An Analysis of the Needham-Schroeder Public-Key Protocol with Membranes

Olivier Michel[1] and Florent Jacquemard[2]

[1] LaMI CNRS umr 8042 – Université d'Évry,
Tour Évry-2, 523 place des terrasses de l'agora, 91000 Évry France.
email: `michel@lami.univ-evry.fr`

[2] INRIA FUTURS and LSV, CNRS umr 8643 – ENS de Cachan,
61 avenue du Président Wilson, 94235 Cachan Cedex, France.
email: `florent.jacquemard@lsv.ens-cachan.fr`

Abstract. In this paper, we develop an analysis of the Needham-Schroeder Public-Key Protocol in the framework of membrane computing. This analysis is used to validate the protocol and exhibits, as expected, a well known logical attack. The novelty of our approach is to use multiset rewriting in a nest of membranes. The use of membranes enables to tight the conditions for detecting an attack. The approach has been validated by developing a full implementation for several versions of the analysis.

1 Goal and Motivations

Since the 1994 landmark demonstration by Adleman of the possibilities of DNA to solve a class of combinatorial problems, biocomputing has often be advocated to develop "chemically combinatorial problem solvers". In this paper, we want to use an approach belonging to the membrane computing [Pau02] area to address a well known combinatorial problem: the analysis of a cryptographic protocol.

Our starting point is the logical analysis of the Needham-Schroeder Public-Key Protocol (NSPK). The goal of the logical analysis is to find an interleaving of elementary actions (sending and answering messages) that allows an intruder to obtain confidential information. We have chosen this problem because it is simple to explain, at the same time it requires sophisticated data-structures for the exploration of its state space, it is paradigmatic of this kind of applications, and its solution is well-known – hence we can validate our result.

The approach taken in this paper is brute force and consists in the exploration of the state space of the protocol for a systematic search of attacks. Indeed, we are interested in the study of the states representation and generation, rather than in designing a new and smart search strategy. This approach is motivated by the opinion that the representation of data is a central problem in biocomputing.

The idea of using the membrane computing formalism in the area of communication protocols and security is not new but has not been applied before in the area of logical analysis. In [Ata03], the author proposes the use of a P system to specify a message authentication protocol. The problem addressed in our paper focus on a different point: we do not specify new protocols but rather develop a verification tool for an existing one. References [Obt01] and [KR03] study cryptographic functions (inverting one-way functions and breaking DES) which is not of our concern since encryption is handled here as a black box and is considered as unbreakable (see section 2).

The rest of this paper is organized as follows. In section 2 we give some background on the logical analysis of cryptographic protocols. Section 3 describes precisely the Needham-Schroeder public-key protocol. Section 4 presents the technical meat of the paper. We develop a version of the analysis of NSPK that improves on a similar analysis initially proposed within the ELAN rewriting framework [Cir99], with a more accurate representation of states using nesting.

2 Formal Verification of Cryptographic Protocol

In this section, we give a brief introduction to the verification problem we shall consider. Cryptographic protocols define the exchange of a few messages between parties in order to distribute some secrete data like cryptographic keys or to authenticate themselves. These messages are built using strong cryptographic primitives, but it has appeared that even though when these functions are assumed unbreakable, the security of a protocol can be compromised by an unexpected interleaving of messages between honest agents and a malicious intruder which has some limited control over the communication network.

Various formal methods have been proposed for analyzing automatically the vulnerability of cryptographic protocols to such *logical* attacks, both for searching of such flaws or for the formal proof of their absence. Several systems have been implemented in purpose for the search of flaws, *e.g.* [MCF87, Mea95, Hui99]. But many general purpose languages and tools have also appeared appropriate in this setting, with the advantage of a greater expressive power, efficiency and maturity. To cite only a few examples, there are model checkers like FDR [Low96] or murφ [MMS97], first order theorem provers [Wei99, JRV00] and declarative languages used as model checkers [DMT98, Cir99].

