Security protocols analyzed symbolically*

Michele Boreale
Dipartimento di Sistemi e Informatica, Università di Firenze
e-mail: boreale@dsi.unifi.it

Abstract

In the field of security protocol analysis, a class of automated methods relies upon the use of symbolic techniques. We illustrate this approach by focusing on one such method. We outline the underlying protocol model, the concept of symbolic execution and the resulting verification method. We then discuss the benefits of the symbolic approach when contrasted with traditional methods baded on finite-state model-checking.

1 Introduction

Security protocols are by now an essential ingredient of communication infrastructures. When executed in a hostile environment, these protocols are subject to a variety of attacks, that can compromise the security of the data being exchanged over the communication network. As a result of an attack, an attacker might typically learn a piece of information which is supposed to remain secret, or it might fool an agent into accepting a compromised key as authentic. Proving a protocol resistant to such attacks is notoriously a difficult task. In the last decade, a lot of research effort has been directed towards automatic analysis of crypto-protocols.

The existing automatic methods all stem from a conceptual model due to Dolev and Yao [10]. In this model, the communcation network is under control of a powerful *adversary*. As explained in the next section, in order to fool the honest participants the adversary can inject traffic on the network at its will. This feature makes the operational model of the system infinite-state, even if restricted to a limited number of protocol sessions. This can be regarded as a state explosion problem specific to crypto-protocols.

Finite-state verification methods ([15, 17]) deals with approximate, finite models, for which a well established model-checking technology is at hand. Recently, infinite-state approaches, based on a variety of symbolic techniques ([2, 4, 9, 13, 18]), have emerged. These methods seem to be very promising in two respects. First, at least if the number of protocol sessions is bounded, they can accomplish a complete exploration of the state space: thus they provide *proofs or disproofs* of correctness - under the Dolev-Yao assumptions. Second, symbolic data representation allows for compact models, thus improving on time and memory efficiency of verification.

The approach outlined here, *symbolic trace analysis* [4], belongs to the class of infinite-state methods. Specifically, this approach is based on: (a) representing protocols as a concurrent processes, described using a dialect of the spi-calculus [1], and (b) analysing the execution traces generated by the resulting system. Central to the method is a concept of *symbolic execution* based on unification, which makes it possible to get round the state-explosion problem.

Symbolic trace analysis has been implemented as part of a prototype verification tool, STA (*Symbolic Trace Analyzer*), which is available from the author's web page [5].

In the rest of the paper, we will first discuss the model underlying the verification method (Section 2), then touch upon the problem of state-explosion and introduce symbolic execution (Section 3). Based on the latter, we explain the verification method (Section 4). We then discuss the benefits of symbolic methods when compared to traditional model checking (Section 5). We conclude with a few remarks and comparison with related work (Section 6).

*This work has been partially supported by EU within the FET - Global Computing initiative, project MIKADO IST-2001-32222 and by MIUR project NAPOLI. The funding bodies are not responsible for any use that might be made of the results presented here.

2 Overview of the model

The model underlying trace analysis can be traced back to Dolev and Yao [10]. As hinted in the Introduction, agents executing the protocol communicate through a public network that is under the control of an adversary. The adversary records all messages that transit over the network, and can generate messages by either replaying old ones, or by combining the latters (e.g. by pairing, encryption and decryption), and/or by generating fresh quantities. Sending a message on the network amounts to handing the message to the adversary. Conversely, receiving a message from the network amounts to non-deterministically accepting any message among those the adversary can generate.

Formally, a state of the system is a pair $s \triangleright P$, called *configuration*: s is a string of I/O actions, and represents the current adversary's knowledge; P is (essentially) a spi-term [1], describing the intended behavior of honest participants. For the sake of presentation, we concentrate here on a language supporting shared-key encryption only. We write $\{M\}_k$ the ciphertext obtained by encrypting the cleartext M with the key k: only knowing this key it is possible to decrypt $\{M\}_k$ and retrieve the cleartext M. Thus the set of messages is:

$$M,\, N \quad ::= \quad x \;\mid\; a \;\mid\; \langle M, N\rangle \;\mid\; \{M\}_k$$

where $x, y, \ldots$ range over a set of variables, and $a, \ldots k, \ldots$ range over an set of names (both sets are assumed to be countable). The syntax of processes is given below:

