Workshop on Formal Methods and Computer Security

Thursday, July 20, 2000

Chicago

Workshop Program

9:00 - 10:30: Session I: Keynote Address

- D. Tygar (UC Berkeley)

10:30-10:45: Break

10:45 - 12:15: Session II

- "Formal Verification of Non-Repudiation Protocols – A Game Approach"
  S. Kremer, J.-F. Raskin (Universite Libre de Bruxelles, Belgium)
- "On The Security of Authenticated Key-Exchange Protocols"
  R. Canetti (IBM T.J.Watson Research Center),
  H. Krawczyk (Technion, Israel, and IBM T.J.Watson Research Center)
- "A Comparison and Combination of Theory Generation and Model Checking for Security Protocol Analysis"
  N. Hopper, S. Seshia, J. Wing (Carnegie Mellon University)

12:15-1:30: Lunch

1:30-2:45: Session III

- "A CAPSL Connector to Athena",
  J. Millen (SRI International)
- "Design of a CIL Connector to Maude",
  G. Denker (SRI International)
- "Local Secrecy for State-Based Models",
  H. Ruess (SRI International), J. Millen (SRI International)

2:45-3:15: Break

3:30-4:45: Session IV

- "A Brutus Model Checking of a Spi-Calculus Dialect"
  S. Gnesi (CNUCE-CNR, Pisa, Italy), D. Latella, (IEI-CNR, Pisa, Italy),
  G. Lenzini (CNUCE-CNR, Pisa, Italy)
- "Categorizing Attacks on Cryptographic Protocols Based on Intruder's Roles and Behaviour"
  C. Xu (Duke University), G. Kedem (Duke University),
  F. Gong (Advanced Networking Research, MCNC)
- "Verification Method for Possibility of Parallel Attack on Multiple Sessions"
  K. Negishi (Hitachi and Tokyo Insitute of Technology, Japan),
  N. Yonezaki (Tokyo Institute of Technology, Japan)
- "Interpreting Strands in Linear Logic"
  I. Cervesato (ITT Industries), N. Durgin (Stanford Unversity),
  M. Kanovich (University of Pennsylvania), A. Scedrov (University of Pennsylvania)

A "Brutus" model checking of a spi-calculus dialect Extended Abstract

S. Gnesi[†] D. Latella [*] G. Lenzini [†]

June 23, 2000

1 Introduction

Recently there has been a wide interest in applying formal methods to specify and verify cryptographic protocols (see for example [2, 7, 4, 16, 19, 23, 22, 25, 10, 17]). These approaches range from the use of a process calculus to model cryptographic protocols and using equivalence relations to prove security properties on them, to the use of a general or special purpose model checkers. In this paper we propose a model checking approach for verifying security properties for cryptographic protocols expressed into a process calculus derived from the spi-calculus [2], a cryptographic version of the π-calculus [20, 21].

The semantics of the spi-calculus is usually given via labeled transition systems, while secrecy and integrity properties can be expressed via may-testing and barbed equivalences [3]; unfortunately this approach is not immediately suitable for automatic verification. Indeed, Abadi and Gordon defined in [1] a bisimulation, called frame bisimulation, which is sound respect to testing equivalence, but that requires several levels of quantification over infinite domains making any automatic verification impossible. Later works tried to reduce quantifications in order to move towards the design of verification tools. In [12] an alternative notion of framed bisimilarity, called fenced bisimilarity, has been defined in such a way that completeness and soundness with respect to frame bisimilarity are preserved. Although one

[*]CNUCE-CNR, Pisa (Italy). (e-mail: {d.latella}@cnuce.pi.cnr.it)
[†]IEI-CNR, Pisa (Italy). (e-mail: {gnesi,lenzini}@iei.pi.cnr.it)

of the original quantifications was eliminated, this result is not not sufficient to guarantee the development of a fully automatic verification. In [5] a trace equivalence and weak bisimulation semantics have been defined over an environment-sensitive labeled transition system. The former preserves may-testing while the latter preserves barbed equivalence. This approach avoids any quantification over contexts, but to our knowledge, no verification tool has been developed yet on these semantics.

