A Realistic Environment for Crypto-Protocol Analyses by ASMs

Giampaolo Bella[1] and Elvinia Riccobene[2]

[1] Computer Laboratory, University of Cambridge, Cambridge CB2 3QG (UK)
Giampaolo.Bella@cl.cam.ac.uk
[2] Dipartimento di Matematica, Università di Catania, ITALY
riccobene@dipmat.unict.it

Abstract. The ASM formalism is tailored to develop a general, realistic environment — a computer network possibly monitored by an active eavesdropper — on which any crypto-protocols can be faithfully analysed. The well known Needham-Schroeder protocol with public keys is considered as a case study, and is proven to establish secure sessions between honest agents. The eavesdropper model can reproduce Lowe's attack, and Lowe's correction to the protocol is shown to be effective. ASM proofs are based on induction, and can be carried out without automated tools.

1 Introduction

Crypto-protocols are well known to be incredibly error prone. Although the use of formal methods to analyse them has become common practice during the current decade, the notion of *correctness* of a crypto-protocol is still open ground for research.

State enumeration tools search for flaws (i.e. breach of security) by the exhaustive visit of all states reachable by the model. Despite the necessity of drastic simplifying assumptions to keep the state space at a manageable size, this approach has achieved brilliant results [6, 14]. Chief among them, the discovery of a flaw by Lowe [9] on the public-key version of the Needham-Schroeder protocol, after two decades from its publication [11]. However, failure to discover flaws does not necessarily mean that a protocol is correct w.r.t. its goals.

A suitable *belief logic* is defined by Burrows et al. [4] to express correctness in terms of the ability of parties to authenticate to each other, and to agree on certain information. Proofs are short and abstract. Although reasoning about secrecy is not allowed, this logic is considered a milestone in the field, and has been extended in several directions (e.g.[3]).

The simple observation that a crypto-protocol attempts to keep certain properties true during its execution triggered the intuition that reasoning by *induction* could help a formal analysis in this field. Meadows uses state enumeration with inductive features [10], Paulson develops a purely inductive method based on theorem proving [12], and Fábrega et al. more recently propose the only

(non-straightforward) application of induction by hand [5]. Therefore, it seems widely accepted that induction is suitable to formally specify the goals of a crypto-protocol.

We believe that the little formal overhead of ASM models can galvanise the potentialities of induction. This is supported by the complete analysis of the Kerberos Authentication System [2], which verifies a possible attack by the eavesdropper of the network (called *spy* below).

We present a general ASM model of spy that is independent from the protocol being analysed. The model poses no limit to the spy's knowledge, thus exceeding the limits of state enumeration, and is plainly inspired by Paulson's, acknowledged by many as the most expressive. However, our spy can save the nonces learnt from the traffic, and invent new ones directly.

The network traffic is modelled as a set of messages that agents can extend by sending new messages, and restrict by deleting the messages they receive. We believe that this representation is particularly close to the real world.

The ASM formalism proves to be easily adapted to the analysis of crypto-protocols. The modularisation techniques clarify the entire treatment. The features of information hiding make the spy model more intuitive. The proofs can be performed without mechanised support, and are intelligible to humans.

Section 2 presents an ASM reminder. Section 3 describes the general environment for the analysis of crypto-protocols. Section 4 considers the public-key version of the Needham-Schroeder protocol as a case study. We prove that both nonces remain secure from the spy in a run between honest agents, as we believe that this kind of *positive* guarantees lack formal treatment. Then, we prove that our model can reproduce Lowe's attack, and that Lowe's correction to the protocol is effective. Related work is illustrated in Sect.5, and Sect.6 concludes.