$P,\, Q$	$::=$		
		$\mathbf{0}$	$(null)$
	$\mid$	$\mathsf{a}(x).\, P$	$(input\ prefix)$
	$\mid$	$\overline{\mathsf{a}}\langle M\rangle.\, P$	$(output\ prefix)$
	$\mid$	$[M = N]P$	$(match)$
	$\mid$	$\mathsf{case}\ M\ \mathsf{of}\ \{y\}_k\ \mathsf{in}\ P$	$(decryption)$
	$\mid$	$\mathsf{pair}\ M\ \mathsf{of}\ \langle x, y\rangle\ \mathsf{in}\ P$	$(splitting)$
	$\mid$	$(\nu\, b)P$	$(restriction)$
	$\mid$	$P \mid Q$	$(parallel\ composition).$

Since we seek for automatic methods we have omitted replication, which would make the problem undecidable (see e.g. [11]). Also, note that labels $\mathsf{a}, \mathsf{b}, \ldots$ are just tags attached to I/O action for ease of reference (as mentioned, the model assumes one public network). The set of all configurations is denoted by $\mathbf{C}$. The dynamics of configurations is given by a transition relation $\longrightarrow \subseteq \mathbf{C} \times \mathbf{C}$, that describes elementary steps of computations. Below, we report the rules defining this transition relation (we omit a few details concerning substitutions, which are not essential here).

$$(\textsc{Inp}) \qquad s \triangleright \mathsf{a}(x).\, P \;\longrightarrow\; s \cdot \mathsf{a}\langle M\rangle \triangleright P[{}^{M}\!/_{x}] \quad \text{where } s \vdash M$$

$$(\textsc{Out}) \qquad s \triangleright \overline{\mathsf{a}}\langle M\rangle.\, P \;\longrightarrow\; s \cdot \overline{\mathsf{a}}\langle M\rangle \triangleright P$$

$$(\textsc{Case}) \qquad s \triangleright \mathsf{case}\ \{M\}_k\ \mathsf{of}\ \{y\}_k\ \mathsf{in}\ P \;\longrightarrow\; s \triangleright P[{}^{M}\!/_{y}]$$

$$(\textsc{Split}) \qquad s \triangleright \mathsf{pair}\ \langle M, N\rangle\ \mathsf{of}\ \langle x, y\rangle\ \mathsf{in}\ P \;\longrightarrow\; s \triangleright P[{}^{M}\!/_{x},\ {}^{N}\!/_{y}]$$

$$(\textsc{Match}) \qquad s \triangleright [M = M]P \;\longrightarrow\; s \triangleright P$$

$$(\textsc{Res}) \qquad s \triangleright (\nu\, a)P \;\longrightarrow\; s[{}^{a'}\!/_{a}] \triangleright P[{}^{a'}\!/_{a}] \quad \text{where } a' \text{ is fresh for } s,\ P$$

$$(\textsc{Par}) \qquad \frac{s \triangleright P \;\longrightarrow\; s' \triangleright P'}{s \triangleright P \mid Q \;\longrightarrow\; s' \triangleright P' \mid Q}$$

plus symmetric version of (PAR).

Some comments are in order. Rules (INP) and (OUT) concern sending and receiving messages, respectively. Since sending a message just means handing the message to the adversary, any output action $\overline{\mathsf{a}}\langle M\rangle$ fired by a process is recorded in the adversary's current knowledge s (rule (OUT)). Conversely, receiving a message just means accepting any message among those the adversary can produce. Therefore, in rule (INP) the variable x can be replaced by any message M non-deterministically chosen among those the adversary can synthesize from its current knowledge s. The synthesis of a message M from a set of known messages S is formalized by a deduction relation $\vdash$. Here is a sample of the deduction rules defining $\vdash$ (see [4]):

$$\frac{M \in S}{S \vdash M} \qquad \frac{S \vdash M \qquad S \vdash k}{S \vdash \{M\}_k}$$

$$\frac{S \vdash \{M\}_k \qquad S \vdash k}{S \vdash M}\,.$$

The remaining operational rules govern how a process decrypts a message ($\mathsf{case}\, M\, \mathsf{of}\, \{y\}_k\, \mathsf{in}\, A$), splits a pair ($\mathsf{pair}\, \langle M, N\rangle\, \mathsf{of}\, \langle x, y\rangle\, \mathsf{in}\, A$), compares two messages for equality ($[M = N]A$), handles a new name ($(\nu\, a)P$) and interleaves execution of parallel threads ($A \mid B$).

