Attacks are Protocols Too*

Anders Moen Hagalisletto
Department of Informatics, University of Oslo

Abstract

In this paper we show how to simulate attacks on authentication protocols in a realistic environment. The attack on the Needham-Schroeder public key protocol found by Gavin Lowe is run inside a malicious agent communicating with normal agents. The simulator is also able to detect flaws in attacks: a previously unreported error on an attack on the Shamir Rivest Adelman is detected by simulation.

1 Introduction

Attacks on authentication protocols constitute one of the most severe threats to the security of open distributed systems. Attacks on protocols are typically constructed in two ways: either by protocol experts manually discovering deficiencies in the protocol or by tools for the automated detection of attacks.

Both methods have limitations. The problem with manually constructed attacks is that it is a slow and potentially error prone process. The problem with the model checking approach is that the protocol analyzers are only able to analyze protocols of a limited size because of state space explosion. Both the number of transmissions and the structural complexity of the message content can cause state space explosion. The actual protocols used in e-commerce and Internet banking are large, often combining several protocols. Hence there is a problem: tools for protocol analysis cannot explore realistic commercial applications involving possibly several security protocols, yet it is increasingly important for the service providers and customers to have trust in distributed applications.

We provide a pragmatic approach to this problem by making it possible to *simulate* attacks. Thus attack simulation is a compromise between the security expert at the blackboard and the model-checker: manually written attacks can be computerized and there is no limit to how complex the protocol and attack may be.

*Thanks to Wolfgang Leister, Olaf Owe, Peter Csaba Ölveczky, Thor Kristoffersen, Boriss Mejias, and the referee for comments on earlier drafts of this paper.

In practice, an attack specification is first constructed manually, then an agent possessing the specification runs the attack protocol in an environment including other good agents. The good agents possess local copies of the intended protocol which they run. Finally queries about secrecy and authenticity can be asked about the resulting state. Attack simulation have proven to be useful in several respects:

- Several severe errors in attacks have been found recently [7], by static validation of attack specifications. Some attacks could only be detected by simulation, one such example is the attack on the Shamir Rivest Adelman.

- Large industrial applications involving nesting of protocols or sequentially composed protocols can be analyzed using simulation. These applications and protocols are too large to fully automated analysis, yet a formal framework supporting the specification of attacks can ease analysis where the security expert has some knowledge of the potential weaknesses in the protocol.

In this paper a high level language for *attack* specifications is presented. The two key notions for expressing attacks are the notions of *impersonation* and *wild-carding*. We deploy a recent result [6] stating that there is an explicit algorithm for refining attack specifications in an automated way. The resulting refined protocol contains information about the assumptions implicitly stated in the original attack specification. Assumptions are of two kinds: local *assertions*, beliefs the agent should possess in order to participate in the protocol execution, and local *actions*, construction of fresh nonces, timestamps, keys and cipher-texts. This information can be used to statically extract the beliefs possessed by the participating agent after a protocol execution. The following aspects of agents are considered:

- Agents are equipped with *beliefs* about required cryptographic entities.

- Agents communicate with the environment they inhabit. Messages are sent from the *outbuffer* and received in the *inbuffer*. The personality of the agent,

its trust relations, and locally possessed protocols determines how the message impacts the beliefs of the agent.

- In addition agents have *personality* which defines high level conditions for the way agents handle messages. Personalities are roughly divided into good and bad agents: The bad agents intercepts every message in the network and is capable of running *attack protocols*.

Cryptography is assumed to be *ideal* [11] in two respects: First, the underlying cryptographic algorithms cannot be broken. Second, encryption and decryption are performed in one single computation step. Security properties like confidentiality and authenticity of particular sentences can be re-framed in our language. This means that the simulations might be subject to standard security analysis, exemplified by the analysis of the classical attack on Needham-Schroeder protocol [8]. The language, the simulator, and formal denotational semantics have been implemented in the declarative language Maude [3]. The results we present for the agent-oriented simulation approach coincide closely with Gavin Lowe's original approach, although the security goals are formalized in temporal epistemic logic [5] and the simulation of the attack has been performed by agents running the protocol locally by protocol machines. Standard approaches, including Lowe's work, have a global view on the agents and protocols. The advantage of the local view proposed in this paper is that we can perform static validations and simulations whereby the correctness of attacks can be checked. The approach inherently guarantees that an agent does not perform actions depending on non-local variables nor knowledge, without any explicit specification of this.