2 ASM Sketch

The proposed model is a distributed Gurevich's Abstract State Machine. We assume that the reader is familiar with the semantics of the Abstract State Machine defined in [7] and updated in [8]. We point out the following rule constructors

- `do forall` u `in` U [`and` v `in` V] abbreviates `do forall` u[, v]: $u \in U$ [`and` $v \in V$];
- `do in-par` $R_1, \ldots, R_n$ `end-par` states parallel execution of rules $R_1, \ldots, R_n$;
- `choose` $V \subset U$ abbreviates `choose` $v_1, \ldots, v_{|V|}$: $v_1, \ldots, v_{|V|} \in U$;

R	R
`end-choose`	`end-choose`

  if V is the empty set, then the rule R never fires.

Recall that a function in the signature of an agent A is *controlled* or *monitored* when it is directly updatable by and only by the rules of A, or respectively by the environment of A. Monitored functions may be thought as oracles.

3 Modelling a Crypto-Protocol Environment

A *crypto-protocol* is a sequence of messages between two or more parties, in which encryption is used to provide authentication, and/or to distribute cryptographic keys for new conversations.

The *environment* of a crypto-protocol is a computer *network* that might be monitored by a *spy*. We consider the worst scenario, and model a network under total control of an active spy. Note that this model can be used for the analysis of any crypto-protocols.

3.1 A Network Model

This section presents a general model for the network on which crypto-protocols are executed.

Define $AGENT$ as the set of network agents. Agents may send *nonces* inside messages. These are random numbers used to identify a single protocol run. Define the set $NONCE$ of nonces, and the monitored function

$$nonce : AGENT \times AGENT \longrightarrow NONCE$$

that any agent invokes to generate a random nonce during an instance of the protocol with another agent. State a *non-clashing constraint* on it, to formalise that the probability of generating the same nonce at different invocations may be considered zero (Lowe, Meadows, and Paulson make the same assumption [9, 10, 12]). Each agent saves the nonces he has generated into a private set, image of the controlled function

$$Nonce : AGENT \longrightarrow \wp(NONCE)$$

$\wp$ being the powerset operator. Agents keep track of the agent to whom they have associated a certain nonce by the controlled function

$$recipient : NONCE \longrightarrow AGENT$$

In public-key protocols each agent has a public key to encrypt messages, and a private key to decrypt them. The set $PUBKEY$ denotes all agents' public keys, and the set $PRIKEY$ denotes all agents' private keys. The bijective function

$$inv : PUBKEY \longrightarrow PRIKEY$$

associates a public key to its private inverse. For simplicity, given $k \in PUBKEY$, $inv(k)$ is denoted by k^{-1}. The fact that k and k^{-1} are inverse of one another can be formalised straightforwardly. The functions

$$pubkey : AGENT \longrightarrow PUBKEY$$
$$prikey : AGENT \longrightarrow PRIKEY$$

associates an agent to his public or private key respectively. Given $X \in AGENT$, we denote $pubkey(X)$ by K_x, and $prikey(X)$ by K_x^{-1}. In shared-key protocols, each agent has a unique key, so $PUBKEY$ and $PRIKEY$ denote the same set, inv is the identical function, $pubkey$ and $prikey$ are the same function associating an agent to his key.

Each message, encrypted or in clear-text, may contain agent's names, nonces, timestemps, session keys. Define $TIMESTAMP$ as the set of timestamps, $SESKEY$ as the set of session keys, $COMPONENTS \triangleq AGENT \cup NONCE \cup TIMESTAMP \cup SESKEY$ as the set of message components.

MESSAGE is intuitively the set of messages that can be built by concatenation and/or encryption of elements of *COMPONENTS*. More formally, let $CONC \triangleq \bigcup_{i=1}^{\infty} (COMPONENTS)^i$ be the set of messages obtained by unlimited concatenation of the basic components, and $CRYPT \triangleq CONC \times PUBKEY$ be the set containing the messages encrypted under public keys, thus

$$MESSAGE \triangleq CONC \cup CRYPT$$

To improve readability, an element of *CONC* is denoted under the form $\{comp_1, \ldots, comp_n\}$, and an element of *CRYPT* under the form $\{comp_1, \ldots, comp_n\}_k$, $k \in PUBKEY$. Note that also single-element messages belong to *CONC*.