It is worthwhile to point out that, in this formalization, there is no need for an explicit description of the adversary's behavior, as the latter is wholly determined by its current knowledge – the s in $s \triangleright P$ – and by the deduction relation $\vdash$. This is somehow in contrast with other proposals [15, 17], where the adversary must be explicitly described.

3 Symbolic execution

When synthesizing a message, the adversary can apply operations like pairing, encryption and generation of fresh names repeatdly. Thus, at any time, the set of messages the adversary can synthesize is actually infinite. Any such message can be non-deterministically sent to a participant willing to receive it; therefore every model based on Dolev and Yao's is in principle infinite, more precisely infinite-branching. Our basic model makes no exception: in rule (INP) the set of M's s.t. $s \vdash M$ is always infinite, and this makes the model infinitely-branching. In other words, message exchange induces a form of state explosion.

To overcome this problem, the STA tool implements a verification method based on a notion of symbolic execution. A new transition relation (written $\longrightarrow_{\mathrm{s}}$, below) is introduced in order to condense the infinitely many transitions that arise from an input action (rule (INP) of operational semantics) into a single, *symbolic* transition. The received message is now represented simply by a free variable. The set of values this variable may take on is constrained as the execution proceeds. Technically, a constraint takes the form of *most general unifier* (mgu), i.e., a most general substitution that makes two expressions equal. The set of traces generated using the symbolic transition relation constitutes the *symbolic model* of the protocol. It is worth noting that, differently from the standard model given by $\longrightarrow$, the symbolic model is finite, because each input action just gives rise to one symbolic transition, and agents cannot loop.

For a flavor of how symbolic execution works, let us consider the following example. Suppose that agent P, after receiving a message, tries decrypting this message using key k; if decryption succeeds and y is the result, the agent checks whether y equals b and, if so, proceeds like P'. This is written as

$$P \stackrel{\text{def}}{=} \mathsf{a}(x).\, \mathsf{case}\, x\, \mathsf{of}\, \{y\}_k\, \mathsf{in}\, [y = b]P'.$$

Let us explain how the symbolic execution proceeds, starting from the initial configuration $\sigma \triangleright P$, where $\sigma = \overline{\mathsf{lost}}\langle b,\, k\rangle$ (i.e. the adversary has somehow got to know names b and k). After the first input step, in the second step the decryption $\mathsf{case}\, x\, \mathsf{of}\, \{y\}_k\, \mathsf{in}\, \cdots$ is resolved by unifying x and $\{y\}_k$: this results in the unifier (substitution) $[\{y\}_k/x]$, which is propagated globally. In the third step, the equality test $[y = b]$ is in turn resolved by unifying y and b, that results in $[b/y]$, again propagated globally. Formally,

$$\begin{array}{rcl} \sigma \triangleright P & \longrightarrow_{\mathrm{s}} & \sigma \cdot \mathsf{a}\langle x\rangle \;\triangleright\; \mathsf{case}\, x\, \mathsf{of}\, \{y\}_k\, \mathsf{in}\, [y = b]P' \\ & \longrightarrow_{\mathrm{s}} & \sigma \cdot \mathsf{a}\langle \{y\}_k\rangle \;\triangleright\; [y = b]P'[\{y\}_k/x] \\ & \longrightarrow_{\mathrm{s}} & \sigma \cdot \mathsf{a}\langle \{b\}_k\rangle \;\triangleright\; P'[\{y\}_k/x][b/y]. \end{array}$$

An important point is that symbolic execution actually *ignores* the deduction relation $\vdash$, and thus there is not an exact correspondence between concrete and symbolic traces. In fact, every concrete trace *is* an instance of some symbolic

trace, but the converse is not true in general. There may be 'inconsistent' symbolic traces, that do not correspond to any concrete trace. For example, consider again the process P above, but this time let us start with the *empty* trace ϵ. In other words, let us consider $\epsilon \triangleright P$. The trace $\mathsf{a}\langle\{b\}_k\rangle$, which is symbolically generated by this configuration, is not consistent. To see why, consider that the adversary simply cannot generate the message $\{b\}_k$ out of an empty knowledge (the trace ϵ): i.e. $\epsilon \not\vdash \{b\}_k$.