The paper is organized as follows: the high level language for specifying attack descriptions is presented in Section 2. Section 3 gives an operational semantics for ordinary protocols. As described in Section 4, this semantics must be adjusted in order to represent the Dolev-Yao attacker. Finally Section 5 shows how a denotational semantics in temporal epistemic logic can be given in order to formalize the notions of confidentiality and authenticity.

2 Attack protocols

In this section we first present a language for specification of *intended* protocols, and then show how this language is extended to tackle *attack* protocols.

A language for specifying security protocols $\mathscr{L}_P$ can be defined as follows: $\mathscr{L}_P$ consists of terms of four sorts, *agents*, *nonces*, *time-stamps*, *keys*, and natural numbers. The *agent-names* are typically written "Alice", "Bob", "Server", *agent-variables* are written x, y, $x_1, x_2, \ldots$, and *agent-terms* a, b, c, $\ldots$, or in the general style t^A, t_1^A, t_2^A, $\ldots$, indicating that the terms have the *agent sort*. Variables for the new sorts are labeled with x^N, x^T, and x^K respectively, when their sorts are emphasized. Constants include *protocol names* μ, μ_1, μ_2, *encryption methods* for cryptography s (symmetric) and a (asymmetric), in addition to the indicators i (private) and u (public). There are function symbols for nonces $\mathsf{n}(N, a)$, time-stamps $\mathsf{stamp}(t^T, t^A)$, keys $\mathsf{key}(\mathsf{s}, a, b, x^K)$, $\mathsf{key}(\mathsf{a}, \mathsf{i}, a, x^K)$, $\mathsf{key}(\mathsf{a}, \mathsf{u}, a, x^K)$, and for the concatenation of protocol names.

Definition 1 *$\mathscr{L}_P$ is the smallest language such that:*

(i) Each of the following atomic formulas are in $\mathscr{L}_P$

ε	the empty sentence
$a = b$	equality
$\mathsf{Agent}(a)$	a is an agent
$\mathsf{isKey}(k)$	k is a key
$\mathsf{isNonce}(\mathsf{n}(N, a))$	$\mathsf{n}(N, a)$ is a nonce
$\mathsf{Time}(\mathsf{stamp}(N, a))$	$\mathsf{stamp}(N, a)$ timestamp
$\mathsf{playRole}(a, x, \mu)$	a plays the x-role in protocol μ
$\mathsf{role}(a)$	a is a role in a protocol

(ii) If φ, ψ, $\xi^{\mathcal{T}}$, $\xi^{\mathcal{A}}$, $\xi^{\mathcal{S}} \in \mathscr{L}_P$, then so are;

$\neg\varphi$, $\varphi \rightarrow \psi$	propositional logic
$\mathsf{Transmit}(a, b, \varphi)$	a sends the message φ to b
$\mathsf{Bel}_a(\varphi)$	a believes φ
$\varphi \,\mathcal{U}\, \psi$	φ holds until ψ holds
$\forall x\, \varphi$	first order quantification
$\forall X\, \varphi$	second order quantification
$E[k : \varphi]$	encrypt φ using key k
$D[k : \varphi]$	decrypt φ using key k
$\mathsf{Enforce}_{t^A}(\varphi)$	enforce agent t^A to do φ
$\mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \Phi]$	protocol operator

As usual $\wedge$, $\vee$ and $\leftrightarrow$ are definable using $\neg$ and $\rightarrow$ and $\top = \varepsilon \rightarrow \varepsilon$. Examples of properties that may be expressed are *honesty*, *dishonesty* and *trust*:

$$
\begin{array}{ll}
\mathrm{Honest}(a) & \Leftrightarrow \forall X\, \forall x\, (\mathsf{Transmit}(a, x, X) \rightarrow \mathsf{Bel}_a(X)) \\
\mathrm{Dishonest}(a) & \Leftrightarrow \forall X\, \forall x\, (\mathsf{Transmit}(a, x, X) \rightarrow \mathsf{Bel}_a(\neg X)) \\
\mathrm{Trust}(a, b, \varphi) & \Leftrightarrow \mathsf{Transmit}(b, a, \varphi) \rightarrow \mathsf{Bel}_a(\varphi) \\
\mathrm{Trust}(a, b) & \Leftrightarrow \forall X\ \mathrm{Trust}(a, b, X)
\end{array}
$$

