RAPPORT TECHNIQUE EVA

The EVA environment: Experimentation Report

Date	:	24 Décembre 2003
Auteurs	:	Gustavo Betarte, Daniel Le Métayer (avec des contributions de Véronique Cortier, Michäel Périn et Yassine Lakhnech)
Titre	:	The EVA environment: Experimentation Report
Rapport No. / Version	:	15/ 1.0

TRUSTED LOGIC S.A.
5 rue du Bailliage
78000 Versailles, France
www.trusted-logic.fr

Laboratoire Spécification Vérification
CNRS UMR 8643, ENS Cachan
61, avenue du président-Wilson
94235 Cachan Cedex, France
www.lsv.ens-cachan.fr

Laboratoire Verimag
CNRS UMR 5104,
Univ. Joseph Fourier, INPG
2 av. de Vignate,
38610 Gières, France
www-verimag.imag.fr

Contents

1 Introduction

This report summarizes the main contributions of the project, devoting special attention to the verification tools that have been developed.

The report is the result of an experimentation effort conducted within the EVA project. The goals were twofold:

- to experiment with the protocol verification environments developed during the project and
- to define avenues for further research.

The criteria that have been used for this experimentation stem from the original objective of EVA which was to contribute reducing the gap between the state of the art in verification technology and the practical use of verification tools in the context of security certifications. On one hand, a longstanding and very successful series of academic work on verification has resulted in a number of well-founded and general purpose techniques which are powerful enough to deal with more and more complex problems. On the other hand, while some of these techniques have found their way into hardware verification environments, very few software verification tools are really used in industry. It is still the case though, that some specific areas such as security would benefit from innovative verification tools. The partners of the EVA project decided to work in two directions to reduce this gap:

- Focus on a well identified and critical type of software, namely cryptographic protocols, and exploit this specialization to propose dedicated tools offering an increased level of automation.
- Enhance the usability of the tools through user friendly interaction and explanation facilities.

Needless to say, this ambitious objective goes far beyond a three years time project such as EVA. It is thus of prime importance, at the end of this period, to review the results achieved within the project and the progresses that remain to be done. It is hoped that such conclusions can provide useful input to define a roadmap for further research in the area.

In order to avoid any bias, the experiments reported herein were mainly conducted by a newcomer in the project with background in security certification (Common Criteria) and formal methods, but without specific knowledge in cryptographic protocol verification. Help was provided, but only when needed, by the implementers of the tools. Four main issues were distinguished through the experimentation process, which correspond to the main stages in the use of the tools:

- Protocol definition in LAEVA.
- Protocol analysis using (at least one of) the verification tools (Hermes and Securify).
- Exploitation of negative results of the verification tools (failure): understanding the origin of the failure, the conclusions to be drawn, the way to go forward (protocol debugging), etc.
- Exploitation of positive results of the verification tools (successful proof) including the use of such results in a security evaluation process.

The rest of this document ir organized as follows: Section 2 provides an overview of the tools, describing the theoretical settings upon which they were built and the mechanisms supporting their operation. The protocols and properties that have been used and the results of the conducted experiment are presented in Section 3. For each of the usage stages described above both useful features and limitations of the tools are pointed out. In Section 4 some suggestions for further work are put forward. These suggestions are of very different types (long-term or mid-term research, engineering, documentation, etc.). Finally, Section 5 provides an overview of the main conclusions and perspectives for further work.

2 Highlights of the Verification Tools

2.1 Technical Underpinnings

The experiments described here were conducted on the HERMES and SECURIFY tools. This section briefly describes the theoretical settings, developed by the research teams participating in the project, upon which rely the verification mechanisms supported by those tools.

The development of HERMES is based upon theoretical results concerning the symbolic verification of cryptographic protocols. This work is described in detail in [BLP02a] and [BEL03a]. The main results are:

1. An expressive second-order logic on terms for symbolically describing infinite sets of states of a protocol.

2. This logic allows to describe in a natural manner secrecy, authentication and other safety properties.

3. A decision procedure for the satisfiability of the logic.

4. A complete predecessor calculus based on this logic.

5. An abstract interpretation based extension of the calculus to unbounded protocols involving the definition of a specific widening operator for termination.

This work has been extended to timed cryptographic protocols in [BEL03b].

The starting point of the algorithm implemented by SECURIFY is the decomposition theorem of Millen and Rueß model [MR00]. In that work the proof that a protocol satisfies a secrecy property is separated into two parts: a protocol-dependent part and a protocol-independent one. A careful analysis of the second class of proofs led to notice that all of these proofs follow a standard pattern that is amenable to mechanization, and which also has a convenient visual representation. A crucial part of this procedure is to reduce messages to simplified components called *branches*. The core of the verification algorithm is a search procedure for establishing secrecy results. This procedure is sound and automatic but incomplete in that it may fail to establish secrecy for some secure protocols. A detailed description is provided in [CMR01].

To help express secrecy goals, SECURIFY makes use of the spell events introduced in [MR00]. They are specification of secrecy, handled like messages sent through the network and local states of the participants. A state of the state-transition system is a set of *spell* events, messages and local states, which represents an history of what has already happened.

2.2 Features

The verification mechanisms of the tools are described in more detail in this section. The working of the tools is illustrated through their application to the verification of a particular cryptographic protocol.

2.2.1 Description of a protocol and its properties in LAEVA

As an example we consider the Needham-Schroeder-Lowe protocol [Low95]. The specification of this protocol in terms of the language LAEVA [JM01, GL01] is presented below. LAEVA is the high level specification language designed in the EVA project for describing security protocols and their properties.

```
Needham_Schroeder
A, B : principal
N1, N2 : number
keypair pbk,prk (principal)
```

pbk and prk are key constructors that take a principal and return an asymmetric key: pbk(A) stands for public key of principal A. The private key of A, denoted by prk(A), is the inverse of pbk(A).

```
everybody knows pbk
A knows A, B, prk(A)
B knows B, prk(B)
```

The knowledge of the principals is needed to generate the operational model of the protocol ; it is used to rule out ambiguities.

```
1.A->B: {A, N1}_(pbk(B))
2.B->A: {N1, N2}_(pbk(A))
3.A->B: {N2}_(pbk(B))

s.session* {A,B,N1,N2}
```

*The protocol specification is close to the standard notation. It describes an ideal session between an initiator (role A) and a responder (role B). The roles A, B, and the nonces N1, N2 that they create are the parameters of a session. The * symbol asks the tool to consider an unbounded number of sessions in parallel. For debugging purpose, some tools (e.g.,* HERMES*) can also run with a fixed number of sessions.*

```
assume
  secret(prk(A)@s.A),
  secret(prk(B)@s.B),
  secret(prk(B@s.A)),
  secret(prk(A@s.B))
```

`secret(prk(B@s.A)` *means that the private key – of the entity playing the role B, from A's point of view in session s – is unknown to the intruder. Secrecy hypothesis on keys are needed to reason about encrypted messages.*

```
claim
  *A*G secret(prk(A)@s.A),
  *A*G secret(prk(B)@s.B),
  *A*G secret(N1@s.A),
  *A*G secret(N2@s.B)
```

The three verification tools check that secrecy properties hold `*A`lways *and* `*G`lobally*. The first two claims require that the initial secrets remain secret. The two others asks that the nonces N1 and N2 created by role A (resp. role B) in session s are secret.*

A specification like the one above is compiled, using EVATRANS, a front-end translator developed within the EVA project [Jac03], into a concrete operational model and a security property to be verified. These outputs can in turn be provided as inputs to either of the three automatic verification tools developed in the EVA project: SECURIFY [CMR01, Cor02], CPV [GL02] and HERMES [BLP02a, BLP02b].

The diagram of Figure 1 illustrates the relation between the specification of the Needham-Schroeder protocol given in LAEVA *(on dashed arrow)* and the process that describes the role of the initiator A and the role of the responder B *(in column)*.