The good news is that inconsistent symbolic traces can be detected (and discarded). We have a procedure that, when given a symbolic trace as input, 'refines' it until a form is reached by which consistency can be checked syntactically. The refinement procedure is further discussed in the next section.

4 Property verification

Given a configuration $s \triangleright P$ and a trace s', we say that $s \triangleright P$ *generates* s' if $s \triangleright P \longrightarrow^* s' \triangleright P'$ for some P' ($\longrightarrow^*$ is the reflexive and transitive closure of $\longrightarrow$, i.e. zero or more steps of $\longrightarrow$). We express properties of the protocol in terms of the traces it generates. In particular, we focus on correspondence assertions of the kind

> for every generated trace, if action β occurs in the trace, then action α must have occurred at some previous point in the trace

that is concisely written as $\alpha \hookleftarrow \beta$. More accurately, we allow α and β to contain free variables, that may be instantiated to ground values. Thus $\alpha \hookleftarrow \beta$ actually means that *every instance* of β must be preceded by the corresponding instance of α, for every generated trace. We write $s \triangleright P \models \alpha \hookleftarrow \beta$ if the configuration $s \triangleright P$ satisfies this requirement. This kind of assertions is flexible enough to express interesting secrecy and authentication properties. As an example, the final step of many key-establishment protocols consists in A's sending a message of the form $\{N\}_k$ to B, where N is some authentication information, and k the newly established key. A typical property one wants to verify is that any message encrypted with k that is accepted by B at the final step should actually originate from A (this ensures B he is really talking to A, and that k is authentic). If we call $\mathsf{final_A}$ and $\mathsf{final_B}$ the labels attached to A's and B's final action, respectively, then the property might be expressed by $\overline{\mathsf{final_A}}\langle\{x\}_k\rangle \hookleftarrow \mathsf{final_B}\langle\{x\}_k\rangle$, for x a variable. The scheme also permits expressing secrecy as a reachability property (in the style of [2, 12]): to this end, it is convenient to fix an 'absurd' action $\perp$ which is supposed to be never executed and consider the property $\perp \hookleftarrow \beta$, that is "no instance of action β is ever executed". It is possible to let action β correspond to the adversary's getting a secret. Thus $\perp \hookleftarrow \beta$ precisely says that the adversary will never get this secret.

Let $\alpha \hookleftarrow \beta$ be a property and $\mathcal{C}$ a configuration. Denote by $\mathbf{Mod}_{\mathcal{C}}$ the symbolic model generated starting from $\mathcal{C}$. The verification method $\mathbf{M}(\mathcal{C}, \alpha \hookleftarrow \beta)$, presented in Table 1, checks whether $\mathcal{C}$ satisfies $\alpha \hookleftarrow \beta$ or not. Moreover, if the property is not satisfied, $\mathbf{M}(\mathcal{C}, \alpha \hookleftarrow \beta)$ computes a trace violating the property, that is, an attack on $\mathcal{C}$. The method relies on the refinement procedure $\mathbf{Refinement}(\cdot)$, that takes any symbolic trace σ and yields the most general instances of σ that are syntactically consistent, if any (details on how to compute this set can be found in [4, 5]).

The functioning of the method is best explained in the case $\alpha = \perp$. This means verifying that in the *concrete* semantics, no instance of action β is ever executed starting from $\mathcal{C}$. By a correspondence theorem between symbolic and concrete semantics, this amounts to checking that for each symbolic trace σ in the symbolic model, no ground instance of σ contains an instance of action β. To check this, the method proceeds as follows. First, , for every action γ in σ, it checks whether there is a mgu θ of γ and β. If no such σ, γ and θ exist, then clearly no instance of β is ever executed, thus the property holds true. If they exist, but $\sigma\theta$ is not consistent (i.e. the check $\exists\, \sigma' \in \mathbf{Refinement}(\sigma\theta)$ at step 5 fails), then again the property holds true. Otherwise there is a consitent symbolic trace containing an instance of β, thus the property does not hold true: the trace σ' violating the property is reported.

The verification method based on symbolic execution is sound and complete w.r.t. the standard model, in the sense that every consistent attack detected in the symbolic model (relation $\longrightarrow_s$) corresponds to some attack in the standard model (relation $\longrightarrow$), and vice-versa (see [4, 5]). In other words, the symbolic model captures all and only the attacks of the standard model.