Define *TRAFFIC* as the set of all information sent over the network. It is a big container any agents can extend by sending messages through the macro

$$send\ Sender.Recipient.msg \equiv \texttt{Extend}\ TRAFFIC\ \ \texttt{with}\ M$$
$$val(M) := Sender.Recipient.msg$$

with *Sender*, *Recipient* $\in$ *AGENT*, and *msg* $\in$ *MESSAGE* [1]. A traffic element is denoted by the result of *val* on it. We make no distinction between a traffic item and its value, so that *TRAFFIC* can be identified with the image of the function *val* on it. Define

$$cont : TRAFFIC \longrightarrow MESSAGE$$

so that $cont(Sender.Recipient.msg) = msg$. The image of the function *cont* is denoted by $cont(TRAFFIC)$. Note that *TRAFFIC* must be allowed to contain more occurrences of a message because of the spy's activity (see Sect.3.2).

3.2 A Spy Model

The spy's illegal activity is twofold: altering information on the traffic and saving the nonces that can be learnt. Altering information consists either in sending new, fake information or in destroying information retrieved from the traffic. This is formalised abstractly in Fig.1.

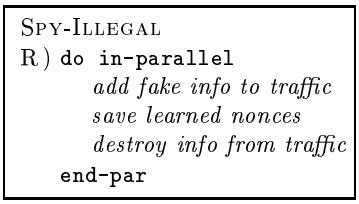

Fig. 1. Spy's illegal activity (abstract model)

The spy can also act as an honest agent — $Spy \in AGENT$, $pubkey(Spy) \in PUBKEY$, $prikey(Spy) \in PRIKEY$ — so the submodule in Fig.1 will be extended by the submodule performing the actual protocol steps. This enphasises the distinction between the spy's illegal activity, independent from the protocol, and the legal activity in accordance with the rules of the protocol.

[1] The dot notation in *Sender.Recipient.msg* is used to concatenate items.

COMPR is the subset of *AGENT* containing the agents whose private keys are compromised to the spy (they have possibly conjured with the spy). *KEYSPY* is the subset of *PRIKEY* containing the private keys of compromised agents, namely items of the form $k^{-1} = prikey(A)$, with $A \in COMPR$. It is convenient to state that $Spy \in COMPR$.

In order to refine the abstract specification of the spy's illegal activity, define the functions

$$analz : \wp(MESSAGE) \times \wp(PRIKEY) \longrightarrow \wp(COMPONENTS)$$

$$\overline{analz} : \wp(MESSAGE) \times \wp(PRIKEY) \longrightarrow \wp(MESSAGE)$$

Let $M \subseteq MESSAGE$, and $K \subseteq PUBKEY$; $analz(M,K)$ yields the collection of all components of messages of any length in M that are decryptable by a key in K. More formally

$$\frac{n \in \mathbb{N} \qquad \{comp_1, \ldots, comp_n\}_k \in M \qquad k \in K}{comp_i \in analz(M,K) \;\; i = 1, \ldots, n}$$

Let $\hat{M}$ be the subset of M containing all messages that can be decrypted by some keys in K — i.e. $analz(\hat{M},K) = analz(M,K)$. Define $\overline{analz}(M,K) \triangleq M \setminus \hat{M}$. In other words, $analz(M,K)$ denotes what can be *analysed* from M using K, and $\overline{analz}(M,K)$ denotes everything that remains unintelligible.

To formalise the synthesis of new messages, define the function

$$synth : \wp(COMPONENTS) \times \wp(PRIKEY) \longrightarrow \wp(MESSAGE)$$

Let $C \subseteq COMPONENTS$, and $K \subseteq PUBKEY$; $synth(C,K)$ yields all the messages that can be built by concatenation of elements of C, and/or encryption under keys in K. More formally

$$\frac{n \in \mathbb{N} \qquad comp_i \in C \;\; i = 1, \ldots, n}{\{comp_1, \ldots, comp_i\} \in synth(C,K) \;\; i = 1, \ldots, n}$$

$$\frac{n \in \mathbb{N} \qquad comp_i \in C \;\; i = 1, \ldots, n \qquad k \in K}{\{comp_1, \ldots, comp_i\}_k \in synth(C,K) \;\; i = 1, \ldots, n}$$

