning system blackbox [19] against our benchmark of protocols, indicate that even if the performance of the tools are comparable, the PDDL language used by blackbox (and by state-of-the-art planners) to specify planning problems is not appropriate to express in a compact way security protocols. This is due to the fact that PDDL is unable to specify complex term structure (only individual constants are allowed). Hence, in order to translate our compact protocol specifications into PDDL, time consuming operations such as flattening and grounding of fluents and operators must be executed.

Gavin Lowe and his group at the University of Leicester (UK) have analyzed problems from the Clark/Jacob library [10] using Casper/FDR2 [13]. This ap-

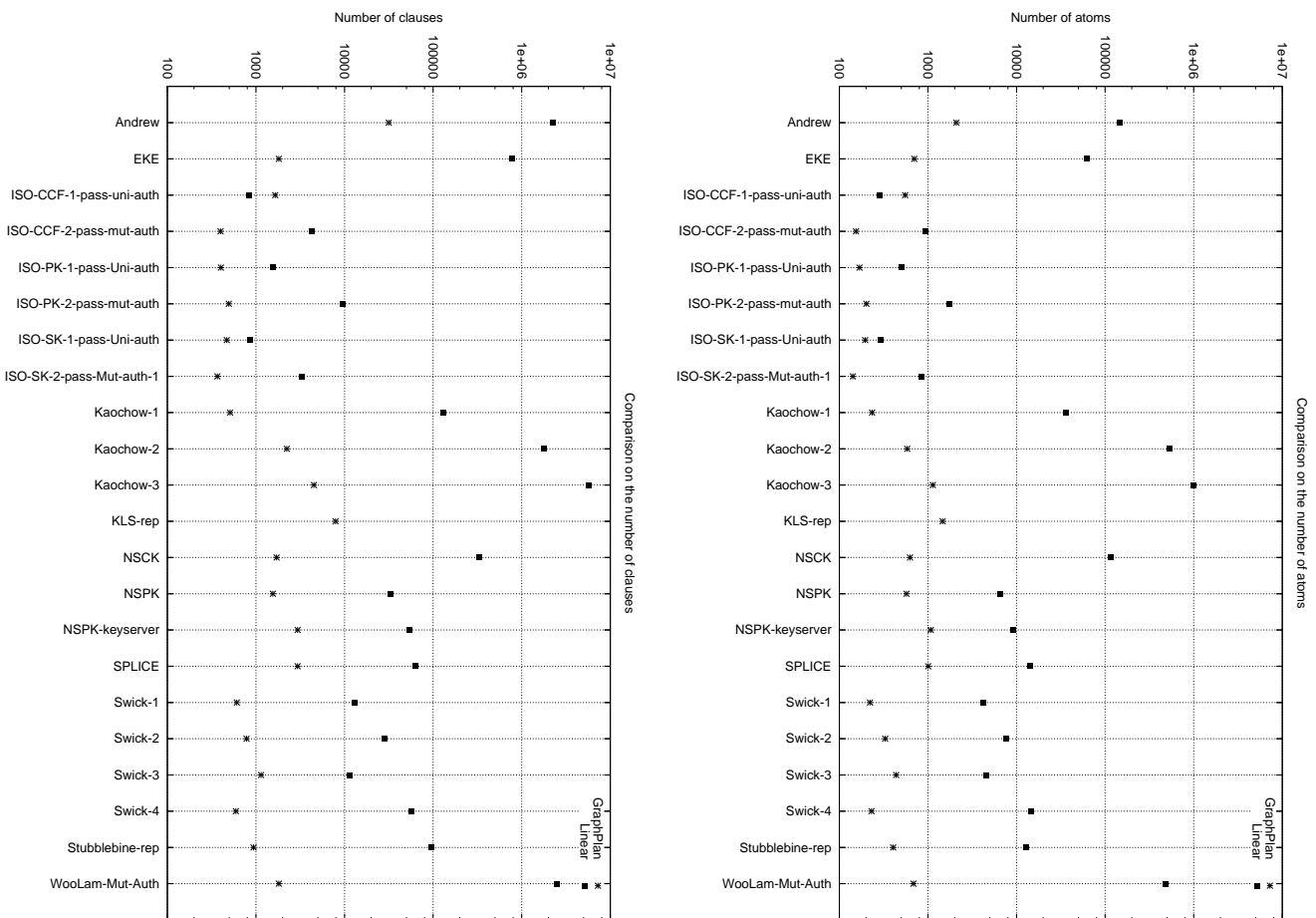


Fig. 1. Comparison between Graphplan-based and linear encodings on the number of atoms and clauses