5 Discussion

We summarize what we think are the main benefits of the symbolic approach when compared to traditional finite-state methods. The discussion takes two aspects into account, model's accuracy and efficiency.

$\mathbf{M}(\mathcal{C}, \alpha \leftrightarrowtail \beta)$

1. compute $\mathbf{Mod}_{\mathcal{C}}$ from the symbolic operational semantics;
2. **foreach** $\sigma \in \mathbf{Mod}_{\mathcal{C}}$ **do**
3. **foreach** action γ in σ **do**
4. **if** $\exists\, \theta = \mathrm{mgu}(\beta, \gamma)$ **and**
5. $\exists\, \sigma' \in \mathbf{Refinement}(\sigma\theta)$ **where** $\sigma' = \sigma\theta\theta'$ **and**
6. $\alpha\theta\theta'$ does not occur prior to $\beta\theta\theta'$ in σ'
7. **then return**(No, σ');
8. **return**(Yes);

Table 1: The verification method

Accuracy of the model In finite-state methods (e.g. [15, 17]), analysis of security protocols is carried out by modelling both honest participants *and* the adversary as communicating processes, and then putting them in parallel. An operational model for the resulting system is then explicitly generated and model-checked. In order to keep the model finite and rely on standard model checking, a bound on the number of possible messages the intruder can generate is fixed. In order to control state-explosion, and given the combinatorics of message-generation, the chosen bound must necessarily be low (no more than, say, one dozen). This is sometimes achieved by imposing restrictions on the type of messages the adversary can generate at any stage.

Differently from finite-state model checking, the symbolic approach makes no assumption on the type and number of messages the adversary can generate: STA performs a complete exploration of the whole infinite-state model. For instance, the method detects "type-dependent" attacks. In this kind of attacks, the adversary cheats on the type of some messages, e.g. by inserting a nonce where a key is expected according to the protocol description. Type-dependent attacks usually escape finite-state analysis (see e.g. [16]). Under certain circumstances, it may be sensible to assume that the attacker obey a given type discipline (e.g. because non well-typed message can be easily detected and discarded), but in general, an appropriate typing cannot be established automatically: it seems that some (fairly accurate) knowledge is required of how the protocol works is required. This makes the whole analysis process potentially error-prone.

Efficiency Symbolic trace analysis does not suffer from any state explosion problem depending on message exchange: any input action in the protocol gives rise to *a single* transition in the symbolic model. In finite-state model checkers, clever assumptions may reduce the state space to explore, but input actions give rise to a branching factor greater than one: then, as the number of participants, and/or of possible data values, increases, the size of the model is expected to increase dramatically. In view of these considerations, symbolic methods seem to have some advantage over finite-state model checkers. For example, this is true for those instances of Needham-Schroeder and Kerberos analyzed with STA in [6] and with Murφ in [17]. Memory occupation can be very well controlled in tools based on symbolic methods, because they naturally lend themselves to a depth first search strategy.

Note that the word 'efficiency' should be taken here in a practical sense, not in the formal sense of 'worst-case complexity'. In fact, the problem of protocol analysis is NP-hard even in its simplest forms (see e.g. [20]).

6 Conclusions and related work

We have illustrated a unification-based method for the analysis of security protocols. In contrast with finite-state model checking, the method can analyze the whole infinite state space generated by a limited number of sessions. The method is efficient in practice, because the symbolic model is compact, and the refinement procedure at its heart is only invoked on demand and on single symbolic traces.

Early work on infinite-state analysis is due to Huima. In [13], the execution of a protocol generates a set of equational constraints. Only an informal description is provided of the kind of equational rewriting needed to solve these constraints. Roughly contemporary to ours is the approach of Amadio *et al.* [2, 3]. Unlike our approach, symbolic execution and consistency check are not kept separate, and this may have a relevant impact on the size of the computed symbolic model. Another point worth noting is that, in [2, 3], a brute-force method is used to resolve variables in key position: all possible instantiations for these variables are in fact considered. The decision technique in [9] is based on a reduction to a set constraint problem which is in turn reduced to an automata-theoretic problem. Cycling processes are allowed, but completeness is proven by assuming rather severe restrictions on protocol syntax.

The technique in [18] focuses on reachability properties and is based on constraint solving. Symbolic reduction and knowledge analysis are kept separate; the latter is performed by a constraints solving procedure.