Therefore, $analz(cont(TRAFFIC), KEYSPY)$ contains the message components that the spy can analyse from the traffic by decryption of the messages that are encypted under the public keys of compromised agents, or under his own. In particular, the set contains the nonces that the spy can learn from the traffic. Conversely, $\overline{analz}(cont(TRAFFIC), KEYSPY)$ contains the messages that the spy intercepts without understanding, and can simply reply as they are.

The illegal messages that the spy can introduce on the network comprise replied messages, and new fake messages

$$FakeMsg \equiv$$
$$\overline{analz}(cont(TRAFFIC), KEYSPY) \cup$$
$$synth(analz\,(cont(TRAFFIC), KEYSPY) \cup Nonce(SPY) \cup AGENT, \; KEYSPY\,)$$

Note that the new messages might contain the nonces learnt from the previous traffic, and also agent names not hitherto involved. The spy's illegal actions are refined in Fig.2. It is understood that *clear info* abbreviates $val(info) := undef$, which influences the message reception (see the function *receive* below).

add fake info to traffic ≡
  `do forall` X, Y `in` $AGENT$ `and` msg `in` $FakeMsg$
    send $X.Y.msg$
  `end-do`

save learned nonces ≡
  $Nonce(SPY) := Nonce(SPY) \cup (analz(cont(TRAFFIC),\ KEYSPY) \cap NONCE)$

destroy info from traffic ≡
  `choose` $MARK \subset TRAFFIC$
    `do forall` *info* `in` $MARK$
      clear info
    `end-do`
  `end-choose`

Fig. 2. Spy's illegal activity

We formalise a spy able to send on the network all the fake messages he can build up, in order to analyse the protocols in the worst possible scenario (other models make the same choice — e.g. [12, 5]). A less powerful and probably more realistic spy could be easily achieved by replacing the `do forall` by the `choose` constructor in the *add fake info to traffic* macro of Fig.2.

4 A Case Study

The Needham-Schroeder protocol with public keys is chosen to test our environment model, for this protocol has been analysed by other formalisms, and therefore offers an ideal benchmark. We discover that known results can be proven with a small mathematical burden. Moreover, we prove a crucial guarantee from the viewpoint of honest agents, addressing a non-trivial protocol goal that has lacked formal treatment thus far. A protocol run between honest agents keeps both nonces safe from a spy able to monitor the entire traffic, to replay old nonces, to build his owns, and to exploit the private keys of a set of compromised agents.

The Needham-Schroeder protocol with public keys (Fig.3) aims to establish mutual authentication between an *initiator* A and a *responder* B by way of three simple messages. Any agents can encrypt a message m using A's public key obtaining $\{m\}_{K_a}$, but only A can retrieve m from it, as A only should know her private key K_a^{-1}. N_a and N_b denote the fresh nonces generated by A and B respectively.

A is the initiator who seeks to establish a session with the responder B. A selects a fresh nonce N_a, and sends it along with her identifier to B, encrypting

the message under B's public key (message 1). At reception of message 1, B decrypts it and returns N_a with a new nonce N_b to A, encrypting the message under A's public key (message 2). A then returns the nonce N_b to B, encrypted under B's public key. The goals of this protocol are *authentication* goals, namely A should be able to infer B's presence from reception of message 2, and B should be able to infer A's presence from reception of message 3. A formal analysis can answer whether or not these goals are met.

1. $A \rightarrow B : \{N_a, A\}_{K_b}$
2. $B \rightarrow A : \{N_a, N_b\}_{K_a}$
3. $A \rightarrow B : \{N_b\}_{K_b}$

Fig. 3. Public-key Needham-Schroeder protocol

4.1 Specifying Needham-Schroeder

All agents run the same module AGENT in Fig.4.