The operator *before* $\mathscr{B}$ is definable from $\mathcal{U}$ by $\varphi \,\mathscr{B}\, \psi = \neg(\neg\varphi \,\mathcal{U}\, \psi)$. An event is the minimal unit in a protocol specification. The basic examples of events are transmissions or the empty event ε. A protocol is a chain of events between agents. A *chain of events* is a sentence of the form $\varphi_1 \,\mathscr{B}\, \varphi_2 \wedge \varphi_2 \,\mathscr{B}\, \varphi_3 \wedge \ldots \wedge \varphi_{n-1} \,\mathscr{B}\, \varphi_n$, where each φ_i is an event. We let $\Phi = \varphi \,\mathscr{B}\, [\Phi']$ denote a chain of events written by recursion, hence φ is a single event and Φ' chain of events. The sentence $\mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \Phi\,]$ reads "protocol named μ with session number N with the total roles $\xi^{\mathcal{T}}$, the agent specific roles $\xi^{\mathcal{A}}$, the start-roles $\xi^{\mathcal{S}}$ and the protocol body Φ". We say that a *textbook protocol* is a valid protocol where each single event is of the form $\mathsf{Transmit}(x_j, x_k, \xi)$, where x_j, x_k are agent-variables and

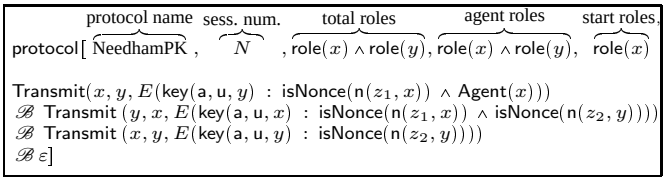

Figure 1. The Needham-Schroeder public key protocol.

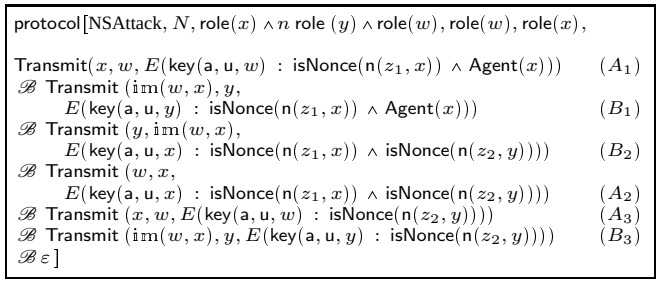

Figure 2. Attack on the Needham-Schroeder public key protocol.

ξ is a sentence in $\mathscr{L}_P$. Figure 1 shows the textbook specification of the Needham-Schroeder public key protocol. The protocol body consists of three transmission sentence, and ends with the empty event.

2.1 Specification of attacks

Consider the Needham-Schroeder authentication protocol for public keys [12]: The authentication part of the protocol is given as follows, only involving Alice (A) and Bob (B):

$$\begin{array}{lll} (P_1) & A \longrightarrow B: & E(PK_B : N_A, A) \\ (P_2) & B \longrightarrow A: & E(PK_A : N_A, N_B) \\ (P_3) & A \longrightarrow B: & E(PK_B : N_B) \end{array}$$

The goal of the protocol is to provide mutual authentication both for the initiator A and the responder B. The assumption of the protocol is that the agents possess their own private keys and the required public keys. The means to achieve this goal are to use freshly generated nonces that should be kept secret by encryption using their public keys. Gavin Lowe [8] found an attack on the protocol where two parallel sessions are interleaved, one intended session α between Alice and Intruder, and one corrupted session β where Intruder impersonates Alice:

$$\begin{array}{lll} (\alpha_1) & A \longrightarrow I & : E(PK_I : N_A, A) \\ (\beta_1) & I(A) \longrightarrow B & : E(PK_B : N_A, A) \\ (\beta_2) & B \longrightarrow I(A) & : E(PK_A : N_A, N_B) \\ (\alpha_2) & I \longrightarrow A & : E(PK_A : N_A, N_B) \\ (\alpha_3) & A \longrightarrow I & : E(PK_I : N_B) \\ (\beta_3) & I(A) \longrightarrow B & : E(PK_B : N_B) \end{array}$$