In this paper we propose a preliminary framework in which protocols, expressed in a dialect of the spi-calculus, can be verified using model checking algorithms [8]. In particular we define a formal semantics for a dialect of the spi-calculus based on labeled transition systems in such a way that the model checking environment developed by Clarke, Marrero and Jha [10], can be re-used. Recently this environment has been extended with both a first order linear temporal logic [9], used to specify security properties, and partial order reduction techniques [11]. These two new results make this approach interesting and effective for automatic verification.

In addition, while defining the semantics of our spi-calculus dialect we introduce a definition of knowledge that makes our labeled transition system finite branching on input actions. In fact, one of the most difficult issues regarding finite-state behavior of spi-calculus is related to the possibility of receiving messages drawn from an infinite set of data. In this paper we present a first attempt for coping with this problem. We want to point out that the use of finite knowledge notion it still under study and actually it can be applied only under the hypothesis that any honest agent involved in a protocol knows, a priori, the *structure* of all the messages it receives. This constraint holds for many security protocols proposed in the literature.

The rest of the paper is organized as follows. In Section 2 we summarize the assumptions under which we have developed our study, in Section 3 we describe the syntax of our spi-calculus dialect, in Section 4 we define the notion of knowledge used in the language semantics which is defined in Section 5. Finally in Section 6 we introduce the main intuitions explaining how model checking can be performed using the "Brutus" model checker of Clarke, Jha and Marrero. In Section 7 we conclude pointing out some future work.

2 Assumptions

Generally, when reasoning about correctness of cryptographic protocols one assumes the presence of a malicious external entity, called *attacker*. As

some authors have already proposed (see for example [24, 18, 5, 14]) rather than trying to describe the behavior of the attacker it is much simpler to assume that the communication network is totally under the control of the *environment*. Under this assumption the environment can store, duplicate, hide or replace messages passing through the network; in addition according to the rules followed by honest agents, the environment can syntethize new messages by encryption or by pairing, or it can analyze compound messages by decryption using the relative decryption key, or by splitting tuples of sub-messages.

Although many different environment capabilities can be defined, the above ones informally depict the *most powerful* environment. As Focardi and Martinelli in [14] formally proved, satisfying properties in presence of the most powerful environment is a sufficient condition for satisfying them in presence of less powerful attackers.

In addition we suppose that no private channels between processes exist and then: (1) whenever a process performs an output action the environment performs an input action; (2) whenever a process performs an input action the environment performs an output action; (3) internal process actions do not require interactions with the environment.

Finally we work under the Perfect Encryption Assumption [18], the aim of which is to keep separated security issues regarding the protocol itself, from the robustness of the cryptosystem used to encrypt/decrypt the messages.

3 Language Specification

In this section we describe our spi-calculus dialect. The syntax of the language is given in Table 1. We assume given: a set $\mathcal{L}$ of *channel labels* [1]; a set $\mathcal{K}$ of *atomic keys*; a set $\mathcal{N}$ of *atomic names*; a set $\mathcal{V}$ of *variables*. We let a range over channel labels, k range over atomic keys, n range over atomic names, m range over atomic names and keys when no distinction is required, and x range over variables. Atomic names and keys are used to build *messages*, via encryption and pairing. We also suppose the presence of *message terms* built using variables or messages still via encryption and pairing. In messages and message terms we intend to maintain the structure explicitly visible. This will be used to perform efficient matching operations.

[1] Although there are no private channels, we want to distinguish different public channels. Also if not explained in this paper, that is related to the definition of authentication properties in the logic supported by Brutus.

A *protocol* is a parallel composition of agents. *Agents* syntax is a modified version of spi-calculus [2]. In particular it includes a null process, **0**; the generation of a new atomic name or key, $(\text{new } m)P$; an input action on channel a, $a\langle T\rangle$, that allows the receiving of a message M whose structure exactly matches the structure of the message term T. An output action, $\overline{a}\langle T\rangle$ allows the sending of a message M over a channel a; in this case the message M is obtained from the term T by instantiating each variable with atomic names or keys. An equality test on messages, $[S{=}T]$, can be put as guard on actions. Also in this case the matching is to be intended guided by the structure of terms. Finally a fork action, fork P, allows an agent to create a copy of itself, useful to describe those protocols composed by multiple runs.