The rule R1 models an agent initiating a protocol session (see message 1, Fig.3). The (monitored) predicate $wishToInitiate(A, B)$, $A, B \in AGENT$, holds when and only when A wishes to initiate a session with B. Given $N \in NONCE$ and $K_b = pubkey(B)$, state the obvious constraint

$$A.B.\{N, A\}_{K_b} \in TRAFFIC \Longrightarrow \neg\, (wishToInitiate\,(A, B))$$

in any state, which prevents R1 to fire after A has sent message 1.

To formalise an agent saving a freshly generated nonce, define the macro

$$add\ NewNonce\ in\ Nonce\,(Self) \equiv Nonce\,(Self) := Nonce\,(Self) \cup \{NewNonce\}$$

The rule R2 formalises the reply of an agent (see message 2, Fig.3) at reception of a message that the agent can decrypt, and that contains a nonce and an agent name. The (monitored) predicate $receive\,(B,\ A.B.msg)$, $A, B \in AGENT$, $A.B.msg \in TRAFFIC$, holds if and only if the agent B receives a message addressed to him. Recall that $TRAFFIC$ might contain more than one message addressed to the same agent, and more than one copy of the same message. Given an agent and an item of the traffic, it is not understood when the function holds on them, because of the delay that the network traffic might accumulate. Similarly, it is not understood if the function will ever hold on them, because the spy could brutely delete msg from the traffic.

In general, sender and receiver mentioned by a traffic item are not reliable, as they could have been faked by the spy. As a matter of fact, at reception of $A.B.msg$, B typically tries to infer the sender's identity from the contents of msg instead of relying on (the label) A. For this reason, we often use the notation $receive\,(B,\ \_.B.msg)$. When (the occurrence of) a message is received, it is deleted from the traffic by means of the macro *clear*.

The function $decrypt\,(msg,k)$, $msg \in MESSAGE$, $k \in PRIKEY$ formalises the decryption of a crypted message msg by a private key k. It yields *undef* if msg can not be decrypted by k.

The rule R3 models an agent's reply (see message 3, Fig.3) upon receiving a message that contains two nonces, and is decryptable by the agent's private key.

Note that the agent only replies in case he/she has built the first nonce. The sender of the message is ignored, and the reply is sent to the intended recipient of the first nonce.

The rule R4 simply models an agent's reception of a message of the form of the last line of the protocol. This message should authenticate the initiator to the responder.

AGENT

R1)
if *wishToInitiate(Self, X)*
then
 let *NewNonce* = *nonce*(*Self, X*)
  do in-parallel
   recipient(*NewNonce*):= *X*
   add NewNonce in Nonce(*Self*)
   send Self.X.{*NewNonce, Self*}$_{K_x}$
  end-par
 end-let
end-if

R2)
if *receive*(*Self, _.Self.msg*) ∧
  decrypt(*msg*, K^{-1}_{self}) = {*Nonce0, X*}
then
 let *NewNonce* = *nonce*(*Self, X*)
  do in-parallel
   recipient(*NewNonce*):= *X*
   add NewNonce in Nonce(*Self*)
   send Self.X.{*Nonce0, NewNonce*}$_{K_x}$
   clear _.Self.msg
  end-par
 end-let
end-if

R3)
if *receive*(*Self, _.Self.msg*) ∧
  decrypt(*msg*, K^{-1}_{self}) = {*Nonce1, Nonce2*}
then
 if *Nonce1* ∈ *Nonce*(*Self*)
 then
  let *X* = *recipient*(*Nonce1*)
   send Self.X.{*Nonce2*}$_{Kx}$
  end-let
  clear _.Self.msg
 end-if
end-if

R4)
if *receive*(*Self, _.Self.msg*) ∧
  decrypt(*msg*, K^{-1}_{self}) = {*Nonce0*}
then
  if *Nonce0* ∈ *Nonce*(*Self*)
  then
   clear _.Self.msg
  end-if
end-if

Fig. 4. Modelling the Needham-Schroeder protocol

The module SPY_ILLEGAL (Fig.1) is extended with the module AGENT (Fig.4) so that the spy can also act as an honest agent.