Our purpose in this paper is to describe an experiment to use membranes for modeling a cryptographic protocol and finding of attacks by state exploration. The declarative style supported by the membrane computing framework is strongly advocated by the intruder-centric model which is generally considered in order to apply formal methods to cryptographic protocol verification. In this model, often referred as “Dolev-Yao model” [DY83], the agents executing the protocol communicate asynchronously via a unique channel which has been compromised by an intruder. The intruder is able to spy and divert every message on the channel, to analyze read messages, with the restriction that he must know the appropriate encryption key in order to decipher an encrypted message. He can also build and send new messages, possibly under a fake identity. The global state of the system can hence be represented by a heterogeneous set containing the local states of each agent (with a bounded memory), the messages and submessages known to the intruder and the messages sent and not yet received by an agent. The actions of the agents (receiving and sending messages) as well as of the intruder can be modeled using rewriting rules on multisets. The search of an interleaving leading to an attack can be coded very simply with an appropriate pattern expression to find sequences of value or arbitrary length.

The problem of finding attacks of protocols is highly undecidable, the state space being infinite for several reasons: the unboundedness of the number of agents in presence, the ability of agents to generate fresh random data (nonces), the unlimited size of terms generated by the intruder. In order to restrict our exploration to a finite search space, while keeping our procedure reasonably complete, we shall rely on some theoretical results on protocol verification. It is shown in [RT01] that the problem of protocol security (non-existence of attacks) becomes decidable when the number of agents considered is bounded. Indeed, [RT01] shows that in this case, whenever there exists an attack, there exists an attack involving messages of a bounded size. We can use this result here to ensure the completeness of our attack search procedure, given a finite number of agents.

3 The Needham-Schroeder Public-Key Protocol

The Needham-Schroeder public key protocol [NS78] (NSPK for short) is the favorite case study for formal methods applied to the verification of cryptographic protocols. This popularity certainly comes from one of the most famous success story in this domain, which is the discover in 1994 by G. Lowe [Low95] of a replay attack in this protocol 16 years after its publication. In [Low95], G. Lowe models the protocol in the CSP process algebra and uses the model checker FDR to explore the state space.

3.1 Description of the Protocol

The Needham-Schroeder public key protocol involves two participants Alice (A), Bob (B) which are willing to authenticate reciprocally with three messages using public keys. The original protocol of [NS78] involves also a server distributing the public keys to A and B with three additional messages. We omit the server and its three messages here, assuming that A and B both initially know each other's public key, since they are not necessary in Lowe's attack. The messages are described in the usual notation :

$$\begin{array}{l} \texttt{REQ}\ \ A \to B : \{A, N_a\}_{K(B)} \\ \texttt{CHAL}\ B \to A : \{N_a, N_b\}_{K(A)} \\ \texttt{AUTH}\ A \to B : \{N_b\}_{K(B)} \end{array}$$

In the first message (labeled REQ), Alice generates a random number (nonce) N_a, appends it to her name A (the append operator is $\_,\_$) encrypts the results with Bob's public key $K(B)$ (public key encryption is denoted with the binary operator $\{\_\}_\_$) and sends the result to the network. When Bob receives a message of the form of REQ, he deciphers it and retrieves the identity A of Alice and the nonce N_a. Then he generates a second random number N_b, appends it to N_a and sends the result encrypted with Alice's public key $K(A)$ (message `CHAL` for a challenge). Alice, receiving message `CHAL`, can decipher it and check whether the first component corresponds to the nonce she sent in message `REQ`. Then, she resend Bob's nonce N_b encrypted with Bob's public key (message `AUTH`). Bob can check that the message `AUTH` contains the nonce N_b he has generated at second step (`CHAL`).

3.2 A Replay Attack

Receiving the message `AUTH` ensures Bob that Alice has really received the message `CHAL` and answered, because Alice is the only one able to decipher this message. We assume indeed that each agent, as well as the intruder (let us call him Charly, C), knows only its own private key, and that this key is necessary to decipher a message encrypted with the corresponding public key.