Since we want to avoid any infinite state behavior, we could assume syntactical restrictions on the use of parallelism and recursion (see for example [13]), in such a way to obtain *finite* control systems. In any case the potential of infinite behavior is of no practical impact since it will result in all resources exhausted.

$M, N ::=$	messages $\mathcal{M}$
$\quad m \mid \{M\}_k \mid \langle M, N\rangle$	
$S, T ::=$	terms $\mathcal{T}$
$\quad x \mid \{S\}_x \mid \langle S, T\rangle \mid M$	
$P, Q ::=$	processes $\mathcal{P}$
$\quad \mathbf{0}$	nil
$\quad \mid a\langle T\rangle \cdot P$	input
$\quad \mid \overline{a}\langle T\rangle \cdot P$	output
$\quad \mid P \parallel Q$	parallel composition
$\quad \mid$ fork P	fork
$\quad \mid (\text{new } m)P$	new name (m is bound in P)
$\quad \mid [S{=}T]P$	match
$\quad \mid p$	agent identifier
$A ::=$	process definitions
$\quad p \stackrel{def}{=} P$	defining equation

The definition of free/bound names and variables are defined as usual.

Table 1: Syntax specification.

To illustrate how a cryptographic protocol can be described in our lan-

guage, let us consider a very simple example, where two agents A and B share keys K_{as} and K_{sb} with an authentication server S. This protocol can be informally described with the following sequence of actions:

$$\begin{array}{lll} 1. & A & \rightarrow S : \{K_{ab}\}_{K_{as}} \\ 2. & S & \rightarrow B : \{K_{ab}\}_{K_{sb}} \\ 3. & A & \rightarrow B : \{M\}_{K_{ab}} \end{array}$$

where with $A \rightarrow B : M$ we indicate that A sends an atomic message M to B and respectively that B receives the message M from A. We can express this protocol in our language as follows:

$$\begin{array}{rcl} A & \stackrel{def}{=} & (\text{new } K_{ab}).(\text{new } M)(\overline{c_{as}}\langle\{K_{ab}\}_{K_{as}}\rangle.\overline{c_{ab}}\langle\{M\}_{K_{ab}}\rangle).A' \\ B & \stackrel{def}{=} & c_{sb}(\{y\}_{K_{sb}}).c_{ab}(\{x\}_y).B' \\ S & \stackrel{def}{=} & c_{as}(\{x\}_{K_{as}}).\overline{c_{sb}}\langle\{x\}_{K_{sb}}\rangle.S' \\ \mathit{Simple} & \stackrel{def}{=} & (\text{new } K_{as}).(\text{new } K_{sb})(A \parallel S \parallel B) \end{array}$$

In order to deal with the structure of messages and message terms, we introduce the notion of *depth*.

Definition 1 *Let M be a message. The* depth *of M, $depth_{\mathcal{M}}(M)$, is defined on the structure of M as follows:*

$$\begin{array}{lcl} 1.\ depth_{\mathcal{M}}(m) & = & 1 \\ 2.\ depth_{\mathcal{M}}(\{M\}_k) & = & depth_{\mathcal{M}}(M) + 1 \\ 3.\ depth_{\mathcal{M}}(\langle M, N\rangle) & = & depth_{\mathcal{M}}(M) + depth_{\mathcal{M}}(N). \end{array}$$

Let T be a message term. The depth *of T, $depth_{\mathcal{T}}(T)$, is defined on the structure of T as follows:*

$$\begin{array}{lcl} 1'.\ depth_{\mathcal{T}}(x) & = & 1 \\ 2'.\ depth_{\mathcal{T}}(\{S\}_x) & = & depth_{\mathcal{T}}(S) + 1 \\ 3'.\ depth_{\mathcal{T}}(\langle S, T\rangle) & = & depth_{\mathcal{T}}(S) + depth_{\mathcal{T}}(T) \\ 4'.\ depth_{\mathcal{T}}(M) & = & depth_{\mathcal{M}}(M). \end{array}$$

In the following we shall write $depth()$ the particular function meant being determined by the context.