4.2 Verifying Needham-Schroeder

We prove that Needham-Schroeder with public keys assesses strong authentication goals in any run between uncompromised agents: their nonces are kept secret from the spy. We also show that the initiator of a conversation with the spy can be later impersonated by the spy to cheat other parties (Lowe's *middle person* attack [9]).

Definition 1. Let $N \in NONCE$. N is *compromised*, if and only if $N \in Nonce(Spy)$.

Definition 2. Let $N \in NONCE$. N is *associated by* X *with* Y, if and only if $N \in Nonce(X)$ and $recipient(N) = Y$.

Definition 3. Let $N \in NONCE$. N is *associated to a run between* X *and* Y, if and only if there exists a run initiated by X with Y, during which either N is associated by X with Y, or N is associated by Y with X.

The existence of a nonce N associated to a run between two uncompromised agents X and Y such that N is associated by Y with X, implies that the message 2 sent by X to trigger Y to issue N has in fact reached Y, namely it has not been destroyed from the traffic by the spy before reaching Y.

Theorem 1 (Authentication). *Let $A, B \notin COMPR$. If the nonces N_a and N_b are associated to a run between A and B, then N_a and N_b are not compromised.*

Proof. N_a is associated by A with B, and N_b is associated by B with A, so either one could become compromised only by execution of the rule R (in the module of *Spy*) on it, which requires N_a or $N_b \in analz(cont(TRAFFIC), KEYSPY)$. Define

- $P1 \equiv N_a \notin analz(cont(TRAFFIC),\ KEYSPY)$
- $P2 \equiv N_b \notin analz(cont(TRAFFIC),\ KEYSPY)$

We prove by induction that $P1$ holds in any state.

Let $\hat{t} \geq 0$ be the minimum index such that the nonce N_a appears on the traffic at state $S_{\hat{t}}$. If $t < \hat{t}$, $P1$ trivially holds at S_t.

Let msg be the message containing N_a such that $A.B.msg \in TRAFFIC$ holds in $S_{\hat{t}}$. By minimality of $\hat{t}$, and definition of *synth* and *analz*, it is $msg = \{N_a, A\}_{K_b}$. By the hypothesis $B \notin COMPR$ and the non-clashing constraint on the function *nonce*, $P1$ holds at $S_{\hat{t}}$.

Let $t > \hat{t}$. By inductive hypothesis $P1$ holds at S_t. Let us that prove $P1$ holds at S_{t+1}. Consider the transition $S_t \longrightarrow S_{t+1}$ by execution of all agents' rules.

- If by R1 between any pair of agents, then the thesis by the non-clashing constraint on *nonce*.
- If by R2 or R3 between any pair of agents different from A and B, then the thesis by the non-clashing constraint on *nonce*.
- If by R2 between agents A and B, it is $Nonce0 = N_a$ at S_{t+1}, then the thesis by the non-clashing constraint on *nonce* and by $A \notin COMPR$.
- If by R3 between agents A and B, then R2 fired before time t implying $Nonce0 \neq N_a$ at S_{t+1} by the non-clashing constraint on *nonce*; then the thesis.
- If by R4, then the thesis (obvious).
- If by R, then the thesis by definition of *synth* and *analz* and by $A, B \notin COMPR$.

By similar reasoning, $P2$ holds in any state. Thus, the thesis holds. □

Theorem 2 (Lowe's Attack). *Let $A, B \notin COMPR$. If A starts a run with Spy, then there exists a nonce N_b associated by B with A, such that N_b is not associated to a run between A and B, and N_b is compromised.*

Proof. It is straightforward to verify that there exists an ASM computation so that the following items are (subsequently) sent on the traffic

- by R1 in A's module: $A.Spy.\{N_a,\ A\}_{K_{spy}}$ (with $recipient\,(N_a) = Spy$)
- by R in Spy's module: $Spy.B.\{N_a,\ A\}_{K_b}$
- by R2 in B's module: $B.A.\{N_a,\ N_b\}_{K_a}$ (with $recipient\,(N_b) = A$)
- by R3 in A's module: $A.Spy.\{N_b\}_{K_{spy}}$

Therefore, by R in Spy's module, $N_b \in NONCE\,(Spy)$. □

Theorem 2 shows that our model can reproduce Lowe's attack.