The attack on the Needham-Schroeder protocol may be formalized in $\mathscr{L}_P$ required that the notions of impersonation and wild-cards are introduced.

2.1.1 Impersonation

Impersonation is the phenomenon whereby an agent c behaves as if it were another agent a, without letting the environment become aware of it. In terms of communication, impersonation comes into play both in sending and receiving messages. Technically, we extend only the term language with one new function symbol $\mathtt{im}(c, a)$, that reads "agent c impersonates agent a". An *attack protocol* is a protocol P, that contains occurrences of agent terms $\mathtt{im}(c, a)$ either as the sender or the receiver of a transmission. In the former case $\mathsf{Transmit}(\mathtt{im}(c, a), b, \varphi)$, the real sender of the message is c, while the message itself claims that a is the originator. In the latter case $\mathsf{Transmit}(b, \mathtt{im}(c, a), \varphi)$, the message is intercepted by c, hence agent a never receives the message.

If we translate the roles of Alice, Bob, and Intruder, by the concrete variables x (for Alice), y (for Bob), w (for Intruder), we obtain the following protocol depicted in Figure 2. One problem with the specification in Figure 2, is that it gives to much explicit information about the intended execution. More specifically, the nonce $\mathsf{isNonce}(\mathsf{n}(z_1, x))$ has explicit reference to the agent y. If x possess a copy of the protocol, then y should not have any knowledge of the originator of the nonce, in this case x. The way to bypass this problem is by introducing *wild-cards*.

2.1.2 Wild-cards

In a realistic implementation of a cryptographic protocol, the internal logical structure of cipher-text and other cryptographic entities, like nonces, is unknown to the receiver. Wild-cards have been introduced as an efficient technique to represent indefinite protocol information by several authors ([10], [4]). In the context of this paper wild-cards are particular variables. If x is an arbitrary first order variable, then $x^{\%}$ denotes the *wild-carding* of x. Given a protocol specification, there are several ways to wild-card it. The wild-carding depends on the perspective of the concrete role the agent plays in a protocol session. If Alice possess the protocol and plays the responder role y in the protocol, then the wild-carding should result in a protocol where cryptographic entities wild-card the reference to other agents. For

instance the first message (A_1) is wild-carded by:

$$\mathsf{Transmit}(x, y, E(\mathsf{key(a,u,}y) : \mathsf{isNonce(n}(z_1, x^{\%})) \wedge \mathsf{Agent}(x)))$$

Hence in a simulation, the wild-carding of a protocol is performed not once and for all, but *on the fly* depending on which role the agent play in the protocol. The attack specification given in Figure 2 can be refined to a protocol specification including the local cryptographic assumptions in a fully automated way [6]. The refined specification is taken as input to the simulator, and if the specification in sound then it is executable in the simulator.

3 Operational semantics for protocols

In this section we shall first describe an operational semantics for executing standard protocols and then see how the semantics should be modified in order to include attacks. Agents communicate with other agents over a network. A message in the network consists of a message content m, the sender's name and the receivers name. If a and b are agent names, and m is the message content we write $\mathsf{msg}\ m\ \mathsf{from}\ a\ \mathsf{to}\ b$. The entities *agents* and *messages* are the only inhabitants in the model. We introduce the *parallel operator* $\parallel$, and use the notation $o_1 \parallel o_2 \parallel \ldots \parallel o_n$, to express that the entities $o_1, o_2, \ldots, o_n$ coexists concurrently. A rewrite rule $t \longrightarrow t'$ can be interpreted as a local transition rule allowing an instance of the term t to evolve into the corresponding instance of the pattern t'. Each rewrite rule describe how a part of a configuration can evolve in one transition step. A *configuration* is a snapshot of a dynamic system evolving. The parallel operator is an abelian monoid over the set of configuration with an identity element (the empty configuration). The agent is a structure containing four slots:

$$\prec \underbrace{\mathsf{id}}_{\text{agent name}} \mid \underbrace{\mathsf{bel},}_{\text{set of sentences}} \underbrace{\mathsf{in},\ \mathsf{out}}_{\text{buffers}} \succ$$

where id denotes the *identity* or *name* of the agent, while bel denotes its current *set of beliefs*. The variable denoted in represents the *inbuffer* - messages waiting for processing in protocol while out denotes the *outbuffer* - messages intended for transmission.

3.1 Asynchronous communication

In case of the honest agent there are two rules for communication, the rule for sending and receiving messages. Referring to figure 3, the send rule involves updating the agents beliefs, and puts the message $\mathsf{msg}\ F\ \mathsf{from}\ a\ \mathsf{to}\ b$ into the network. Since a is required to be honest, the transmission of the rule requires that $\mathsf{Bel}_a(F)$. Operationally this

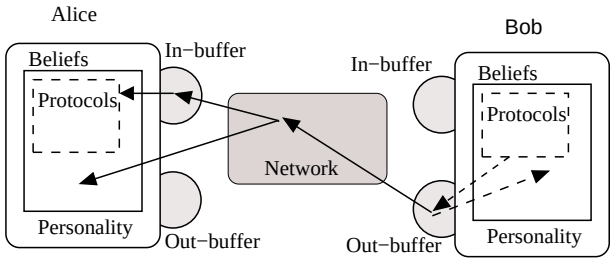

Figure 3. Uncompromized message flow.

is interpreted by:

$$\begin{array}{l} \prec a \mid \mathsf{bel, out} \cup \{\mathsf{Transmit}(a, b, F)\} \succ \\ \longrightarrow \\ \prec a \mid \mathsf{bel} \cup \{\mathsf{Transmit}(a, b, F)\}, \mathsf{out} \succ \parallel \mathsf{msg}\ F\ \mathsf{from}\ a\ \mathsf{to}\ b \\ \qquad \text{if } \mathsf{Honest}\ (a) \in \mathsf{bel} \end{array}$$

The receive rule involves transferring the message from the network and memorizing the message:

$$\begin{array}{l} \prec b \mid \mathsf{bel, in} \succ \parallel \mathsf{msg}\ F\ \mathsf{from}\ a\ \mathsf{to}\ b \longrightarrow \\ \prec b \mid \mathsf{bel} \cup \{\mathsf{Transmit}(a, b, F), \mathsf{Agent}(a)\}, \mathsf{in} \cup \{\mathsf{Transmit}(a, b, F)\} \succ \end{array}$$

3.2 Protocol execution - the protocol machine

The operational semantics contains some meta-concepts required for executing protocol specifications. We introduce a meta-predicate $\mathbb{E}(a, \mathsf{P})$ (for *execute*), modeling the concept *protocol machine*. The predicate $\mathbb{E}(a, \mathsf{P})$ reads "agent a at stage P" in the protocol μ, and is an operator for executing any protocol μ in $\mathscr{L}_P$. The execution predicate consumes one protocol event at a time, starting from the top. The rule for transmitting a message in a protocol session then reads:

$$\begin{array}{l} \prec a \mid \mathsf{bel} \cup \{\mathbb{E}(a, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \mathsf{Transmit}(a, b, F) \mathscr{B}\ [\Phi]\ ])\}, \mathsf{out} \succ \\ \longrightarrow \\ \prec a \mid \mathsf{bel} \cup \{\mathbb{E}(a, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \Phi\ ])\}, \mathsf{out} \cup \{\mathsf{Transmit}(a, b, F)\} \succ \end{array}$$