2.2.2 HERMES

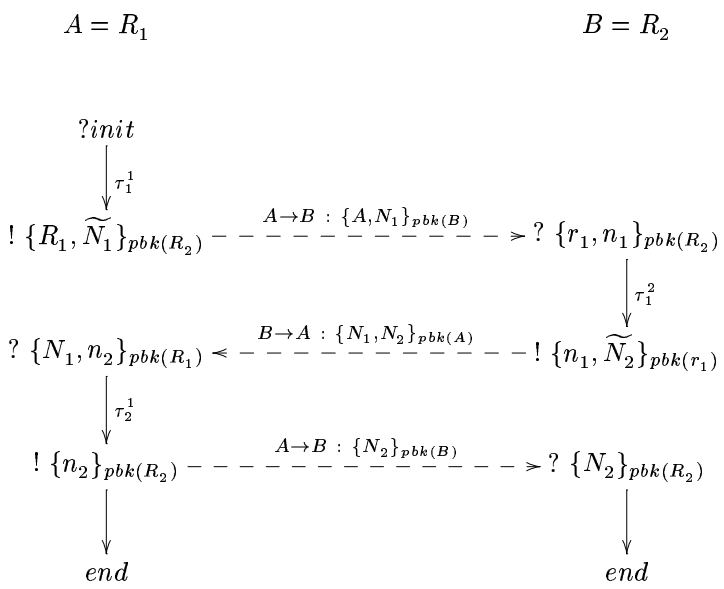
2.2.2.1 Hermes front-end HERMES runs its computation on a protocol defined by the following transitions extracted from the LAEVA specification.

role R_1				*role* R_2			
(1.)	$?init$	$\xrightarrow{\tau_1^1}$	$!\{R_1, N_1\}_{pbk(R_2)}$				
				(2.)	$?\{R_1, n_1\}_{pbk(R_2)}$	$\xrightarrow{\tau_1^2}$	$!\{n_1, N_2\}_{pbk(R_1)}$
(3.)	$?\{N_1, n_2\}_{pbk(R_1)}$	$\xrightarrow{\tau_2^1}$	$!\{n_2\}_{pbk(R_2)}$				

They correspond to the transitions of role R_1 and R_2 in Figure 1. These transitions describe a generic session of the protocol, they are parameterized by (R_1, R_2, N_1, N_2). Then, a session of the protocol is completely defined by instantiating the roles R_1 and R_2 with some principals, and N_1, N_2 with two fresh nonces. Note that n_1 and n_2 denote free variables which are local to the session.

When the user asks for a verification with an unbounded number of sessions (set by the $*$ symbol), HERMES computes an abstract model of the protocol. It considers only one honest principal H and one dishonest principal I (the intruder) and studies the following session instances :

$$(H, H, N_1, N_2),\ (H, H, N_1^{HH}, N_2^{HH}),\ (H, I, N_1^{HI}, N_2^{HI}),\ (I, H, N_1^{IH}, N_2^{IH})$$



- $A = R_1$ and $B = R_2$ means that the principal A plays the role 1 (initiator) and B plays the role 2 (responder).
- τ_i^r denotes the i^{th} transition of role R_r. A transition is a reception of a message that matches a pattern, written $?pattern$, followed by a emission of a reponse message, written $!reponse$. Both parts of a transition are extracted from the specification of the protocol in LAEVA.
- $\widetilde{N}$ denotes the creation of a fresh nonce N and its first use.
- Capital letters like N_i, R_i are constants of the given session ; small letters denote variables.
- $pbk(p)$ is the public key of principal p.

Figure 1: Needham-Schroeder protocol specification

They suffice to model unbounded sessions. The first one is the distinguished session whom secrets are under attention. In this session, the honest principal H plays all the roles of the protocol without lost of generality. The other instances model the three possible type of sessions : between two honest principals (session HH), between one honest and one dishonest principal initiated either by the honest one (session HI) or by the dishonest one (session IH). These instances lead to four instantiation of the generic rules (see Figure 2).

Generic session parameterized by (R_1,R_2,N_1,N_2)	The abstract models consists in four specific instantiations of the generic session: the fixed session (H,H,N_1,N_2)	sessions of type HH (H,H,N_1^{HH},N_2^{HH})	sessions of type HI (H,I,N_1^{HI},N_2^{HI})	sessions of type IH (I,H,N_1^{IH},N_2^{IH})
$\frac{init}{\{R_1,N_1\}_{pbk(R_2)}};$	$(\tau_1^1)\frac{init}{\{H,N_1\}_{pbk(H)}};$	$\frac{init}{\{H,N_1^{HH}\}_{pbk(H)}};$	$\frac{init}{\{H,N_1^{HI}\}_{pbk(I)}};$	$\frac{init}{\{I,N_1^{IH}\}_{pbk(H)}};$
$\frac{\{r_1,n_1\}_{pbk(R_2)}}{\{n_1,N_2\}_{pbk(r_1)}};$	$(\tau_1^2)\frac{\{r_1,n_1\}_{pbk(H)}}{\{n_1,N_2\}_{pbk(r_1)}};$	$\frac{\{r_1,n_1\}_{pbk(H)}}{\{n_1,N_2^{HH}\}_{pbk(r_1)}};$	$\frac{\{r_1,n_1\}_{pbk(I)}}{\{n_1,N_2^{HI}\}_{pbk(r_1)}};$	$\frac{\{r_1,n_1\}_{pbk(H)}}{\{n_1,N_2^{IH}\}_{pbk(r_1)}};$
$\frac{\{N_1,n_2\}_{pbk(R_1)}}{\{n_2\}_{pbk(R_2)}};$	$(\tau_2^1)\frac{\{N_1,n_2\}_{pbk(H)}}{\{n_2\}_{pbk(H)}};$	$\frac{\{N_1^{HH},n_2\}_{pbk(H)}}{\{n_2\}_{pbk(H)}};$	$\frac{\{N_1^{HI},n_2\}_{pbk(H)}}{\{n_2\}_{pbk(I)}};$	$\frac{\{N_1^{IH},n_2\}_{pbk(I)}}{\{n_2\}_{pbk(H)}};$
	only once with (τ_1^1) *before* (τ_2^1)			

Figure 2: Inference rules that define the abstract model used in HERMES

```
s0.session(H,H,N1,N2)
s1.session(H,H,N1^HH,N2^HH)
s2.session(H,I,N1^HI,N2^HI)
s3.session(I,H,N1^IH,N2^IH)
```

Then, HERMES proceeds to the verification in the same way as for a finite number of sessions, except that the transitions of sessions `s1, s2, s3` can be played an infinite number of times, in any order, while for a finite number of sessions each transitions is played only once and each role fires its transitions one after the other. In formal terms, the abstract model corresponds to a set of rules on the right hand side of Figure 2: the nine inference rules of session `s1, s2, s3` which can be played any number of times and in any order ; plus the set of the three rules of session $\tau_1^1, \tau_2^1, \tau_1^2$ of the fixed session $(H, H, N1, N2)$ which are played only once and respecting the ordering constraint on transitions of each role. R_1 IMPOSES to fire τ_1^1 before τ_2^1. This is the only ordering constraint since R_2 has only one transition, τ_1^2.

2.2.2.2 Verification principle From the abstract version of the protocol and the secrecy property, both extracted from the LAEVA specification of the protocol. HERMES computes what are the conditions that suffice to guarantee that messages sent by honest principals during a protocol session cannot be used by the intruder to get to know secrets that should be preserved during that session.

Given a protocol specification, a set S of secrets and a collection of hypotheses over those secrets a HERMES run produces:

1. A set G of message patterns which protect the secrets interchanged during a protocol session, so called *Good patterns*, and

2. A set B of message patterns which do not protect those secrets, called *Bad patterns*. The instantiations of these bad patterns are the messages that can be used to mount an attack.

The protecting messages are all ciphered message where K is a key whose inverse is not in the possession of the intruder. These messages can be derived from the protocol specification by using

the keys that have been specified as secret keys in the hypothesis. The protecting messages are finitely represented by the set of *good patterns* G defined as all patterns $\{x\}_K$ where x denote a variable and K is a secret key (given in the hypotheses).

The output of HERMES (the algorithm is shown to be terminating) consists of:

1. A set S' of secrets, which contains in particular the declared secrets of S, and
2. A set P of protecting messages, which is defined as the difference of two sets. The protecting messages are the instances of a good pattern which are not an instance of a bad pattern, that is formally:
$$P = \mathit{Instance}(G) \setminus \mathit{Instance}(B)$$

The Secrecy property is verified if, in every message initially known by the intruder, every secret belonging to S' is embedded messages that belong to P.

2.2.2.3 Correctness of the verification principle We briefly present the verification principle implemented in HERMES for it is useful to understand some technical points in Section 2.2.2.6 about interpretation of HERMES output. We select the knowledge that should make a user at ease with HERMES. We here concentrate on the general idea and keep out the technical material and formal definitions. The verification principle implemented in HERMES is fully described in [BLP02a].

The verification of a protocol $\mathscr{P}$ described as a set of generic transitions is a fixpoint computation. HERMES starts with a given set of secrets S, the set of good patterns $G = \{ \{x\}_k \mid k^{-1} \in S \}$ and the empty set of bad patterns $B = \emptyset$. It explores all possible sequence of transitions of the protocol and reduces the set of protecting messages (by adding patterns to B) and augments the set of secret S until it reaches a stable triple (S', G, B') such that:

> *whatever the messages sent along this sequence of transitions, the secret of S' are always embedded in a protecting messages – defined as $\mathit{Instance}(G) \setminus \mathit{Instance}(B')$.*

The correctness of HERMES verification relies on the following proposition.