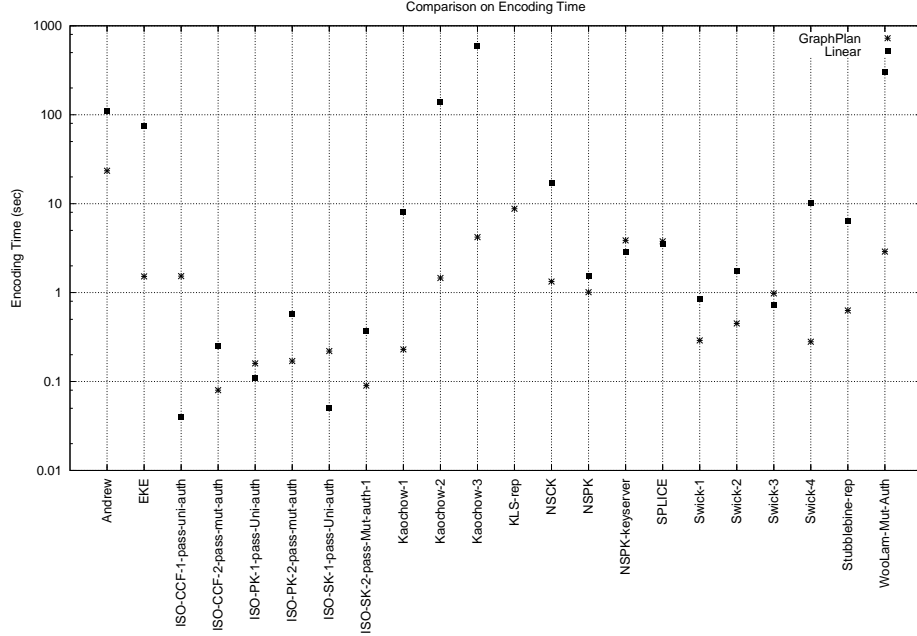


Fig. 2. Comparison between Graphplan-based and linear encodings on the encoding time

proach has been very successful for discovering new flaws in protocols. However, first experiments on the search time indicate that SATMC is more effective than Casper/FDR2. Besides that Casper/FDR2 limits the size of messages that are sent in the network and is not able to handle non-atomic keys.

The Murphi model-checker has been used in [21] for analyzing some cryptographic protocols such as the Needham-Schroeder Public-Key (NSPK). Experimental results indicate that Murphi suffers from state-space explosion. To cope with this problem several restrictions on the model are put in place. For instance, the size of the channel is fixed a priori.

The Athena approach [24] combines model checking and theorem proving techniques with the strand space model to reduce the search space and automatically prove the correctness of or find an attack on security protocols (when it terminates). However, Athena does not support non-atomic keys, which is a real drawback for analyzing e-commerce protocols. A comparison in term of performance is left for the future work.

We conclude this overview of relevant related work by mentioning that SATMC is one of the back-ends of the AVISS tool and that in the context of that work we have performed a thorough comparison between the back-ends integrated in

it.¹⁴ The On-the-Fly Model-Checker (OFMC) [5] performs uniformly well on all the Clark/Jacob library. However, it is interesting to observe that in many cases the time spent by the SAT solver is equal to or smaller than the time spent by OFMC for the same protocol. The Constraint-Logic-based model-checker (CL) [9] is able to find type-flaw attacks (as well as OFMC). However, the overall timing of SATMC is better than that of CL. Detailed results about these experiments can be found in [2].

6 Conclusions and Perspectives

We have enhanced our SAT-based Model-Checker, SATMC, with a sophisticated SAT reduction based on the Graphplan-based encoding technique. Experimental results obtained by running SATMC against a selection of security protocols drawn from the Clark-Jacob's library clearly show that the Graphplan-based encoding is superior to the linear encoding. This is due to the fact that the Graphplan-based encoding reduces dramatically both the size of the SAT instances and the time spent to generate them. As a consequence SATMC can now solve problems that can not be tackled with the linear encoding (e.g. the *KLS rep.* protocol). However, as pointed out in Section 4, when applied to protocol insecurity problems with multiple initial states the linear encoding can be preferable to the Graphplan-based encoding. This kind of scenarios will be subject of further investigations.