Similarly, when receiving the message `CHAL`, Alice is ensured that it really comes from B (and is not a fake message from Charly), as proven by the presence of N_a because the knowledge of Bob's private key is necessary for the extraction of N_a from message `REQ`. Hence, N_a and N_b are used as *authenticators* in this protocols. Therefore, they must remain secret. However, the attack of [NS78], described below, shows that it is not the case, even with the above hypotheses concerning the private keys.

$$\begin{array}{rrl}
\mathtt{REQ} & A \to C : & \{A, N_a\}_{K(C)} \\
\mathtt{REQ}' & C(A) \to B : & \{A, N_a\}_{K(B)} \\
\mathtt{CHAL}' & B \to C(A) : & \{N_a, N_b\}_{K(A)} \\
\mathtt{CHAL} & C \to A : & \{N_a, N_b\}_{K(A)} \\
\mathtt{AUTH} & A \to C : & \{N_b\}_{K(C)} \\
\mathtt{AUTH}' & C(A) \to B : & \{N_b\}_{K(B)}
\end{array}$$

This attack involves two sessions in parallel. In the first session, Alice enters in communication with Charly (without knowing that he is an intruder). Since the message `REQ` is encrypted with Charly's public key $K(C)$, Charly can retrieve A, N_a and encrypt it with Bob's public key $K(B)$, and send this message as the first message `REQ′` of a second session between A and B. In this step `REQ′`, Charly impersonates A, which is denoted $C(A)$. Bob answers to `REQ′` and Charly diverts this message `CHAL′` (it is by denoted $C(A)$). Then Charly, with two messages `CHAL` and `AUTH` of the first session uses A as an oracle in order to obtain the Bob's nonce N_b.

4 Finding an Attack on the NSPK Using Membranes and Rules

We shall describe here the specification of the Needham-Schroeder public key protocol and the implantation of an attack-search procedure using rules and membranes. For the implementation, we rely on rewriting modulo associativity and commutativity (AC) on terms representing nested multisets (membranes). Rewriting rules can be guarded by arbitrary conditions. This model is similar to the chemical computations presented in [BFM01] and we use the term "chemical solution" to denote the content of a membrane. Examples of systems implementing such model of computations are Gamma, ELAN [BKK+96], MAUDE [CDE+99] or `MGS` [Gia03]. The ingredients of this model of computation are rather sophisticated but we believe that a translation into the fundamental core mechanisms of P systems seems possible and we give in [MJ04] some elements that support this opinion.

Because of the limited room, we present in this paper only the principles of a simple version of the search procedure. This version improves on a similar analysis initially proposed within the ELAN rewriting framework, because we use a more accurate representation of states using nesting. The functional representation of the interleaving of actions is also a new idea. Note that another version is described and fully detailed in [MJG04]. This last version goes further by generalizing the approach to the exploration of general state spaces and does not rely on the assumption that attacks involve messages of bounded size.

The description of the protocol involves two different kind of components: *entities* and *evolution rules*. The entities are records and evolution rules are given by rewrite rules. A *system state*, we shall also write *solution*, is a finite collection of entities which are of three kind: agents, messages transmitted trough the network and messages components memorized by the intruder. Several entities in a state shall react, firing an evolution rule which transforms a system state into a successor state. The model is organized into the following parts, detailed in the next sections:

- record definitions are used to describe the three kind of entities (Section 4.1);
- various predicates used to select, in the set of reacting entities, a specific entity of a given kind – an agent, a message (Section 4.1);
- rules specifying the abilities of the intruder to collect all the messages that have been exchanged between agents and extracts pertinent informations (Section 4.2);
- rules specifying the abilities of the intruder to produce fake messages from the informations gathered so far (Section 4.2);
- rules specifying the reception and sending of messages by agents: such rules are defined as reactions between an agent and a (received) message which fulfills some conditions (Section 4.3);
- rules implementing a state exploration procedure which halts with a predicate checking whether a bad state is reached, hence that the search of an attack was successful (Section 4.3).

4.1 Representing Agents, Messages and Intruder Knowledge

The three different kinds of entities (unstructured informations) found in the system states (solutions) are represented using records.

Agents. We shall distinguish the *roles*, Alice and Bob in our example, which are programs, from the *agents* executing the programs, characterized by an identifier (agent's name), a role and a bounded memory. In particular, there can be several agents for one role. An agent consists in:

- an *identity* `id` (its name, note that several agents may have the same identity),
- two *stores* `ni` and `nr` to memorize the session-specific values of the nonces N_a and N_b,
- a *program counter* `pc`, which can take the value described below.