> Consider a protocol $\mathscr{P}$, a set S of secrets, two sets, G and B, of good and bad patterns. Assume that the triple (S, G, B) is a fixpoint for HERMES computation on the protocol $\mathscr{P}$, then we conclude that:
>
> > *if, in the messages initially known by the intruder, the secrets of S are embedded in protecting messages defined by the pair (G, B), then, the secrets are protected in all messages that the intruder can deduce when playing any number sessions of the protocol $\mathscr{P}$.*

This result is formally stated by Proposition [11] of Figure 3 and it is automatically proved in COQ for a given protocol $\mathscr{P}$, secrets S and patterns G, B computed by HERMES.

2.2.2.4 Hermes results: Proof obligations The results of Hermes must be interpreted as constraints that defines the acceptable initial knowledge of the intruder. These constraints can be interpreted as conditions on the manner the protocol must be used. In the context of the EVA project, however, the initial conditions are fixed, and they define what is the information known by the intruder. Therefore the constraints should actually be understood as proof obligations.

The initial conditions of a protocol are specified in the LAEVA language by using declarations of the form `assume *A*G secret(K@s.A)`. This declaration states that the value of the key K (in session s and from A's point of view) is supposed to be unknown to the intruder. Hypotheses like this one are used in order to develop the proof that the constraints imposed by HERMES are satisfied.

2.2.2.5 Hermes back-end: counter example or correctness proof HERMES is provided with a back-end that helps in interpreting the results (S', G, B). The back-end includes an heuristic algorithm to detect if the extended set of secrets S' contains a message that can be forged by the intruder. For instance, a message m in S' cannot be secret if it is only made of public knowledge (the identity of principals, their public keys) and data known by the intruder (e.g., shared in sessions between honest principals and the intruder). In this case, HERMES detects that the proof obligations cannot be fullfiled and provides the sequence of transitions that has led to a requirement of secrecy on m.

Counter example This sequence can be interpreted as the trace of an attack in the abstract model. As it is, it can be difficult to understand without a deep knowledge in HERMES verification principle. So, the back-end provides a counter example feature that computes a corresponding attack on the concrete model and shows it graphically as a Message Sequence Chart (see Section 2.2.2.7 and Figure 4 for an example).

All the traces explored by HERMES are summarized graphically as a tree whom leaves conclude either with an attack on a secret, or with a stabilization of computation that allows to cut the exploration of the branch. All the traces leading to an attack leaf can be given to the counter example generator.

Correctness proof in Coq On the other hand, when no attack is detected, HERMES conclude to the correctness of the protocol under some assumption on the initial knowledge of the intruder. In order to emit a certificate based on HERMES results, the certifying authority do not need to check and gain confidence neither in the verification principle underlying HERMES, nor in its implementation. To leave no doubt on the results of the verification, HERMES ends with a *proof of correctness of the protocol* that can be replayed in COQ proof-checker.

2.2.2.6 Certifying Hermes computation This section enumerates what the user should admit to recognize the validity of the correctness proof. A COQ theory is a collection of type definitions, predicate and functions definitions, and axioms which are the basis to express and to prove the wanted proposition. To prove a secrecy property of a given cryptographic protocol $\mathscr{P}$ in the presence of an intruder, the theory must contain:

1. The definition in COQ of the execution model of a protocol (predicate definition [6] of Figure 3).

2. The translation in COQ of the specification of protocol $\mathscr{P}$ and of its secrecy property S (definitions [7] of Figure 3).

3. The definition in COQ of the inference rules of Dolev-Yao's model that defines the intruder capabilities (predicate definition [3] of Figure 3). Henceforth, the proof of correctness ensures that the protocol is not vulnerable to an intruder with deduction capabilities and that controls the network. But it tells nothing on an intruder with other capabilities like, for instance, those of guessing a week key or weak password.

4. The proposition that states that protocol $\mathscr{P}$ satisfies the secrecy property S (proposition [11] of Figure 3).

A certifying authority only has to check these four points to recognize the "`proved`" answer of COQ engine as a proof of correctness of the protocol. This task should be very light since the specification in COQ of the protocol, the secrets, and all definitions are obtained by direct translations. Obviously, this approach requires to put confidence in the COQ proof-checker. Figure 3 describes the COQ theory generated by HERMES back-end to prove that a given protocol $\mathscr{P}$ satisfies the secrecy property S. We can make the following observations on Figure 3:

- The verification principle of HERMES bears on Predicate [2] $E\langle G, B\rangle S$ which is true if in all messages of E, the secrets of S are embedded in the protecting message defined by the pair (G, B). This predicate is defined as a check on the structure of messages in E, S, G, B. It does not directly use the definition of Dolev-Yao's intruder deduction.
- Predicate [3] $E \vdash m$ is true if the message m can be deduced from the set of messages E using Dolev-Yao's inference system; This predicate defines the intruder deduction capabilities.
- Predicate [4] $E \nvdash m$ states that a message m cannot be deduced from the set E of messages. It is useful to express the secrecy property.
- Proposition [5] relates the predicate $E\langle G, B\rangle S$, specific to HERMES, to the predicate $E \vdash m$ which defines Dolev-Yao's model. This proposition is proved in COQ, thus ensuring that HERMES predicate gives a sufficient condition to avoid deduction by the intruder.
- Predicate [6] defines the concrete execution model: $(E,\Omega) \xrightarrow{\tau} (E',\Omega')$ is true if the transition τ leads from the state (E,Ω) to the state (E',Ω') where:
  - Ω, Ω' denote sets of protocol sessions in execution. In particular, this data structure records the next fireable transitions (see [EVA-RT5] for details on the concrete model).
  - τ can either be a fireable transition in a session of the protocol that will be fired and deleted from Ω, or the creation of a new session that will be added to Ω.
  - E, E' denote sets of messages that represent the knowledge of the intruder, that is all messages that were sent on the network.
- Proposition [9] is an easy check that is completed automatically using COQ tactic.
- Proposition [11] is proved from [10] by induction on the length of the trace $\langle \tau_1 \dots \tau_n \rangle$, followed by an application of Propositions [5] and [9]. So, assuming that Proposition [10] is established, the proof of [11] is no more dependent of $\mathscr{P}, S, S_h, G_h, B_h$. Then, the proof steps have been recorded to be replayed as they are at each protocol verification.
- Eventually, Proposition [10] is the main proposition to proved. This proposition tells that the predicate $E\langle G_h, B_h\rangle S_h$ is stable with respect to any transitions of any sessions of the protocol $\mathscr{P}$. This is exactly the condition used in HERMES to conclude that it reached a fixpoint (S_h, G_h, B_h). Hence, the last computation step of HERMES that concludes to the stability of the predicate produces the arguments needed for the automatic proof of Proposition [10]. This proof is the only one that is specific to the protocol $\mathscr{P}$ and secrets S. It has been automated using tactics extracted from HERMES computation strategy. The tactic mimics the case that are considered by HERMES and recursion in HERMES computation corresponds to calls to the induction tactic in the proof.

  The certification feature of HERMES outputs a file for the COQ engine. It contains the theory of Figure 3 ; the proof of all propositions which are independent of $\mathscr{P}$ and S ; and some tactics dedicated to the specific proof of Proposition [10].

  When this file is loaded into the COQ engine, the proof-checker prints all the proved lemmas related to Proposition [10]. The proof is split in several lemmas: each one considers one transition of the protocol $\mathscr{P}$. Then, gathering all this lemmas leads to proposition [10].

  This proof cannot be done using a general tactic as for other propositions. Indeed, it depends on HERMES computation in the following way: COQ asks for a witness to prove some existential property that appear in the proof of Proposition [10]. The needed witness is recorded during HERMES run and a specific tactic is generated that provides the witness computed by HERMES at the right place in the proof.

Note that HERMES does not explicitly play a part in the proof. It is only used to compute a fixpoint (S_h, G_h, B_h) for a given protocol $\mathscr{P}$ and secrets S. Then, the property of this fixpoint and the conclusion it implies are checked by COQ. Hence, the proof is made independent of the HERMES theory and its implementation.