4.3 Specifying and Verifying Needham-Schroeder-Lowe

In order to prevent the attack reproduced in Theorem 2, Lowe proposes to mention B explicitly in message 2, as shown in Fig.5.

1. $A \to B : \{N_a, A\}_{K_b}$
2. $B \to A : \{N_a, N_b, B\}_{K_a}$
3. $A \to B : \{N_b\}_{K_b}$

Fig. 5. Needham-Schroeder-Lowe protocol

The model given in Fig.4 is adapted to the new protocol by the updates in Fig.6.

R2')
if *receive*(*Self*, _.*Self.msg*) ∧
  decrypt(*msg*, K^{-1}_{self}) = {*Nonce0*, *X*}
then
 let *NewNonce* = *nonce*(*Self*, *X*)
  do in-parallel
   recipient(*NewNonce*):= *X*
   add NewNonce in Nonce(*Self*)
   send Self.X.$\{$*Nonce0, NewNonce, Self*$\}_{K_x}$
   clear _.*Self.msg*
  end-par
 end-let
end-if

R3')
if *receive*(*Self*, _.*Self.msg*) ∧
  decrypt(*msg*, K^{-1}_{self}) =
    = {*Nonce1*, *Nonce2*, *X*}
then
  if *Nonce1* ∈ *Nonce*(*Self*) ∧
    recipient(*Nonce1*) = *X*
  then
    send Self.X.$\{$*Nonce2*$\}_{K_x}$
    clear _.*Self.msg*
  end-if
end-if

Fig. 6. Modelling Needham-Schroeder-Lowe protocol (portion)

To show that Lowe's correction is effective, let $\mathcal{A}$ denote the ASM for the Needham-Schroeder protocol, and $\mathcal{A}'$ the ASM for the protocol refined by Lowe.

Remark 1. Let ρ be the computation in $\mathcal{A}$ sketched by the proof of Theorem 2. Does ρ exist in $\mathcal{A}'$? If $Spy.B.\{N_a,\ A\}_{K_b}$ appears on the traffic in a computation in $\mathcal{A}'$, then B may reply by R2' sending the message $B.A.\{N_a,\ N_b,\ B\}_{K_a}$ with $recipient\,(N_b) = A$. R3' can not fire because $recipient\,(N_a) \neq B$, so A would not disclose N_b to the spy.

Theorem 3 (Lowe's Correction). *The computation ρ does not exist in $\mathcal{A}'$.*

Proof. Straightforward by remark 1. □

Theorem 3 shows that Lowe's correction to the original protocol prevents effectively the *middle-person* attack.

5 Related Work

Burrows et al. [4] analyse the public-key Needham-Schroeder protocol by means of a suitable belief logic, stating that the protocol meets the expected authentication goals. They model no eavesdropper. Theorem 1 provides a stronger guarantee, showing the same results to hold in a network under the total control of an active eavesdropper.

The work of Meadows (e.g. [10]) is historically the first successful attempt to exceed the limits of state enumeration. The induction comes in her proofs that certain states are unreachable.

More recently, Paulson has developed a purely inductive approach based on theorem proving [12]. The approach deals with systems of infinite size, and considers multiple runs and accidental leaks of secrets. His model of spy is acknowledged amongst the most powerful ever proposed, and has been extended by the first author to handle timestamping [1]. Our operators *analz* and *synth* are inspired by his. He proves secrecy goals of the intuitive form

```
[| Says A B (Crypt (pubK B) {|Nonce Na, Agent A|}) : set evs;
   A ∉ bad; B ∉ bad; evs : ns_public |]
⟹ Nonce Na ∉ analz (spies evs)
```

where `bad` is the set of compromised agents. The traffic is modelled at a low level of abstraction: `ns_public` is the set of all traces of events.