Every agent with either role Alice or Bob shall create a nonce and receive another one during the execution of the protocol of Section 3.1. The fields `ni` and `nr` store these two values, for Alice, `ni` stores N_a and `nr` stores N_b, and reciprocally for Bob (`ni` stands for *nonce initial*, because we can assume that each agent initially creates the nonces before starting a session of the protocol, and `nr` stands for *nonce received*).

The program counter `pc` of an agent can take the following values (these values are arbitrary symbols and prefixed by a backquote), according to the role: `` `REQ ``, `` `AUTH `` and `` `FINISHED `` for Alice and `` `CHAL ``, `` `WAIT `` and `` `FINISHED `` for Bob. For Alice, `pc` = `` `REQ `` means that the agent is about to send the message with the corresponding label in the protocol specified in section 3.1, and similarly for `pc` = `` `AUTH `` (role Alice) and `` `CHAL `` (Bob). For an agent playing the role of Bob, `pc` = `` `WAIT `` means that he is waiting for the answer of Alice to his challenge `CHAL`, and `pc` = `` `FINISHED `` means that the agent has completed his session of the protocol.

Messages. Three different kinds of messages are exchanged between Alice and Bob during the protocol. We define a predicate to recognize each kind of messages: `REQ`, `CHAL` and `AUTH`. Messages are also records and they are characterized by the kind of information that they hold. For instance, messages of type `REQ` contain a field `na` representing the content of the message and a field `kb` which is the public key used for encryption. For the sake of simplicity, in our program, every public key or private key is represented by the identity of the owner.

Intruder Knowledge. The knowledge of the intruder is also represented by records with fields `name`, `nonce`, `pub`, `priv`. We define several predicates (`info_name`, `info_nonce`, `info_pub` and `info_priv`) for each kind of information that the intruder will be able to reveal from the whole history of exchanged messages: name, nonce, public key and private key. These predicates are used to determine the presence of a message of a given kind with a given information in the solution.

4.2 The Intruder Transformation Rules

The network is common to all agents and the intruder, hence the latter is able to read and produce new messages. This behavior is implemented by the rules presented in the two following sections.

Reading and Analyzing Messages. In our approach, the existing messages are read by the intruder from the current state and they are put back unchanged. Moreover, the encrypted contents of a message are added as new known information to the state if decryption is possible. More precisely, the intruder can learn a plaintext encrypted with a public key (for instance the nonce `nb` encrypted with `kb` in message `AUTH`) only if he knows the corresponding private key.

The following three rules define the evolution of the knowledge of the intruder, according to the messages present in the network. There is exactly one rule for each kind of message. They will actually not generate all the informations that the intruder can extract from collected message. However, these transformations are sufficient to extract all the information needed to built messages with the forging rules below. For instance, if a message m present in the solution has type `REQ`, and the intruder knows the private key associated to $m.\mathtt{kb}$, then he learns the components $m.\mathtt{na}$ and $m.\mathtt{a}$ of m. Theoretically, he also learns the pair $(m.\mathtt{na}, m.\mathtt{a})$ but storing such an information is useless since we assume that the intruder is able to build pairs arbitrarily.

$$m \longrightarrow m, \{\mathtt{nonce} = m.\mathtt{na}\}, \{\mathtt{name} = m.\mathtt{a}\}$$
$$\mathtt{where}\ \ m \in \mathtt{REQ} \wedge \exists k \in \text{self s.t. } k.\mathtt{priv} = m.\mathtt{kb}$$

$$m \longrightarrow m, \{\mathtt{nonce} = m.\mathtt{na}\}, \{\mathtt{nonce} = m.\mathtt{nb}\}$$
$$\mathtt{where}\ \ m \in \mathtt{CHAL} \wedge \exists k \in \text{self s.t. } k.\mathtt{priv} = m.\mathtt{ka}$$

$$m \longrightarrow m, \{\mathtt{nonce} = m.\mathtt{nb}\}$$
$$\mathtt{where}\ \ m \in \mathtt{AUTH} \wedge \exists k \in \text{self s.t. } k.\mathtt{priv} = m.\mathtt{kb}$$

The keyword "self" used in the rules denotes the current multiset (i.e. the multiset from which m is chosen). The existential quantifier in the guard of the rules checks that some condition is satisfied by an element k in a given multiset: such kind of predicate is easily computed by set of reduction rules.