General theory for proving property of cryptographic protocol in Dolev-Yao's model

`type:`	*message type*	[1], `definition`
`def:`	$E\langle G,B\rangle S$	[2], `predicate definition using` [1]
`def:`	$E \vdash m$	[3], `predicate definition using` [1]
`def:`	$E \not\vdash S \stackrel{def}{=} \forall m.\ E \vdash m \Rightarrow m \notin S$	[4], `predicate definition using` [3]
`prop:`	$E\langle G,B\rangle S \Rightarrow E \not\vdash S$	[5], `proved` $\forall_{E,G,B,S}$

Theory specific to the protocol $\mathscr{P}$ and its secrets S.

`def:`	$(E,\Omega) \xrightarrow{\tau} (E',\Omega')$ *(for protocol transitions and session creation)*	[6], `predicate definition`
`def:`	$\mathscr{P}, S$ *extracted from the* LAEVA *specification*	[7], `definition using` [1]
`def:`	S_h, G_h, B_h *computed by* HERMES *for* $\mathscr{P}$ *and* S	[8], `definition using` [1]
`prop:`	$S \subseteq S_h$	[9], `proved`
`prop:`	$E\langle G_h,B_h\rangle S_h \Rightarrow \forall \tau \in \mathscr{P}.\ (E,\Omega) \xrightarrow{\tau} (E',\Omega') \Rightarrow E'\langle G_h,B_h\rangle S_h$	[10], `proved` $\forall_{E,E',\Omega,\Omega'}$
`prop:`	$E_0\langle G_h,B_h\rangle S_h \Rightarrow \forall \langle \tau_1 \ldots \tau_n\rangle \in \mathscr{P}^*.\ (E_0,\Omega) \xrightarrow{\tau_1} \cdots \xrightarrow{\tau_n} (E',\Omega') \Rightarrow E' \not\vdash S$	[11], `proved` $\forall_{E_0,E',\Omega,\Omega'}$

Figure 3: Proof in COQ certifying the correctness of HERMES results on protocol $\mathscr{P}$ and secrets S

2.2.2.7 Attack as Message Sequence Charts HERMES computations take place on the abstract model. So, it deals with abstract traces which collapsed all honest principals on the principal `H` and all intruders on `I` ; it also merges the nonces that are indistinguishable due to the abstraction. The abstract traces are produced by a backward computation ; they gives the essence of the attack but they forget about the messages that are needed to get an effective attack.

For a user that is not interested in HERMES principle, it is easier to get the attack on the concrete model illustrated as a message sequence chart (MSC). We developed a prototype that take a raw abstract attack given by HERMES and provides a concrete one: it computes the number of sessions, nonces and principals needed to get a concrete attack then it fills the abstract trace with the missing part of the protocol.

Example We consider the running example of Needham-Schroeder protocol specified in LAEVA in Section 2.2.1. From this specification, HERMES computes the abstract model described in Section 2.2.2.1, then the verification process leads to the following trace of an abstract attack:

$$!\{H, N_1^{HI}\}_{pbk(H)} \rightarrow ?\{H, n_1\}_{pbk(H)} \xrightarrow{\tau_1^2} !\{n_1, \widetilde{N_2^\star}\}_{pbk(H)} \rightarrow ?\{N_1^{HI}, n_2\}_{pbk(H)} \xrightarrow{\tau_2^1} !\{n_2\}_{pbk(I)}$$

where

- All nonces and roles are annotated by the session to which they belong. For instance, N^π is a nonce created for the session π ; $N^\star$ is a nonce of the distinguished session, denoted by $\star$ that involves two honest principals.
- H denotes the honest principal, and I denotes the intruder.
- N_1^{HI} is a constant that represents the nonce N_1 in a session between H and I.
- $N_2^\star$ is the second nonce of *the distinguished session* (denoted by $\star$) between two honest principals both represented by H. The secrecy property of Needham-Schroeder protocol states that $N_2^\star$ must remain secret.

The counter example generator concretizes and fills this trace to come to the Message Sequence Chart of Figure 4 where:

- horizontal arrows correspond to message passing between two roles and vertical arrows correspond to transition of a role.

- bold arrows come from the abstract trace given by HERMES; all the other ones (transitions, dashed and plain arrows) are added automatically by the process that reconstruct the concrete attack.
- dashed arrows do not correspond to actual communication; they denote how a message is interpreted (following the protocol) by the principal which receive the message
- (τ)-transitions are played in accordance with the protocol of Figure 1, ($\vdash$)-transitions denote computation of the intruder.

It can happen that HERMES notifies an attack which is not effective on the concrete model. These unprecise results are unavoidable: *false attack* are the so-called *inconclusive answer* in the abstract interpretation theory. They come necessarily with abstraction. If an attack detected by HERMES is not an actual attack in the concrete model, then:

- either the counter example generator will fail to compute the number of sessions needed, the user won't get a concrete attack. Then, the user has to examine HERMES trace to check if the abstract attack corresponds to an actual attack in the concrete model.
- or, it will produce an attack that actually cannot happen in the concrete model – this cannot be detected automatically since it is a problem known to be as hard as the verification problem itself.

The following table summarizes the results obtained by HERMES on protocols from [BFJM01] regarding secrecy properties. In the Table 1, "OK" means that the protocol has been successfully verified for the secrecy property ; "ATTACK" means that an attack have been found. In general, a third result, "INCONCLUSIVE" can be obtained because of abstraction step: it is possible to find an attack at abstract level which is not valid on the concrete level. Surprisingly we have not encountered any false attack on any practical protocol. Although, one could construct a protocol whose verification leads to a false attack.

Protocol Name	Result	Time (sec)
Yahalom	OK	12.67
Needham-Schroeder Public Key	ATTACK	0.01
Needham-Schroeder Public Key (with a key server)	ATTACK	0.90
Needham-Schroeder-Lowe	OK	0.02
Otway-Rees	OK*	0.02
Denny Sacco Key Distribution with Public Key	ATTACK	0.02
Wide Mouthed Frog (modified)	OK	0.01
Kao-Chow	OK	0.07
Neumann-Stubblebine	OK*	0.04
Needham-Schroeder Symmetric Key	ATTACK	0.04
ISO Symmetric Key One-Pass Unilateral Authentication	ATTACK	0.01
ISO Symmetric Key Two-Pass Unilateral Authentication	OK	0.01
Andrew Secure RPC	ATTACK	0.04
Woo and Lam	OK	0.06
Skeme	OK	0.06

* There is a known attack of the untyped version of the protocol. Discovering this type attack automatically requires to deal with non-atomic keys. This is not yet implemented in HERMES.

Table 1: The values in this figure have been obtained by running HERMES on a Pentium III 600MHz PC under Linux.

- π and $\star$ are session identifiers:
  - π is a session between one honest principal R_1^π and one dishonest principal R_2^π.
  - $\star$ is a session between two honest principals $R_1^\star$ and $R_2^\star$.
- All constants and variables are annotated (in exponent) with a session identifier. This indicates the session to which they belong.
- Every principal knows the public key $pbk(p)$ of any principal p.
- The secrecy property (S_{NS}) of Needham-Schroeder protocol specifies that $N_2^\star$, the nonce created by $R_2^\star$ in session $\star$, should only be known by $R_1^\star$ and $R_2^\star$.