The rule for receiving messages in a protocol session reads:

$$\begin{array}{l} \prec a \mid \mathsf{bel} \cup \{\mathbb{E}(a, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \\ \quad \mathsf{Transmit}(t_1, a, F) \mathscr{B}\ [\Phi]\ ])\}, \mathsf{in} \cup \{\mathsf{Transmit}(t_2, a, F')\} \succ \\ \longrightarrow \\ \prec a \mid \mathsf{bel} \cup \{\mathbb{E}(a, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \\ \qquad \underline{\mathrm{sub}}(\mathbb{S}(F', F) \cup \{\langle t_2, t_1 \rangle\}, \Phi)\ ])\}, \mathsf{in} \succ \\ \qquad \text{if } \mathbb{M}(F', F) \wedge \mathbb{M}(t_2, t_1) \end{array}$$

The boolean function $\mathbb{M}(F', F)$ decides if F' may match F. The function $\mathbb{S}(F', F)$ performs the matching of the terms in F' with variables in F, resulting in a set of pairs of variables and terms: $\{\langle x_1, t_1 \rangle, \ldots, \langle x_n, t_n \rangle\}$. Then if the head of the protocol body is a passive transmission, and a message in the inbuffer matches the head, the protocol proceeds by substituting the result of $\mathbb{S}(F', F) \cup \mathbb{S}(t_2, t_1)$, into the rest of the protocol body Φ. The function $\underline{\mathrm{sub}}(S, \mathsf{P})$ recursively substitute a set of matching pairs S in the protocol denoted P.

3.3 Session administration

For the purpose of this paper we omit multi-threading and the mechanisms for listening at sockets. The execute operator is accompanied by an await operator $\mathbb{A}(b, \mathsf{P})$, which reads "agent b awaits in the responder role for an incoming message in order to proceed executing the protocol P". The function $\mathfrak{F}(a, \mathsf{P})$ filters out agent a's relevant sub-protocol: that is, the function keeps transmissions and assumptions where a plays a principal role, and throws the rest of the protocol.

3.3.1 From listening to execution

When the agent receives a message from its environment, it matches the message with one of its running threads of protocol "sockets".

$$
\begin{array}{ll}
\prec b \mid \mathsf{bel} \cup \{\mathsf{protocol}[\mu, M, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \Phi\,]\} \cup & (1)\\
\{\mathbb{A}(b, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}_{\mathcal{I}}, \mathsf{role}(x), \xi^{\mathcal{S}}, \mathsf{Transmit}(t, b, F)\,\mathscr{B}\,[\psi]\,])\}, & (2)\\
\qquad\qquad \mathsf{in} \cup \{\mathsf{Transmit}(a, b, F')\} \succ & (3)\\
\quad \longrightarrow & \\
\prec b \mid \mathsf{bel} \cup \{\mathsf{protocol}[\mu, M+1, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \Phi\,]\} \cup & (4)\\
\{\mathbb{E}(b, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}_{\mathcal{I}}, \mathsf{role}(x), \xi^{\mathcal{S}}, \underline{\mathrm{sub}}(\mathbb{S}(F', F) \cup \{\langle a, t\rangle\}, \psi)\,])\} & (5)\\
\cup \{\mathbb{A}(b, \mathfrak{F}(b, \mathrm{sub}(b, x, \mathsf{protocol}[\mu, M+1, \xi^{\mathcal{T}}_{\mathcal{I}}, \mathsf{role}(x), \xi^{\mathcal{S}}, \Phi\,])))\}, \mathsf{in} \succ & (6)\\
\qquad \text{if } \mathbb{M}(F', F) \wedge \mathbb{M}(a, t) & (7)
\end{array}
$$

The rule combines three operations at once, *message handling, start of session*, and invocation of *new session thread*: The message in the inbuffer (3) is matched with the head of the protocol body (2) and (7), and removed from the inbuffer (6). An active listening (2) is replaced by a new active protocol session (6). A new thread of listening is generated (6), with the current session number, generated from the initial protocol (2).

3.3.2 Done with protocol

Every protocol ends with the empty event ε, which terminates the session:

$$
\begin{array}{c}
\prec a \mid \mathsf{bel} \cup \{\mathbb{E}(a, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \varepsilon])\} \succ \\
\longrightarrow \quad \prec a \mid \mathsf{bel} \cup \{\mathrm{Done}(a, \mu, N)\} \succ
\end{array}
$$

$\mathrm{Done}(a, \mu, N)$ is a signal for the successful termination of the protocol session.