Forging Some New Messages. In the previous section, we have describe the `intruder` rules set which only reveals information according to already known messages an keys. The following rule *produces* a new fake `REQ` message from know informations in the solution:

$$k, n, m \longrightarrow \{\mathtt{na} = m.\mathtt{nonce}, \mathtt{a} = n.\mathtt{name}, \mathtt{kb} = k.\mathtt{pub}\}$$
$$\mathtt{where}\ \ \mathtt{info\_pub}(k) \wedge \mathtt{info\_name}(n) \wedge \mathtt{info\_nonce}(m)$$

There is one such rule for the two other kinds of messages `CHAL` and `AUTH`. These rules are used to produce by saturation (fixed point computation) all possible fake message that can be forged from the known facts in a multiset.

An attack consists in revealing all possible informations using the above rules of the intruder after having forged all possible fake messages. Actually, we'll see in the following that a *real* attack always consists in the application of the attack rules of Section 4.3 until a fixed point is reached.

4.3 *Nested* Multiset Rewriting to Explore the State Space

The first idea to implement the logical analysis of NSPK is to aggregate all the entities involved into the protocol in a single multiset acting as a chemical solution containing the agents, the messages and the revealed informations. The agents and the intruder will react with messages to augment the solution with new informations. All informations are in the solution at the same level. An attack on the NSPK protocol consists here in finding an interleaving of the agents actions described below such that Bob's nonce is revealed.

This approach suffers from the following problem: let S be a solution and a be an agent in a state where he might reply to two different messages m_1 and m_2. The two following scenarii could happen:

1. The agent replies to both messages: to m_1 to give m'_1 and to m_2 to give m'_2. Here, after the agent action, S is equal to $S \cup m'_1 \cup m'_2$. In the future of the protocol, another agent may react to *both* m'_1 *and* m'_2 leading to an incorrect situation, even where the intruder may break the protocol and reveal the nonce.
2. The agent replies to only one of the two messages: to m_i to produce m'_i. In that case, an attack might not be found because the case where the reply should have concerned the other message has not been considered. The protocol analysis is therefore too weak.

The consequence is that we have to take into account the different evolutions of the protocol that might happen when an agent receives more than one message. To model such a situation, we make use of several multisets (membranes) to localize the computation and to avoid the (possible) interferences. The initial state consists in a multiset of multisets. Each element in the top multiset (the *skin* in the language of P systems) is a possible state in the protocol and represent some possible evolution.

The Agents. The behavior of each agent, at each possible `pc`, is described by a set of rules. For example, the behavior of `Alice` with `pc` = `‘AUTH` is to switch to the state `‘AUTH` and to produce a new message:

$$x, t \longrightarrow (x + \{\mathtt{pc} = \mathtt{`AUTH}\}), \{\mathtt{kb} = x.\mathtt{dest}, \mathtt{na} = x.\mathtt{ni}, \mathtt{a} = x.\mathtt{id}\}$$
$$\mathtt{where}\ \ x \in \mathtt{REQ} \wedge x.\mathtt{id} = \mathtt{alice} \wedge t = \mathtt{`OK}$$

the operator $+$ is the asymmetric merge of records and the results of $x + \{\mathtt{pc} = \mathtt{`AUTH}\}$ is a record equal to $x$ except for the field `pc` that takes the value `‘AUTH`. The variable $t$ matches a symbol used to inhibit or activate the rule: if the symbol is present (e.g. $t = \mathtt{`OK}$) then the rule can be triggered. If the symbol `‘OK` is not present in the chemical solution, the rule is inhibited.

There are three additional similar rules to describe the evolution of Alice waiting for the authentication, Bob waiting for a challenge and Bob in the finishing state.

Note that the messages addressed to an agent must not be removed from the solution and are available for other rule applications.

The Initial State. The initial state for the attack search consists in a multiset (of multisets) with only one element:

– the two agents, Alice and Bob, initialized with their respective identity, the destination of the message for Alice, initial nonces to arbitrary integer values, program counter,
– intruder knowledge (public keys for all participants and its own private key).