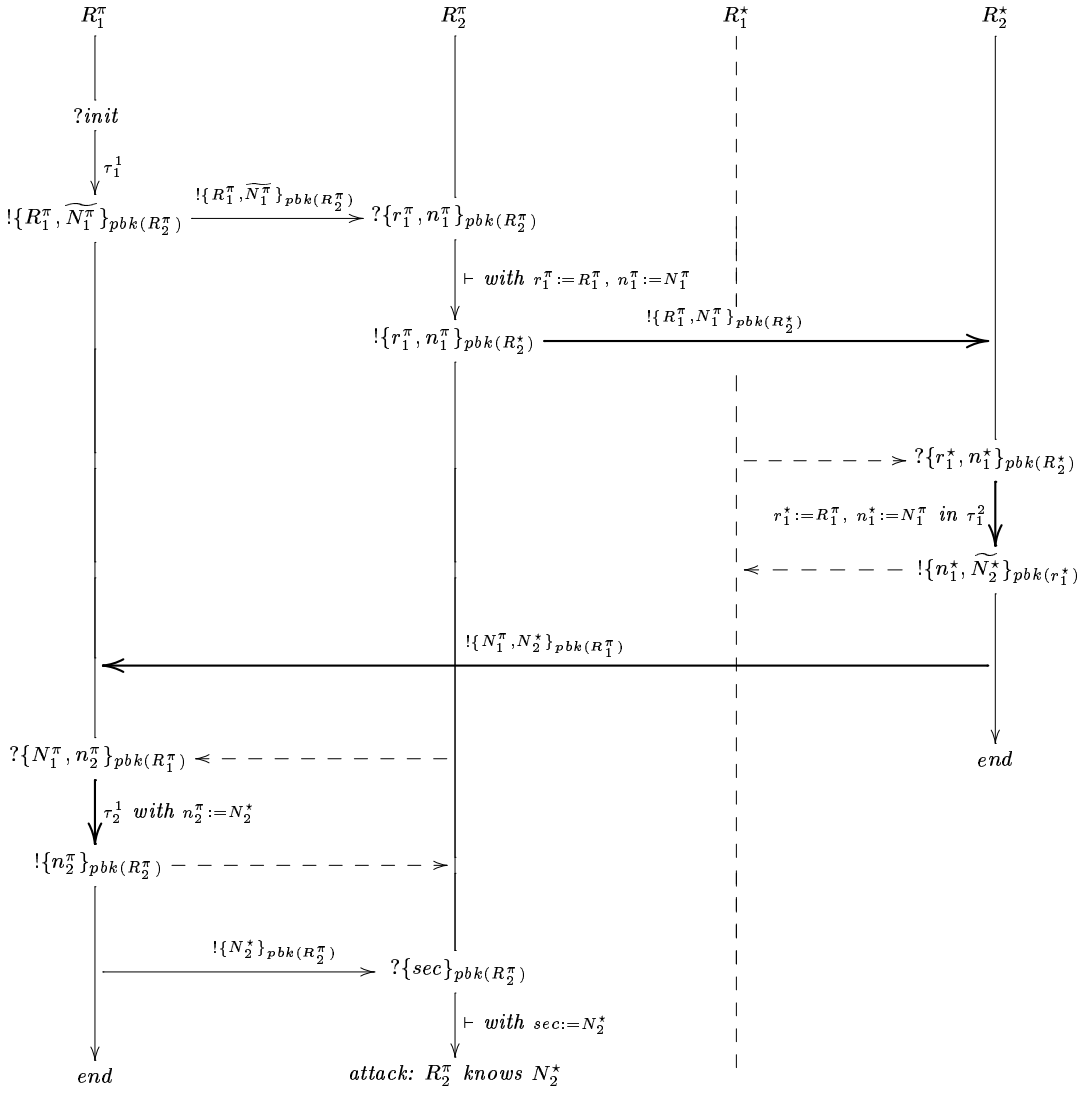


Figure 4: MSC describing the "man in the middle attack" of Needham-Schroeder protocol

2.2.3 SECURIFY

Two versions of SECURIFY are currently available. The first one always answers very quickly while the second one is able to prove the correctness of more protocols but is slower, even for protocols proved correct by the first version.

2.2.3.1 Input The SECURIFY input is a protocol specification written in the LAEVA language (see Section 2.2.1 for an example of such protocol).

The protocols handled by SECURIFY are typically those described in the Clark & Jacob survey [CJ97]. There are however some restrictions:

- Compound keys are not allowed. This implies that a message variable cannot be used in a key position;
- Long-term secret keys (like private keys or keys shared between an agent and a server) must not be used as plaintext;
- Long-term keys are either constant keys, or public or private agent's keys or else shared keys between an agent and a server.

2.2.3.2 Output The standard output includes:

- The translation of the protocol in the MR model (explained below).
- An answer to the following question : does the protocol satisfy its requirements?
- The computation time.

In addition, the correctness proof attempt is displayed in .ps and .pdf files.

SECURIFY first translates the protocol described in the concrete operational model (obtained from the LAEVA language) into the Millen & Rueß model, whose description can be found in [MR00, CMR01]. This translation is written on the standard output. For example, the output corresponding to the verification of the Needham-Schroeder-Lowe protocol is described in Figure 5.

This first output may help the user to verify that he obtained the wanted protocol.

The variables A_i stand for variables of sort agents, variables N_i for variables of sort nonces or keys, variables X_i stand for messages variables.

There are two kinds of events in the MR model: messages and spells. Spells are secrecy specification. For example, the spell `{Prv(A2),Prv(A1),N1}#{}` in the first rule of Figure 5 means that if the values `Prv(A2),Prv(A1),N1` are initially unknown to the intruder then they have to remain unknown to the intruder.

2.2.3.3 Proof principle The main proof technique is that of performing inductive proofs for secrecy invariants. Moreover, the whole secrecy proof is split into protocol-dependent and protocol-independent proofs. The protocol-dependent proof is in turn divided into cases associated with the protocol messages: SECURIFY verifies that none of the protocol's rules compromises a secret.

Three tests are performed for every rule and every part of a sent message:

- If the part is a fresh generated nonce,it cannot compromise any secret; it is denoted by the label `new`.
- If the part was already sent on the network, encrypted with at least the same set of keys, it is denoted by the label `almost_in`;
- If this part is a secret but is encrypted with a protected key, it is denoted by the label `db`.

```
            Formal protocol:
[
  [{Prv(A2),Prv(A1),N1}#{}] \\ If the values Prv(A2),Prv(A1) and
                               N1 are initially secret
  []                   \\ This field is used for fresh values
                          which are not wanted to be secret.
  [[[N1,A1]_Pub(A2)]] \\ Then the message [N1,A1]_Pub(A2)] is
                          added to the trace.
]
[
  [{Prv(A3),Prv(A4),N4}#{},[[N3,A3]_Pub(A4)]]
         \\ If the values Prv(A3),Prv(A4) and N4 are  initially
            secret and if the message [N3,A3]_Pub(A4)] has been sent
  []
  [[[N3,N4]_Pub(A3)]] \\ Then the message [N3,N4]_Pub(A3)] is
                          added to the trace
]
[
  [[[N1,N2]_Pub(A1)],{Prv(A2),Prv(A1),N1}#{},[[N1,A1]_Pub(A2)]]
  []
  [[[N2]_Pub(A2)]]
]
```

Figure 5: The Needham-Schroeder protocol expressed in the MR model.

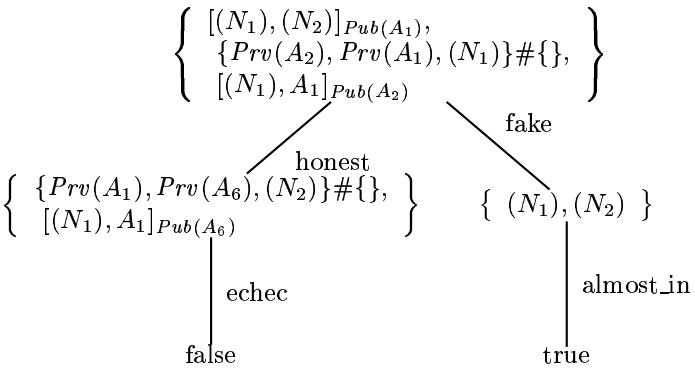


Figure 6: Tree proof attempt corresponding to the last rule of the protocol.

When none of these tests succeed, a backward search is done in order to obtain more information about the messages already sent on the network and the three tests are applied again. For example, let us consider what happens when we try to prove that revealing the nonce `N2` encrypted by the key `Pub(A2)` at the last step of the protocol does not compromised any secret (the tree proof attempt is displayed on figure 6). None of the three tests can be applied but we know that the message `[N1,N2]_Pub(A1)` must be in the trace. Thus, we distinguish two cases: either this message has been obtained be applying an honest rule (one of the rules of the protocol), either it has been built by the intruder. In the first case, looking at the rules, we deduce that the message `[N1,A1]_Pub(A6)` must also be in the trace and nothing else can be deduce: this induces a false node. In the second case, we deduce that the intruder must know the nonces `N1` and `N2`, thus we can apply our `almost_in` test: since no secret was compromised when the intruder knew `N2`, no secret is compromised by revealing `[N2]_Pub(A2)`.

A more complete information about the underlying theory may be found in [CMR01].

2.2.3.4 Tips The following hints are useful to SECURIFY newcomers

- The tool usually works better when messages are typed;
- For some protocols, the number of backward search may be infinite;
- In the web page, the number of backward search is limited to 10. In the full version, the maximal number of backward search is set by the user.

2.2.3.5 Building an attack When every leaf of the proof trees is labeled with `true`, SECURIFY answers "The protocol satisfies the requirements", this means that the protocol satisfies the security specification written in the eva file. When one of the leaves is labeled with `false`, the proof failed but it does not mean that the protocol does not satisfies its requirements, it only mean that the algorithm was not able to conclude.