The notion of *strand space*, developed by Fabrega et al. [5], is based on induction carried out by hand. A strand space is a collection of strands (sequences of events) equipped with a graph structure that allows a detailed protocol analysis. However, further case studies are necessary to benchmark the potentialities of the approach.

6 Conclusion

State enumeration methods can find simple flaws on systems of limited size. However, these methods seldom specify the protocol and its goals formally, which makes it hard to say whether the protocol is correct, even when no attacks are pointed out on the limited system. This motivated the development of an ASM model of a realistic crypto-protocol environment to analyse any crypto-protocols and the goals they meet. Despite the presence of an active eavesdropper, the definitions are short and intuitive, so that proofs can be carried out by hand.

The model has been tested on a well known protocol, Needham-Schroeder with public keys, and a guarantee useful to honest agents has been proven (Theorem 1). We believe that this guarantee has lacked a formal treatment thus far.

Lowe's important considerations on the protocol can be faithfully and clearly reproduced in the model.

Future work includes the analysis of crypto-protocols based on shared keys. The main feature — the existence of shared keys, instead of private and public keys — has been allowed already, so we expect the model to be adapted easily.

We also plan to use an existing theorem prover to exploit and mechanise our mathematical proofs.

Acknowledgments. We are grateful to Larry Paulson for the invaluable suggestions.

References

1. Bella, G., Paulson, L. C.: Mechanising BAN Kerberos by the Inductive Method. Proc. of Conference on Computer Aided Verification. Springer (1998) (to appear)
2. Bella, G., Riccobene, E.: Formal Analysis of the Kerberos Authentication System. Journal of Universal Computer Science: Special Issue on Gurevich's Abstract State Machine **3(12)** (1997) 1337–1381
3. Brackin, S. H.: A HOL Extension of GNY for Automatically Analyzing Cryptographic Protocols. Proc. of Computer Security Foundations Workshop. IEEE Press (1996)
4. Burrows, M., Abadi, M., Needham, R. M.: A logic of authentication. Proceedings of the Royal Society of London **426** (1989) 233–271
5. Fábrega, F. J. T., Herzog, J. C., Guttman, J. D.: Strand Spaces: Why is a Security Protocol Correct?. Proc. of Symposium on Security and Privacy. IEEE Press (1998)
6. Kemmerer, R., Meadows, C., Millen, J.: Three Systems for Cryptographic Protocol Analysis. Journal of Cryptology **7(2)** (1994) 79–130
7. Gurevich, Y.: Evolving Algebras 1993: Lipari Guide. In E. Börger (Ed.): Specification and Validation Methods. Oxford University Press (1995) 9–36
8. Gurevich, Y.: May 1997 Draft of the ASM Guide. Technical Report CSE-TR-336-97 (1997). University of Michigan, EECS Department
9. Lowe, G.: Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR. In Margaria and Steffen (Eds.): Tools and Algorithms for the Construction and Analysis of Systems. LNCS **1055** (1996) 147–166
10. Meadows, C.: The NRL Protocol Analyzer: An Overview. Journal of Logic Programming **26(2)** (1996) 113–131
11. Needham, R. M., Schroeder, M.: Using encryption for authentication in large networks of computers. Communications of the ACM **21(12)** (1978) 993–999
12. Paulson, L. C.: Proving properties of security protocols by induction. Proc. of Computer Security Foundations Workshop. IEEE Press (1997)
13. Paulson, L. C.: Mechanized proofs of security protocols: Needham-Schroeder with public keys. TR 432 (Jan 1997) Cambridge University Computer Laboratory
14. Schneider, S.: Verifying Authentication Protocols Using CSP. Proc. of 10th IEEE Computer Security Foundations Workshop. IEEE Press (1997)