Looking for an Attack. In our definition of the initial state, the number of agents is fixed and remains such. Therefore, the number of execution steps is bounded accordingly. The problem consists in finding the correct interleaving of Alice and Bob actions leading to a successful attack.

The basic idea is to generate all strings of bounded length made of four symbols representing an evolution of one of the agent (see the rule set of an agent described above). The combinatorial generation of such string is easy and can be done randomly. Then a rule is used to trigger the "application" one of such string to an agent to make this agent evolve:

$$m, \text{`}\mathtt{alice\_req}::s \longrightarrow m, \text{`}\mathtt{OK}, s$$

The expression `alice_req::$s$ denotes a string beginning with the symbol `Alice_req. Note that the tail s of the string is released in the solution. The production of the triggering symbol `OK activates the evolution rule on m. By adjoining a trigger to this rule, which is released by the agent evolution rule and consumed by this rule application, we can interleave correctly the evolution of an agent until the exhaustion of the string s.

We still look for an interleaving leading to revealing the nonce. A successful attack is to find in the chemical solution the nonce of Bob revealed. This is done by adding a specific rule, e.g. a rule leading to a dissolution of all enclosing membranes.

Validation in the MGS Programming Language. To validate our propositions, we have completely implemented and validated several versions of the logical analysis using the MGS programming language. MGS is a research project devoted to the design and the development of a programming language dedicated to the simulation of biological processes [Gia03, GMM04]. Based on topological notions, MGS supports the notion of transformation: a localized computation specified by rules. One can for example defines multiset rewriting rules [BFM01] that act on a nest of multisets (i.e., membranes). These rules can be used to move values from a multiset to another one, as well as to dissolve, divide or create new multisets. So, MGS can potentially be used to process membranes. However, we outline that the MGS project focuses on the design of a programming language rather than the development of a well founded computational model.

The search for an attack succeeds in less than a second on a AMD-1.4Ghz-LINUX computer, and reveals that the correct interleaving of functions is, as expected, bob_finish ∘ alice_auth ∘ bob_chal ∘ alice_req. The implemented code and the MGS interpreter are available from URL mgs.lami.univ-evry.fr .

5 Summary

In this paper, we have used the membrane computing approach to describe and analyze the NSPK protocol. This application of membrane computing is new to the best of our knowledge. Existing applications of membrane computing and P systems to cryptography focus on protocol specification [Ata03] or to cryptographic functions [Obt01, KR03]. It has been shown that using our approach, the well-know security hole of [Low96] is easily (in less than one second) discovered by our state exploration procedure.

In the proposed version, we are searching for the correct interleaving of the agents actions leading to a possible attack. Using membranes permits us to handle correctly the fact that an agent may have to react to more than one message leading to more than one evolution of the state.

Nevertheless, this method is tailored for the search of an interleaving of agents actions leading to the revelation of the nonce. This is possible because we actually know that such an interleaving *will* lead to a successful attack. We have proposed in [MJ04] a more general approach where a full state space search is done. The complete running code of the two versions have been implemented in `MGS` and is detailed in [MJG04]. The complete code is particularly simple and readable. Moreover, it is also easy to evolve the initial analysis to more sophisticated ones.

The approach presented here has been developed for this special protocol and heavily rely on the nesting of membranes to localize the computation and to avoid evolution interference leading to more approximate analysis. We believe that the principles of our modeling are general enough to envision a systematic way to derive a program for searching attacks from an abstract description of the messages of a protocol given with the notations of Section 3.1, following [JRV00].

Acknowledgments. The authors would like to thanks the reviewers for their very helpful suggestions. They are also grateful to Jean-Louis Giavitto, Julien Cohen and Antoine Spicher at LaMI for stimulating discussions and thoughtful remarks. This research is supported in part by the CNRS, the GDR ALP, the University of Évry, Genopole©, INRIA and ENS Cachan, the RNTL project PROUVÉ and the ACI-SI Rossignol.

References

[Ata03] A. Atanasiu. Authentication of messages using P systems. In Gh. Paun, G. Rozenberg, A. Salomaa, and C. Zandron, editors, *Membrane Computing: International Workshop, WMC-CdeA 2002, Curtea de Arges, Romania, August 19-23, 2002. Revised Papers.*, volume 2597 of *Lecture Notes in Computer Science*, pages 33–42, Curtea de Arges, Romania, July 2003. Springer-Verlag, Berlin.