The results obtained confirm the effectiveness of the SAT-based approach for the analysis of security protocols and pave the way to its application to large protocols which arise in practical applications.

References

1. L. Carlucci Aiello and F. Massacci. Verifying security protocols as planning in logic programming. *ACM Trans. on Computational Logic*, 2(4):542–580, October 2001.
2. A. Armando, D. Basin, M. Bouallagui, Y. Chevalier, L. Compagna, S. Modersheim, 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. A. Armando and L. Compagna. Automatic SAT-Compilation of Protocol Insecurity Problems via Reduction to Planning. In *22nd IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE)*, Houston, Texas, November 2002. Also presented at the FCS & Verify Workshops, Copenhagen, Denmark, July 2002.
4. A. Armando and L. Compagna. Abstraction-driven SAT-based Analysis of Security Protocols. Submitted to the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT 2003). A preliminary version of the paper has been already accepted and presented to SAT 2003 conference, May 2003.

¹⁴ Notice that the results shown in this comparison and concerning SATMC have been obtained by applying only the linear encoding technique.

5. David Basin, Sebastian Moedersheim, and Luca Viganò. An On-the-Fly Model-Checker for security protocol analysis. Forthcoming, 2003.
6. Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In W. R. Cleaveland, editor, *Tools and Algorithms for the Construction and Analysis of Systems. Part of European Conferences on Theory and Practice of Software, ETAPS'99, Amsterdam*, volume 1579 of *LNCS*, pages 193–207. Springer-Verlag, 1999.
7. Avrim Blum and Merrick Furst. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, pages 1636–1642, 1995.
8. Ilario Cervesato, N. A. Durgin, Patrick Lincoln, John C. Mitchell, and Andre Scedrov. A meta-notation for protocol analysis. In *CSFW*, pages 55–69, 1999.
9. Y. Chevalier and L. Vigneron. A Tool for Lazy Verification of Security Protocols. In *Proceedings of the Automated Software Engineering Conference (ASE'01)*. IEEE Computer Society Press, 2001. Long version available as Technical Report A01-R-140, LORIA, Nancy (France).
10. 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>.
11. Minh Binh Do, Biplav Srivastava, and Subbarao Kambhampati. Investigating the effect of relevance and reachability constraints on SAT encodings of planning. In *Artificial Intelligence Planning Systems*, pages 308–314, 2000.
12. Danny Dolev and Andrew Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.
13. 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.
14. 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.
15. E. Giunchiglia, M. Maratea, A. Tacchella, and D. Zambonin. Evaluating search heuristics and optimization techniques in propositional satisfiability. In *Proceedings of IJCAR'2001*, LNAI 2083, pages 347–363. Springer-Verlag, Heidelberg, 2001.
16. Heather, Lowe, and Schneider. How to prevent type flaw attacks on security protocols. In *PCSF: Proceedings of The 13th Computer Security Foundations Workshop*. IEEE Computer Society Press, 2000.
17. 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.
18. Henry Kautz, David McAllester, and Bart Selman. Encoding plans in propositional logic. In *KR'96: Principles of Knowledge Representation and Reasoning*, pages 374–384. Morgan Kaufmann, San Francisco, California, 1996.
19. Henry A. Kautz and Bart Selman. Unifying SAT-based and graph-based planning. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 318–325, Stockholm, Sweden, July 31-August 6 1999. Morgan Kaufmann.
20. G. Lowe. Breaking and fixing the Needham-Shroeder public-key protocol using FDR. In *Proceedings of TACAS'96*, pages 147–166. Springer-Verlag, 1996.

21. J.C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murphi. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 141–153. 1997.
22. 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.
23. R. M. (Roger Michael) Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. Technical Report CSL-78-4, Xerox Palo Alto Research Center, Palo Alto, CA, USA, 1978. Reprinted June 1982.
24. 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.
25. Daniel S. Weld. Recent advances in AI planning. *AI Magazine*, 20(2):93–123, 1999.
26. H. Zhang. SATO: An efficient propositional prover. In William McCune, editor, *Proceedings of CADE 14*, LNAI 1249, pages 272–275. Springer-Verlag, Heidelberg, 1997.