Shrinking rules

$$\frac{\{M\}_k \quad k}{M}\ (\vdash_{dec}) \qquad \frac{\langle M, N\rangle}{M}\ (\vdash_{fst}) \qquad \frac{\langle M, N\rangle}{N}\ (\vdash_{snd})$$

Expanding rules

$$\frac{M \quad k}{\{M\}_k}\ (\vdash_{cry}) \qquad \frac{M \quad N}{\langle M, N\rangle}\ (\vdash_{pair})$$

Table 2: Inference system for message manipulation.

4 Environment Knowledge

In studying cryptographic protocols it is quite important to define a precise concept of *knowledge*. By knowledge, it is intended the amount of information the environment, or any other agent running the protocol, can generate at a given moment starting from a set W of messages. The set W, we call *basic knowledge*, represents: (a) for the environment the set of messages really transmitted through the network; (b) for an agent the set of messages it has received till a given moment. In an initial state, W can be empty or initialized with a set of messages supposed to be known a priori.

In order to model how messages can be handled, we use the inference system given in Table 2: we say that a message M is deducible from a set W of messages, and we write $W \vdash M$, if there exists a proof of M to have its premises contained in W. The inference system defined encodes all the operations one can perform on messages. In particular a message M can be encrypted with a key k obtaining $\{M\}_k$ (rule $\vdash_{cry}$); a message of the form $\{M\}_k$ can be decrypted if the corresponding decryption key k is known[2] (rule $\vdash_{dec}$); two messages M and N can be combined to form a pair $\langle M, N\rangle$; the messages composing a pair $\langle M, N\rangle$ can be extracted (rules $\vdash_{fst}$ and $\vdash_{snd}$).

The inference systems is used to characterize three different concepts of knowledge:

Definition 2 *Let $W \subset \mathcal{M}$ be a finite set of messages. Then:*

1. *$\mathcal{A}(W)$, the* analysis *of W, is the set of messages W and the ones that can be inferred starting from W using only shrinking rules;*

[2]Here, without loss of generality, we suppose that the inverse key k^{-1} needed to decrypt the message $\{M\}_k$, is equal to k itself. In other words we suppose that the cryptosystem used to crypt/decrypt messages is symmetric.

2. $\mathcal{S}(W)$, *the* synthesis *of* W, *is the set of messages* $\mathcal{A}(W)$ *and the messages that can be inferred using also expanding rules;*

3. $\mathcal{S}^d(W)$, *the* d-synthesis *of* W *where* d *is an positive integer constant, is the set of messages* $\mathcal{A}(W)$ *and the messages that can be inferred using also expanding rules to obtain only messages whose* depth *is at most* d, *that is:*

$$\mathcal{S}^d(W) = \{M \in \mathcal{M} : W \vdash M,\ depth_{\mathcal{M}}(M) \leq d\}$$

We can notice that, given a finite W the sets $\mathcal{A}(W)$ and $\mathcal{S}^d(W)$ are finite sets of messages, while $\mathcal{S}(W)$ is an infinite set of messages.

5 Semantics

In this section we define the semantics of our language. This semantics defined over labeled transition systems, is very close the one used by Clarke, Jha and Marrero in [10]. The main difference is that we obtain finite branching on input transitions by making use of the bounded notion of *d-synthesis*.

The semantics describes how the overall *global state*, consisting of the asynchronous composition of named communicating processes and the environment, is updated by actions. Each instance of an agent involved in the protocol is modeled as one of these named processes each augmented with a *local state*. A local state consists of the name of the agent, an unique identifier of the instance, the set of basic messages received by the instance, and a set of bindings. The unique identifier will be useful in case of multiple instances of the same agent, usually introduced to model multiple runs of a protocol. Formally:

Definition 3 *A* local state I, *of an agent* P_i *is a tuple* $(P_i, ID_i, W_i, \sigma_i, \mathcal{P})$, *where:*

- N_i *is the name of the agent;*
- ID_i *is the unique identifier of the agent;*
- W_i *is the set of messages received by the agent. We will use* W_i *to retrieve the knowledge of the agent via the functions defined in Section 4.*
- $\sigma_i : \mathcal{V} \longrightarrow \mathcal{N} \cup \mathcal{K}$ *is the the function that represents the set of bindings for the variables appearing in the the process specification of the agent* N_i. *We allow a variable to be bounded only to atomic names or keys.*