A very recent trend focuses on the detection of attacks that also exploit features of low-level operations, such as modular exponentiation. The 'perfect encryption' assumption of Dolev and Yao is thus partially relaxed, but the resulting models are in general less amenable to automatic analysis. A step in this direction is [6], which introduces a framework for 'generic' crypto-primitives (subject to certain conditions). An application of this framework to the case of Diffie-Hellman key exchange is in [7]. There, an automated method is given, under the assumption that the adversary can only use a limited set of capabilities. Diffie-Hellman exponentiation is also the subject of a few very recent papers [8, 14, 19], mostly based on re-write techniques, that present (un)decidability results for several flavours of the problem.

References

[1] M. Abadi, A.D. Gordon. A Calculus for Cryptographic Protocols: the spi-calculus. *Information and Computation*, 148(1):1-70, 1999.

[2] R.M. Amadio, S. Lugiez. On the Reachability Problem in Cryptographic Protocols. In *Proc. of Concur'00*, *LNCS* 1877, Springer, 2000.

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

[4] M. Boreale. Symbolic Trace Analysis of Cryptographic Protocols. *ICALP'01*, *LNCS* 2076, pp.667-681, Springer-Verlag, 2001.

[5] M. Boreale, M.G. Buscemi. Experimenting with STA, a Tool for Automatic Analysis of Security Protocols. *ACM Symposium on Applied Computing 2002*, ACM Press, 2002.

[6] M. Boreale, M.G. Buscemi. A framework for the analysis of security protocols. *CONCUR'02*, *LNCS* 2421, Springer, 2002.

[7] M. Boreale, M.G. Buscemi. Symbolic analysis of crypto-protocols based on modular exponentiation. *MFCS'03*, *LNCS* 2747, Springer, 2003.

[8] Y. Chevalier, R. Küsters, M. Rusnowitch, M. Turuani. Deciding the security of protocols with Diffie-Hellman exponentiation and products in exponents. Bericht Nr. 0305, Institut für Informatik und Praktische Mathematik, C-A Universität, Kiel, June 2003.

[9] H. Comon, V. Cortier, J. Mitchell. Tree Automata with One Memory, Set Constraints and Ping-pong Protocols. *ICALP'01*, *LNCS* 2076, pp.682-693, Springer-Verlag, 2001.

[10] D. Dolev, A. Yao. On the Security of Public-key Protocols. *IEEE Transactions on Information Theory*, 2(29):198-208, 1983.

[11] N. Durgin, P. Lincoln, J. Mitchell, A. Scedrov. Undecidability of Bounded Security Protocols. In *Proc. of FLOC Workshop on Formal Methods and Security Protocols*, Trento, 1999.

[12] J. Goubault-Larrecq. A Method for Automatic Cryptographic Protocol Verification. *Proc. 15th IPDPS Workshops*, *LNCS* 1800, pages 977-984, Springer 2000.

[13] A. Huima. Efficient Infinite-State Analysis of Security Protocols. In *Proc. of FLOC Workshop on Formal Methods and Security Protocols*, Trento, 1999.

[14] D. Kapur, P. Narendran, L. Wang. Analyzing protocols that use modular exponentiation: Semantic unification techniques. In *Proc. of RTA 2003*, 2003.

[15] G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. *TACAS'96*, *Proceedings* (T. Margaria, B. Steffen, Eds.), *LNCS* 1055, pp. 147-166, Springer-Verlag, 1996.

[16] G. Lowe. A Hierarchy of Authentication Specifications. In *Proc. of 10th IEEE Computer Security Foundations Workshop*, IEEE Computer Society Press, 1997.

[17] J.C. Mitchell, M. Mitchell, U. Stern. Automated Analysis of Cryptographic Protocols Using Murφ. In *Proc. of Symp. Security and Privacy*, IEEE Computer Society Press, 1997.

[18] J.Millen, M. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. of the 8th ACM Conference on Computer and Communications Security*, ACM Press, 2001.

[19] J.Millen, M. Shmatikov. Symbolic protocol analysis with products and Diffie-Hellman exponentiation. In *Proc. of the 16th IEEE CSFW*, 2003.

[20] M. Rusinowitch, M Turuani. Protocol Insecurity with Finite Number of Sessions in NP-Complete. In *14th Computer Security Foundations Workshop*, IEEE Computer Society Press, 2001.