[BFM01] J.-P. Banâtre, P. Fradet, and D. Le Métayer. Gamma and the chemical reaction model: Fifteen years after. *Lecture Notes in Computer Science*, 2235:17–??, 2001.

[BKK+96] P. Borovansky, C. Kirchner, H. Kirchner, P. E. Moreau, and M. Vittek. ELAN: A logical framework based on computational systems. In *Proc. of the First Int. Workshop on Rewriting Logic*, volume 4 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1996.

[CDE+99] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. F. Quesada. The Maude system. *Lecture Notes in Computer Science*, 1631:240–??, 1999.

[Cir99] H. Cirstea. Specifying authentication protocols using elan. In *Workshop on Modelling and Verification*, Besancon, France, december 1999.

[DMT98] G. Denker, J. Meseguer, and C. Talcott. Protocol specification and analysis in Maude. In *In Proc. of Workshop on Formal Methods and Security Protocols*, 1998.

[DY83] D. Dolev and A. Yao. On the security of pubic key protocols. *IEEE Transactions on Information Theory*, IT-29(2):198–208, March 1983.

[Gia03] J.-L. Giavitto. Invited talk: Topological collections, transformations and their application to the modeling and the simulation of dynamical systems. In *Rewriting Technics and Applications (RTA'03)*, volume LNCS 2706 of *LNCS*, pages 208 – 233, Valencia, June 2003. Springer.

[GMM04] J.-L. Giavitto, G. Malcolm, and O. Michel. Rewriting systems and the modelling of biological systems. *Comparative and Functional Genomics*, 5:95–99, February 2004.

[Hui99] A. Huima. Efficient infinite-state analysis of security protocols. In *Proceedings of FLOC'99 Workshop on Formal Methods and Security Protocols*, 1999.

[JRV00] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and verifying security protocols. In *Logic for Programming and Automated Reasoning (LPAR'00)*, volume 1955 of *Lecture Notes in Computer Science*, November 2000.

[KR03] Shankara Narayanan Krishna and Raghavan Rama. Breaking des using p systems. *Theoretical Computer Science*, 299(1-3):495–508, April 2003.

[Low95] Gavin Lowe. An attack on the needham-schroeder public key authentication protocol. *Information Processing Letters*, 56(3), november 1995.

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In T. Margaria and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, volume 1055, pages 147–166, Passau, Germany, march 1996. Springer-Verlag, Berlin Germany. Also in Software Concepts and Tools, 17:93-102, 1996.

[MCF87] J.K. Millen, S.C. Clark, and S.B. Freedman. The interrogator: Protocol security analysis. *IEEE Transactions on Software Engineering*, SE-13(2), 1987.

[Mea95] Catherine A. Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, february 1995.

[MJ04] O. Michel and F. Jacquemard. An analysis of the needham-schroeder public-key protocol with MGS. In G. Mauri, G. Paun, and C Zandron, editors, *Preproceedings of the Fifth workshop on Membrane Computing (WMC5)*, pages 295–315. EC MolConNet - Universita di Milano-Bicocca, June 2004.

[MJG04] O. Michel, F. Jacquemard, and J.-L. Giavitto. Three variations on the analysis of the needham-schroeder public-key protocol with MGS. Technical Report LaMI-98-2004, LaMI – Université d'Evry - CNRS, May 2004. 25 p.

[MMS97] J. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murphi. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 141–151. IEEE Computer Society Press, May 1997.

[NS78] Roger M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[Obt01] A. Obtulowicz. Membrane computing and one-way functions. *International Journal of Foundations of Computer Science*, 12(4), August 2001.

[Pau02] Gh. Paun. *Membrane Computing. An Introduction*. Springer-Verlag, 2002.

[RT01] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *Proceedings of the 14th Computer Security Foundations Workshop (CSFW'01)*, pages 174–190, Cape Breton, Nova Scotia, Canada, 2001. IEEE Computer Society Press.

[Wei99] C. Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction, CADE'99*, volume 1632 of *Lecture Notes in Computer Science*, pages 378–382, 1999.