- $\mathcal{P}$ *is the process specification, defining the continuation of its behavior.*

Definition 4 *A* global state $\mathcal{G}$, *is a tuple* $(W_\Omega, I_1, I_2, \ldots, I_n)$ *where:*

- $W_\Omega \subset \mathcal{M}$ *is the set of messages fetched by the environment (i.e. the ones passed through the network). In addition we suppose that in a start state,* W_Ω *could be empty or initialized with messages supposed to be known, a priori, by the environment;*

- $\mathcal{I}_i$ *is the local state of a honest agent instance running the protocol, for* $i = 1, \ldots, n$.

All the transitions are labeled with actions. An *action*, α, can be: (a) an output action $ID_i.\overline{a}\langle M\rangle$, (b) an input action $ID_i.a(M)$, or (c) an internal action $ID_i.\tau$, where ID_i is the identifier of the agent performing the action, a is the label indicating the public channel used, and M is the message send or received. Each possible execution of the model corresponds to a *trace*, which is a finite, alternating sequence of global states and actions. Formally a trace t, is such that $t = \mathcal{G}_0 \cdot \alpha_1 \cdot \mathcal{G}_1 \ldots \alpha_n \cdot \mathcal{G}_n$ for some positive integer n such that $\mathcal{G}_{i-1} \xrightarrow{\alpha_i} \mathcal{G}_i$ for $0 < i \leq n$ and for the transition relation defined below.

In the following we only report the most interesting transitions:

(INP) $$(W_\Omega, I_1, \ldots, (P_i, ID_i, W_i, \sigma_i, a\langle T\rangle.P), \ldots, I_n)$$

$$\xrightarrow{ID_i.a\langle M\rangle} (W_\Omega, I_1, \ldots, (P_i, ID_i, W_i', \sigma_i', P), \ldots, I_n)$$

where: $W_i' = W_i \cup \{M\}$, for $M \in \mathcal{S}^d(W_\Omega)$ with $d = depth(T)$, and σ_i' is the minimal[3] the set of binding extending σ_i, such that $T\sigma_i' = M$;[4]

In the input action an agent receives a message from the d-synthesis knowledge of the environment, $\mathcal{S}^d(W_\Omega)$. In particular in order that the receive action is successful there should exist a binding σ_i' extending σ_i, (i.e., $\sigma_i \subseteq \sigma_i'$), which unifies the message term with the input message.

Notice that finitess of $\mathcal{S}^d(W_\Omega)$ implies finite branching on input actions. This surely is a quite reductive assumption but we want to underline that: (a) it can be directly applied to many real protocols and in this case for those protocols we obtain a finite state transition systems; (b) for other critical

[3]Minimal is to be intended respect to the following partial order relation: $\sigma \subseteq \sigma'$ iff $\forall x$ such that $\sigma(x)$ is defined, then $\sigma(x) = \sigma'(x)$.

[4]With $T\sigma$ we indicate the message obtained substituting all the free variable x in T, with $\sigma(x)$.

protocols - like BAN-Yahalom [6] where an agent A forwards a message M without knowing the sub-messages component it, we are studying the possibility to define some simplifying function preserving security properties, as done by Hui and Lowe in [15].

(OUT) $$(W_\Omega, I_1, \ldots, (P_i, ID_i, W_i, \sigma_i, \overline{a}\langle T\rangle.P), \ldots, I_n)$$

$$\xrightarrow{ID_i.\overline{a}\langle M\rangle} (W'_\Omega, I_1, \ldots, (P_i, ID_i, W_i, \sigma_i, P), \ldots, I_n)$$

where: $W'_\Omega = W_\Omega \cup \{M\}$.

In the output action an agent sends a message $M = T\sigma_i$. According to our premises, the message is fetched by the environment, and consequently recorded in W_Ω.