However, in many cases, the proof tree attempts enable the user to build an real attack. Consider again the tree proof attempt displayed on figure 6. The construction starts at the leaf labelled with `false`. Its parent node contains a spell and a message and corresponds exactly the left-hand-side of the second rule of the protocol. Now, we consider a (partial) run of the protocol where all the rules preceding this one in the protocol description are applied in the "natural" order. In this case, there is only one rule, the first one :

$$A_1 \rightarrow A_6 : \{N_1, A_1\}_{\mathsf{pub}(A_6)}$$

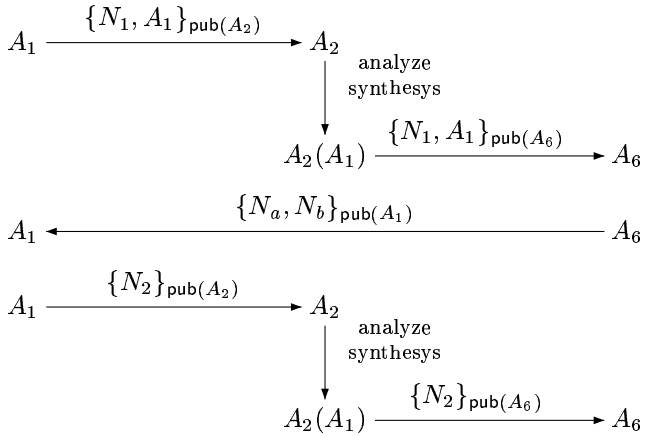
Next, we simulate an attack by following the branch from the `false` leaf up to its root. The parent node of the `false` leaf is directly connected with the root by an honest edge.

$$A_6 \rightarrow A_1 : \{N_1, N_2\}_{\mathsf{pub}(A_1)}$$

and N_2 should be secret between A_1 and A_6. Having reached the root of the tree, one applies the rule for which our algorithm fails.

$$A_1 \rightarrow A_2 : \{N_2\}_{\mathsf{pub}(A_2)}$$

A_1 sends the nonce N_2 to a wrong participant! Using this possibility, we can construct the following scenario, which corresponds to the man-in-the-middle attack:



2.2.3.6 Results The table 2 summarizes the results obtained by SECURIFY on protocols from [BFJM01]. "yes" means that a proof was found, "attack" means that the graphical output enables to find a real attack.

Protocols	Proof	maximal # of "back"	Time ms
Otway-Rees	Yes	0	0,35
Woo and Lam	Yes	0	0,11
Denning-Sacco	Attack	0	0,07
ISO Symmetric Key	Yes	0	0,15
Needham-Schroeder-Lowe	Yes	1	1,08
Needham-Schroeder	Attack	1	1,23
Wide-Mouthed-Frog (modified)	Yes	2	4,76
Kao-Chow	Yes	3	8,94
Yahalom	Yes	3	210

Table 2: The values in this figure have been obtained by running SECURIFY on a Pentium III 933 MHz,256 MB de RAM .

2.2.3.7 Second version of SECURIFY The first version of SECURIFY is not able to prove the correctness of protocols when a message of the form $\{n\}_k$ is sent, where n and k are secret date forged by *distinct* agents. An improvement has been proposed to be able to deal with such cases, by generalizing the `almost_in` test. This improvement has been implemented in the second version of SECURIFY. Using this enhancement, the second version of SECURIFY is now able to verify the Yahalom protocol for example. But since this version has to perform much more tests, it is also slower.

2.3 Documentation of the tools

The documentation of the tools is presented in the reports [BLP02b] and [Cor02, Cor03], which describe HERMES and SECURIFY respectively.

3 Experiments

Several experiments were conducted using the tools to verify properties of cryptographic protocols. Most of the verifications were performed on the LAEVA version of well-known algorithms which are already available on the web page of the project. Some of those algorithms are listed in what follows:

- The Needham Schroeder Public Key protocol. This is an algorithm for the mutual authentication of two principals in a public key environment. Actually, what was verified was a simplified specification of the algorithm with no trusted server and in which each principal is assumed to know the public key of the another one.
- The corrected version, by G. Lowe, of the previous algorithm.
- The Otway Rees protocol. Distribution of a fresh symmetric shared key by a trusted server.
- The Yahalom protocol. Distribution of a fresh symmetric shared key by a trusted server and mutual authentication.
- The MSR protocol. This is a hybrid protocol due to Beller, Chang and Yacobi [BCY93] (see also [BM98]) that uses a public key cryptosystem to establish a shared symmetric key between a mobile M and a base station B.

Both HERMES and SECURIFY have been designed so as to verify secrecy properties, that is to say, properties that state that certain nonces or keys interchanged during the execution of the protocol are not revealed to an intruder.

In what follows we review each experimentation stage in turn (protocol specification, protocol analysis, exploitation of negative results and exploitation of positive results) and we provide for each of them details concerning useful features of the tools and the limitations encountered. These limitations are further addressed in Section 4.

3.1 Protocol definition

Some protocol specifications were also edited from scratch. This was the case, for instance, of MSR. The specification of this protocol (and several variants of it) is included in the EVA test base [BFJM01]. However, its specification is not among those that are already available in the verification environment.

The original version of the protocol was flawed, it was not resistant against impersonation and freshness (replay) attacks. Three improved versions of the protocol are included in the test base, namely IMSR, IIMSR and I3MSR. The first of these specifications has been proposed to fix the flaw allowing for impersonation attacks. The other ones fix in addition the flaws allowing (some) freshness attacks.

In order to verify the various specifications they were first edited using a regular text editor and then entered in the verification window provided by the environment.

3.1.1 Useful features of the tools

The first observation concerns the specification language itself. One can express in LAEVA precise specifications of the algorithms obtaining at the same time formulations that remain quite close to their informal counterparts found in the literature. The language provides constructs whose syntax is quite informative and powerful enough to produce concise and readable specifications. The specifications which are provided in the verification environment of the EVA web page allow a beginner to try the verification environment and in addition they serve to give the user a first impression of the language.

3.1.2 Limitations encountered

The only reference to LAEVA provided in the verification environment is that of the report [JM01], which is essentially a technical presentation of the language. Some kind of tutorial would obviously be welcome for a wider usage of the tools.

So far as expressiveness is concerned, the attempt to verify the MSR protocol allowed us to identify an area for extension of LAEVA, in order to state the honesty of a principal if it is not involved in the execution of the protocol.

3.2 Protocol analysis

All the protocol specifications described above were used for verification using both HERMES and SECURIFY. The following sections provide first a general assesment of the working of the tools and then a detailed account of the verification of the MSR protocol.

3.2.1 Useful features of the tools

Both tools managed to detect in most of the cases the known flaws and to accept the protocols which satisfied the stated secrecy property.

The tools provide as output of the run information to understand why an attack is detected and how it can be produced. A user familiar with the technical notions upon which the corresponding verification algorithms are built up is provided with the necessary elements to find out where the problem lies.

The performance of both tools is quite remarkable, as can be seen in Tables 1 and 2.

3.2.2 The original version of MSR

As already mentioned the original version of the MSR protocol was flawed. The literature documents well-known (impersonation and freshness) attacks to the protocol. These attacks are explained below. The following is the corresponding LAEVA specification of the original protocol, which has been formulated using the concrete syntax accepted by the EVA verification tools.

```
Original Version (Infinite sessions)
        MSR
        msr : asym_algo
        everybody knows msr
        B, M, CA : principal
        b, m : principal
        certM : number
        basetype key
        K : key
        keypair^msr PK, SK (principal)
        alias certM = { CA, M, PK(M) }_SK(CA)^msr
        B knows B, M, PK(B), SK(B), CA, PK(CA), msr
        M knows M, PK(M), SK(M), certM, CA, PK(CA), msr
        {
        1. B -> M : B, PK(B)
        2. M -> B : { K }_PK(B)^msr, { M, certM }_K
        }
        s. session* M=m, B=b
        assume secret (SK(M)@s.M),
        secret (SK(B)@s.B)
        claim secret(K@s.M)
```

The following are the informal formulations of the attacks that can be carried out against this protocol. They are presented and discussed in [Car94] and in [BFJM01]:

- Attack 1. There is no certificate for B's public key PK(B)in the first message, hence it is possible for an intruder to impersonate B.
- Attack 2. Replay (freshness)
  - The intruder may replay an old second message from a previous session of the protocol, impersonating B. In this attack, K is not a fresh key but the key exchanged in the previous session.

– If an old key K, has been compromised (in a previous session), then the intruder is able (masquerading M) to forge and distribute a new shared key K (because he knows M and CertM).

3.2.3 Verifying MSR using HERMES

HERMES manages to detect the attacks described above. It also provides indications on how the attacks can be carried out, by pointing out the rule of the protocol which makes the attack possible and by describing (in terms of a derivation tree) the sequence of rule applications that lead to the attack. All the information is expressed in terms of the abstract rules that are automatically derived out of the specification of the protocol.