(PAR) $$(W_\Omega, I_1, \ldots, (P_i, ID_i, W_i, \sigma_i, P \parallel Q), \ldots, I_n)$$

$$\xrightarrow{ID_i.\tau} (W_\Omega, I_1, \ldots, I_{i-1}, I_{i+1}, \ldots I_n, I_{n+1}, I_{n+2})$$

where: (a) $\mathcal{I}_{n+1} = (P_{n+1}, ID_{n+1}, W_{n+1}, \sigma_{n+1}, P)$, and P_{n+1} is a new name, ID_{n+1} is a new agent identifier, $W_{n+1} = W_i$, and $\sigma' = \emptyset$; (b) $\mathcal{I}_{n+2} = (P_{n+2}, ID_{n+2}, W_{n+2}, \sigma_{n+2}, Q)$, where P_{n+2} is a new name, ID_{n+2} is a new agent identifier, $W_{n+2} = W_i$, and $\sigma' = \emptyset$.

In case of parallel composition, two new instances of a process are created[5]. Their local states are such that a new agent names and a new agent identifiers are created for each agent, the set of basic knowledge is inherited from the father, and the set of binding is reset. All other items in the global environment remain unchanged.

(FORK) $$(W_\Omega, I_1, \ldots, (P_i, ID_i, W_i, \sigma_i, \text{fork } P)$$

$$\xrightarrow{ID_i.\tau} (W_\Omega, I_1, \ldots, (P_i, ID_i, W_i, \sigma_i, P), \ldots I_n, I_{n+1})$$

where: $\mathcal{I}_{n+1} = (P_i, ID_{n+1}, W_{n+1}, \sigma_{n+1}, P)$, and ID_{n+1} is a new agent identifier, $W_{n+1} = W_i$, and $\sigma' = \sigma_i$.

In case of fork, we simply create a copy of the agent, running in parallel with the forking agent. In this case the local state of the process differs from the father only for the identifier.

[5]Indeed we think of the process $P \parallel Q$ to be substituted by the two processes P and Q.

(NEW) $(W_\Omega, I_1, \ldots, (P_i, ID_i, W_i, \sigma_i, (\text{new } m).P)$

$$\xrightarrow{ID_i.\tau} (W_\Omega, I_1, \ldots, (P_i, ID_i, W_i', \sigma_i, P), \ldots I_n)$$

where: $W_i' = W_i \cup \{m\}$.

Whenever a new name m is generated by an agent, the new name is added to the set of basic messages owned by the agent.

6 Model checking for spi-calculus dialects

In this section we briefly explain how to perform model checking of a protocol specified in the dialect of the spi-calculus defined in this paper. It is our intention to resort to the model checking framework developed by Clarke *et. al.* in [10, 9, 11]. In the above mentioned works the authors use a special purpose language to specify a security protocol. The related formal semantics is based on a labeled transition system which is quite close the one we presented here. Indeed it is easy to prove, by structural induction on transitions and on trace length, that it is possible to define a mapping preserving knowledge information, from our semantics to the one defined in [9].

Natural consequence of this reduction is that all the results, from properties specification to the use of the model checker tool "Brutus", can be re-used to perform model checking of protocols specified in our spi-calculus dialect. A positive consequence of this can be the availability of an effective automatic verification framework for the class of spi-calculus-like languages, which have been generally recognized as widely expressive in describing security protocols.

Since in Brutus there exists a first order temporal logic to specify security and authenticity properties, we are actually working to analyze the relationship between the equivalence induced by that logic and the equivalences already defined on spi-calculus via different approaches.

7 Conclusion and Future Work

In this work, we have defined a preliminary framework in which to perform model checking of the spi-calculus. Actually no model checking of the spi-calculus has been proposed in the literature although fundamental steps in defining security and integrity properties as trace equivalences, have been

studied. That lack of automatic tools for the spi-calculus motivated us to re-use a defined model checking environment as the quickest solution.

The most urgent future work would be to prove sound and/or complete results between the relation induced by the built-in login in Brutus and the may-testing, or barbed equivalence defined on the spi-calculus to prove integrity and secrecy properties. As mentioned in the previous sections we are still study on this.

Finally it would be interesting to extend the use of the finite of knowledge notion presented in this paper also for those protocols that actually seems not directly translatable in our formalism.

We want thank to M. Boreale and R. Pugliese for stimulating discussion on basic issues of this work.