3.2.3.1 Finite sessions In order to ease the interpretation of the results of the verification a finite session variant of the protocol was specified. One such specification typically gives raise to a smaller number of abstract rules and in addition the principals involved in those rules are exactly those specified by the sessions definitions.

This version was obtained by replacing the declarations:

```
s. session* M=m, B=b
assume secret (SK(M)@s.M),
secret (SK(B)@s.B)
claim secret(K@s.M)
```

by

```
s1. session M=m, B=b
s2. session M=m, B=I
assume secret (SK(B)@s1.B),
secret (SK(M)@s1.M)
claim secret(K@s1.M), secret(K@s1.B)
```

and by introducing the "new" principal I (the intruder). Notice that this alternative declaration makes use of the knowledge that attacks are perpetrated by the intruder by impersonating the principal B.

3.2.3.2 Limitations A first problem when trying to interpret the outcomes of a HERMES run is that this information is derived from the abstract version of the algorithm. Thus, for a user that is not aware of how the abstraction mechanism works, the answer provided by the tool may be difficult to interpret. Moreover, if the verification algorithm is run on a specification of the protocol that establishes an infinite session scenario, the amount of associated abstract rules renders quite difficult the understanding of the results.

3.2.4 Verifying MSR using SECURIFY

The current version of SECURIFY has some limitations for protocols with exchanges of public keys, like MSR. The public keys have to be publicly known at the beginning of the protocol. This limitation comes from the encryption of a message with certain variable X (which is actually an abstraction of a public key used in the protocol). At some stage, SECURIFY needs to compute the inverse key of X, which is not possible. Encryption with arbitrary messages is not supported by SECURIFY.

THis limitation can easily be circumvected though. A different version of MSR was defined in which a declaration of the form "everybody knows PK" has been added, where PK is the public key in question. SECURIFY returns a failure when attempting to prove the correction of the protocol. The protocol is indeed not secure because, as already explained, the key in the first message is not certified.

3.3 Exploitation of negative results

A typical example of a flawed protocol is the one known as the Needham Schroeder Public Key protocol [NS78]. This is an algorithm that has been proposed for the mutual authentication of two principals in a public key environment. The LAEVA specification of this protocol is available in the verification environment provided in the web page of the EVA project. The property to be verified by the algorithm concerns the nonces that are exchanged during a protocol session.

That specification was fed to both HERMES and SECURIFY, which detected and signaled the freshness flaw of the algorithm immediately.

3.3.1 Useful features of the tools

Both tools provide the user with enough information to understand how an attack, also known as "man in the middle", can be carried out.

In addition to displaying the abstract formulation of the protocol rules, HERMES displays the (abstraction of the) dangerous messages, that is to say, messages that the algorithm deemed to bet secrets, but that do not conform to the definition of such notion. HERMES also provides the rules that can be used to perform the attack. Last but not least, an execution trace illustrating the attack is also made available in the form of a postscript file.

As to SECURIFY, it also displays the abstract formulation of the protocol rules. In addition it provided the tree depicting the failed proof attempt generated while proving that the protocol satisfies the secrecy requirements. This failed proof attempt can be used to show, or to get convinced, that the protocol does not satisfy the secrecy property in question.

3.3.2 Limitations encountered

The information provided so far is very useful for experts but it could still be improved to be more easily understood by a non-expert.

3.4 Exploitation of positive results

The revised version by G. Lowe of Needham Schroeder as well as the Otway Rees and Yahalom protocols have all been verified to satisfy the corresponding secrecy properties.

3.4.1 Useful features of the tools

A very interesting contribution of EVA is the work that has been developed by the HERMES team towards the automatic generation of proofs in the formal setting provided by the proof-assistant COQ. This certification of a HERMES (successful) run has been explained in Section 2.2.2.6 (Certifying HERMES computation).

3.4.2 Limitations encountered

The information provided so far is very useful for experts but it could still be improved to be more easily understood by a non-expert.

As in the case of negative results, the work presented in [CMR01] concerning the justification of a successful verification of a protocol by SECURIFY can serve as the basis of a set of guidelines for understanding a successful proof tree.

4 Suggestions for further work

4.1 Research issues

Several areas for further research on HERMES and SECURIFY have been identified. They are summarized in what follows.

4.1.1 HERMES

In the case that the correctness of a protocol is verified the verification algorithm of HERMES provides as output a tree that can be exploited for certification. The designers of the tool are currently working on the development of mechanisms for extracting out of that tree a proof term that can in turn be verified using the proof assistant COQ.

The method implemented in HERMES for verifying secrecy properties of cryptographic protocols can be extended to also consider non-atomic keys and xor-encryption. To further investigate these issues is a first step towards the automatic verification of algebraic properties of encryption algorithms.

4.1.2 SECURIFY

SECURIFY may be improved in three directions.

First, the proof technique used by SECURIFY can be improved by refining the proof criterion like it was done for the second version of the tool. New improvements may be discovered while testing SECURIFY on new protocols, understanding why SECURIFY fails to prove some correct protocols.

In addition, new research results [CLC03, CLS03, CKRT03b, CKRT03a] have shed some light on the algebraic properties used for encryption, like, for instance, the exclusive or and the modular exponential. It seems possible to extend the SECURIFY's algorithm in order to take into account the properties of such operators.

Finally, SECURIFY's output is not totally satisfactory when it does not find a proof. Indeed, in this case, the user has to analyze the tree proof by himself, in order to understand why the proof failed and try to reconstruct an attack. In several cases, it seems possible to automatically build attack scenarios. Since the proof technique is incomplete (the problem being undecidable), it is impossible to obtain an attack scenario in every case. In practice it may be very useful, though.

4.2 Practical issues

The suggestions for improvements of the tools are structured following the usage stages introduced in Section 3.

4.2.1 Protocol definition

The development of a simple on-line manual of the LAEVA language would be in order if the tools were to be used by outsiders. Decorating the specifications, both in the verification environment and in the report [BFJM01], with informative comments would also be helpful. This would constitute a quick and direct source of explanation of the syntax (and intended meaning) of the language constructs.

The following are hints to further work concerning the facilities provided by the verification tools:

- The display and edition facilities of the EVA's test base specifications on one side and that of the specifications entered by the user on the other side should be independent. It might be useful for a user to be able to inspect the available specifications while entering one of his own.
- A user should be able to save specifications that he has managed to verify. In addition, it would also be interesting to provide facilities for managing centralized user's test base.

4.2.2 Protocol analysis

The presentation of the output of a run of HERMES could be improved as follows:

- When an attack is detected, information is provided by HERMES concerning the kind of messages that are "dangerous" as well as the rules that could be used to carry out the

attack. Just adding some text explaining how that information should be understood would be quite helpful to the user.

- The presentation of the postscript illustrating how a detected attack can be carried out should also be improved.Suggestions for improvements are provided in Sections 4.2.3 and 4.2.4 below.

As to SECURIFY, the file provided in the output of a run depicting the proof tree attempt generated while proving that the protocol satisfies the secrecy requirements would be more useful with a synthetic and intuitive description of the conceptual structure of such tree. For instance, it should be explained what a branch of that tree represents as well as what is the meaning of the (search associated) labels attached to each branch.

4.2.3 Exploitation of negative results

Further work has to be done concerning the elaboration of intuitive and informal explanations of how an attack can be constructed from a failed proof attempt.

In the case of HERMES, for instance, it would be very helpful to provide a clear characterization of what it means that an abstract pattern of a message does not conform to the definition of being an adequate secret. It is not always obvious to realize why the patterns detected by the tool are conflicting ones.

In the case of SECURIFY, a proposal for improving the presentation of attack scenarios was discussed in Section 4.1.

4.2.4 Exploitation of positive results

In what follows are listed some of the issues that were identified during the experimentation. Solutions to the needs below were not expected to be addressed in the project, due to its exploratory nature, but they might well constitute avenues for further work.

In the first place, a precise definition of all implicit assumptions underlying the verification framework (e.g. well-typing or not, perfect cryptography, trusted and untrusted parties, potential behavior of actor, etc.) would be very useful. Without such definition, non-experts are bound to be misled by the results of the tools. On the other hand, a precise definition of such assumptions could also be a contribution in itself because different research groups typically take slightly different sets of assumptions, without necessarily evaluating the impact of such variations. With respect to the Common Criteria, some of these assumptions may correspond to *organizational security policies* [CC99].

It would also be quite helpful to provide a way to identify the assumptions used for one particular proof and allow the user to play with them. The main benefit of a proof is not to establish a definite statement of truth but to establish links between assumptions and conclusions. This is especially the case in our context since security is obviously not an absolute judgment. It would thus be of prime importance for users of the verification tools to be able to vary the assumptions (e.g. trust assumptions) and to understand the consequences of such variations, or to track such implications in the original proof (traceability links).