References

[1] Martín Abadi and Andrew D. Gordon. A Bisimulation Methods for Cryptographic Protocols. In *Proc. of ESOP'98*, 1998.

[2] Martín Abadi and Andrew D. Gordon. A Calculus for Cryptographic Protocols. The Spi Calculus. Technical Report 149, Digital Equipment Corporation Systems Research Center, Palo Alto, California, 1998.

[3] Martín Abadi and Andrew D. Gordon. Reasoning about the Cryptographic Protocols in the Spi Calculus. In *Proc. of CONCUR'98*, 1998.

[4] Dominique Bolignano. An Approach to the Formal Verification of Security Protocols. In *Proc. of 3rd ACM Conference on Computer and Communication Security*, 1996.

[5] Michele Boreale, Rocco De Nicola, and Rosario Pugliese. Proof Techiques of Crytpographic Protocols. In *Proc. of LICS*, 1999.

[6] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. In *Proc. of the Royal Society of London*, volume 426 of *Lecture Notes in Computer Science*, pages 233–271. Springer-Verlag, 1989.

[7] Michael Burrows, Martín Abadi, and Roger Needham. A Logic of Authentication. *ACM Transaction on Computer System*, 8(1):18–36, 1990.

[8] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specification.

ACM Transaction on Programming Languages and Systems, 8(2):244–263, 1986.

[9] E. M. Clarke, S. Jha, and W. Marrero. A machine checkable logic of knowledge for protocols. In *Proc. of Workshop on Formal Methods and Security Protocols*, 1998.

[10] E. M. Clarke, S. Jha, and W. Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *Proc. of IFIP Working Conference on Programming Concepts and Methods (PROCOMET)*, 1998.

[11] E. M. Clarke, S. Jha, and W. Marrero. Partial order reductions for security protocol verification. In *Proc. of TACAS 2000*, 2000.

[12] A. S. Elkjaer, M. Höhle, H. Hütter, and K. O. Nielsen. Towards automatic bisimularity checking in the spi-calculus. In *Proc. of DMTCS'99+CATS'99*, 1999.

[13] Alessandro Fantechi and Stefania Gnesi. How much Expressive are LOTOS Behaviour Expressions? In *Proc. of FORTE90*, volume 431, 1990.

[14] Riccardo Focardi and Fabio Martinelli. A Uniform Approch for the Definition of Security Properties. In *Proc. FM'99*, volume 1708 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[15] M. L. Hui and G. Lowe. Safe Symplifying Transformation for Security Protocols. In *Proc. of the 12th Computer Security Foundations Workshop*, 1999.

[16] Audun Josang. Security Protocol Verification using SPIN. In *Proc. of the 1th Workshop on Automata Theoretic Verification with the SPIN Model Checker — SPIN95*, 1995.

[17] G. Lowe. Towards a completeness result for model checking for security protocols. In *Proc. of the 11th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1998.

[18] Will Marrero, Edmund Clarke, and Somesh Jha. Model Checking for Security Protocols. In *Proc. of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[19] Catherine A. Meadows. Formal Verification of Cryptographic Protocols: a Survey. *Lecture Notes in Computer Science*, 917:133–149, 1995.

[20] Robin Milner, Joachim Parrow, and David Walker. A Calculus of Mobile Processes, I. *Information and Computation*, 100(1):1–40, September 1992.

[21] Robin Milner, Joachim Parrow, and David Walker. A Calculus of Mobile Processes, II. *Information and Computation*, 100(1):41–77, September 1992.

[22] J. C. Mitchell, M. Mitchell, and U. Stern. Automated Analysis of Cryptographic Protocols Using Murϕ. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 141–151, 1997.

[23] Lawrence C. Paulson. Proving Properties of Security Protocols by Induction. Technical Report 409, Computer Laboratory, University of Cambridge, 1996.

[24] Steve Schneider. Security properties and CSP. In *Proc. of the IEEE Symposium on Research in Security and Privacy*, pages 174–187, 1996.

[25] Steve Schneider. Verifying Authentication Protocols in CSP. *IEEE Transaction on Sofware Engineering*, 24(8):743–758, 1998.