The ultimate goal of a proof in the context of evaluations is to improve the confidence of a third party in the security of a product. Such goal cannot be reached through a bare yes or no answer: the user should be able to understand (at least in very general terms) the rational behind the proof. This problem is very complex in itself and goes even beyond research in verification. It involves a number of challenges, including:

- The presentation in a unified language of a composite proof involving different techniques such as abstraction, model checking, inductive proofs, etc. Ideally, as opposed to the general assumptions of the verification framework referred to in the first item, such language should not be tied to one specific tool but rather be as close as possible to a universal (or neutral) logic language.

- The introduction of a link with natural language explanations. Such a link would provide a necessary complement to the previous item for potential users without the required mathematical background. In order to ensure their consistency with the formal proofs, such explanations should ideally be extracted from the proof (semi-formal paraphrasing language).

- The derivation of an abstract version of a proof from which all second order details have been filtered out in order to provide a quick overview of the rational (or general structure) of the proof to the user. Ideally, the user should then be able to explore any part of the proof for closer examination (proof unfolding).

- The derivation of "positive examples" showing how potential attacks would be defeated by the system. This would be the counterpart of the "counter- examples" resulting from negative proofs. Formally speaking, such positive examples could be specific instances or cases of the general proof : they could thus be seen as illustrative sub-proofs providing hints to the users about the reason why specific scenarios or threats are not possible. Another form of positive examples, which could be introduced without much modifications of the tools, could take the form of counter-examples of "false properties" provided by the user to test his own guesses and thus improve his understanding and confidence in the overall proof provided by the tools.

5 Conclusions

The main objective of the EVA project has been the design and development of tools for the definition and automatic verification of cryptographic algorithms. In the context of the project three different tools have been developed:

- CPV, which combines abstraction and automatic theorem proving techniques inspired by research on tree automata.

- HERMES, which relies on well-grounded techniques for the symbolic verification of cryptographic protocols.

- SECURIFY, which exploits techniques concerning the inductive proof of secrecy invariants.

This document reports experiments that have been carried out using two of the tools, namely HERMES and SECURIFY. The experiments were performed using a verification environment which was developed in the context of the project. This environment consists of an interface that provides the basics functionalities to run the verifications on a LAEVA file. This file can either be edited on-line or chosen from a database of protocol specifications which is available to the user.

The input for each of the tools consists of the (same) representation of a concrete operational model that embodies the definition of a protocol, some assumptions concerning the protocol and a security property to be verified. The representations provided to the tools are the result of the compilation of a specification, expressed in LAEVA, of the protocol and the property to be checked. LAEVA is the high level specification language designed in the EVA project for describing cryptographic protocols and stating the secrecy properties that a protocol should satisfy.

Among the main contributions of the project, let us stress the following:

- On the formal side: the LAEVA specification language was defined independently of the techniques that have been used to develop the verification tools. As a result, the three tools accept the same input, which is of great benefit to better understand the essence of security properties and subtleties in their treatments. It is often the case indeed, that verification tools reported in the literature work on their own input language, which makes comparisons and factorization of work more difficult.

- On the practical side: the tools complement each other in diferent ways:

- The verification results may vary, one tool providing an affirmative result when the other is unable to produce a conclusive response, depending on the kind of protocol being verified. The range of protocols covered by the tools altogether is thus strictly greater than the one covered by each of the tools independently.
- They provide different facilities concerning protocol analysis. Results of (non successful) verification runs, for instance, are presented by one tool in terms of derivation trees identifying the rules leading to an attack and by another tool as a tree representing a proof attempt failure.
- SECURIFY usually works more quickly than HERMES on typed protocols while HERMES is more appropriate for untyped protocols. SECURIFY also supports keys as basic keys rather than plain numbers. The negative result analysis is also more accurate in this case. The method implemented in HERMES makes it possible to distinguish between long term and short term keys. This allows a more faithful modeling of some protocols.

As far as proof explanation is concerned, it would be useful to provide the user with an explicit formulation of the assumptions that are implicit in the underlying verification framework. The definition of a common language to express these assumptions seems to be an adequate solution which in addition would benefit from the work carried out concerning the elaboration and definition of LAEVA.

The connections between the proof evidence provided by the tools, in particular HERMES, and proofs that can be automatically verified by a general theorem prover like COQ also contributes to find precise and explicit ways of expressing the complete setting on which the verification framework relies.

In the context of the EVA project a program analysis tool was also developed that allows to detect inconsistencies between the abstract model of a protocol defined in LAEVA and the Java code that is supposed to implement the protocol. This tool constitutes the result of a successful effort for relating the abstract definitions of protocols considered in the project with their real implementation counterparts. It also enhances the industrial relevance of the project.

References

[BCY93] M. Beller, L. Chang, and Y. Yacobi. Privacy and authentication on a portable communications system. *IEEE J. Selected Areas in Communications*, 11(6):821–829, 1993.

[BEL03a] L. Bozga, C. Ene, and Y. Lakhnech. A Symbolic Calculus for Cryptographic Protocols. Technical Report 11, EVA, november 2003.

[BEL03b] L. Bozga, C. Ene, and Y. Lakhnech. Symbolic Verification of Cryptographic Protocols with Time Stamps. Technical Report 12, EVA, november 2003.

[BFJM01] D. Bolignano, F. Fiorenza, F. Jacquemard, and D. Le Métayer. EVA test base. Technical Report 4, EVA, november 2001.

[BLP02a] L. Bozga, Y. Lakhnech, and M. Périn. Abstract Interpretation for Secrecy using Patterns. Technical Report 5, EVA, november 2002.

[BLP02b] L. Bozga, Y. Lakhnech, and M. Périn. L'outil de vérification HERMES. Technical Report 6, EVA, may 2002.

[BM98] C. Boyd and A. Mathuria. Key establishment protocols for secure mobile communications: A selective survey. In *Information Security and Privacy*, volume 1438 of *LNCS*, pages 344–355. Springer-Verlag, 1998.

[Car94] U. Carlsen. Optimal privacy and authentication on a portable communications system. *Operating Systems Review*, 28(3):16–23, July 1994.

[CC99] Common Criteria for Information Technology Security Evaluation. Part 1: Introduction and general model , August 1999. Version 2.1.

[CJ97] John Clark and Jeremy Jacob. A survey of authentication protocol literature : Version 1.0., November 1997.

[CKRT03a] Y. Chevalier, R. Kuester, M. Rusinowitch, and M. Turani. Deciding the security of protocols with diffie-hellman exponentiation and products in exponents. In *FST-TCS 03*, LNCS, 2003.

[CKRT03b] Y. Chevalier, R. Kuester, M. Rusinowitch, and M. Turani. An NP decision procedure for protocol insecurity with xor. In *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science (LICS'03)*, 2003.

[CLC03] H. Comon-Lundh and V. Cortier. New decidability results for fragments of first-order logic and application to cryptographic protocols. In *Proceedings of the 14th Int. Conf. on Rewriting Techniques and Applications (RTA'2003), to appear*, Lecture Notes in Computer Science, Valencia, Spain, June 2003. Springer-Verlag.

[CLS03] H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science (LICS'03)*, pages 271–280, 2003.

[CMR01] V. Cortier, J. Millen, and H. Rueß. Proving secrecy is easy enough. In *14th. IEEE Computer Security Foundations Workshop*, 2001.

[Cor02] Véronique Cortier. L'outil de vérification SECURIFY. Technical Report 7, EVA, may 2002.

[Cor03] Véronique Cortier. A guide for SECURIFY. Technical Report 13, EVA, november 2003.

[GL01] Jean Goubault-Larrecq. Langage de spécification de protocoles cryptographiques de EVA: syntaxe abstraite et sémantique. Technical Report 2, EVA, november 2001.

[GL02] Jean Goubault-Larrecq. Outils CPV et CPV2. Technical Report 8, EVA, may 2002.

[Jac03] Florent Jacquemard. The EVA translator. Technical Report 9, EVA, july 2003.

[JM01] F. Jacquemard and D. Le Métayer. Langage de spécification de protocoles cryptographiques de EVA: syntaxe concrète. Technical Report 1, EVA, november 2001.

[Low95] Gavin Lowe. An Attack on the Needham-Schroeder Public Key Authentication Protocol. *Information Processing Letters*, 56(3):131–136, november 1995.

[MR00] J. Millen and H. Rueß. Protocol-independent secrecy. In *Proceedings of the 21st Symposium on Research in Security and Privacy (RSP'00)*. IEEE Computer Society Press, 2000.

[NS78] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12), December 1978.