3.4 Assumptions and cryptography

Assertions about a belief F that an agent a possess is expressed by $\mathsf{Bel}_a(F)$. Hence the assertion rule is given by:

$$
\begin{array}{l}
\prec a \mid \mathsf{bel} \cup \{\mathbb{E}(a, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \mathsf{Bel}_a(F)\,\mathscr{B}\,\varphi\,])\} \succ \\
\longrightarrow \\
\prec a \mid \mathsf{bel} \cup \{\mathbb{E}(a, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \varphi\,])\} \succ \quad \text{if } F \in \mathsf{bel}
\end{array}
$$

3.4.1 Cryptography

An agent may encrypt a sentence F using the key k only if it possess both F and k:

$$
\begin{array}{l}
\prec b \mid \mathsf{bel} \cup \{\mathbb{E}(b, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \\
\qquad\qquad \mathsf{Enforce}_b(\mathsf{Bel}_b(E[k\ :\ F]))\,\mathscr{B}\,\varphi\,])\} \succ \\
\longrightarrow \\
\prec b \mid \mathsf{bel} \cup \{E[k\ :\ F]\} \cup \{\mathbb{E}(b, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \varphi\,])\} \succ \\
\qquad \text{if } \mathsf{isKey}\ (k) \in \mathsf{bel} \text{ and } F \in \mathsf{bel}
\end{array}
$$

Decryption is given by the following rule:

$$
\begin{array}{l}
\prec b \mid \mathsf{bel} \cup \{\mathbb{E}(b, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \mathsf{Enforce}_b(\mathsf{Bel}_b(D[k\ :\ F]))\,\mathscr{B}\,\varphi\,])\} \succ \\
\longrightarrow \\
\prec b \mid \mathsf{bel} \cup \{D[k : F]\} \cup \{\mathbb{E}(b, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \varphi\,])\} \succ \\
\text{if } F \in \mathsf{bel}
\end{array}
$$

Decryption of cipher-text requires that the agent a possess the appropriate keys as follows:

key possessed	permitted cryptographic action
$\mathsf{key}(\mathsf{s}, a, b)$	$D[\mathsf{key}(\mathsf{s}, a, b) : E[\mathsf{key}(\mathsf{s}, a, b) : F]] = F$
$\mathsf{key}(\mathsf{a}, \mathsf{i}, a)$	$D[\mathsf{key}(\mathsf{a}, \mathsf{i}, a) : E[\mathsf{key}(\mathsf{a}, \mathsf{u}, a) : F]] = F$
$\mathsf{key}(\mathsf{a}, \mathsf{u}, a)$	$D[\mathsf{key}(\mathsf{a}, \mathsf{u}, a) : E[\mathsf{key}(\mathsf{a}, \mathsf{i}, a) : F]] = F$

This gives a distinction between the agent's *intention* of performing a decryption of a cipher-text, and the agent's actual *capability* to decrypt.

3.4.2 Fresh generation of keys, nonces and timestamps

Freshly generated data requires that there is a *generator*, that memorize past values of nonces, timestamps and keys. The rule for constructing fresh symmetric keys reads:

$$
\begin{array}{l}
\prec b \mid \mathsf{bel} \cup \{\mathsf{keyGen}(\mathsf{key}(\mathsf{s}, a, b, M))\} \cup \\
\qquad \{\mathbb{E}(b, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \\
\qquad\qquad \mathsf{Enforce}_b(\mathsf{Bel}_b(\mathsf{newKey}(\mathsf{key}(\mathsf{s}, a, b, x))))\,\mathscr{B}\,\varphi\,])\} \succ \\
\longrightarrow \\
\prec b \mid \mathsf{bel} \cup \{\mathsf{keyGen}(\mathsf{key}(\mathsf{s}, a, b, M+1))\} \cup \\
\qquad \{\mathsf{isKey}(\mathsf{key}(\mathsf{s}, a, b, M{+}1))\} \cup \{\mathbb{E}(b, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \varphi\,])\} \succ
\end{array}
$$

The rules for constructing fresh nonces and timestamps are similar, each requires that there is a suitable generator.

4 Simulation with the Dolev-Yao attacker

In order to run attack-protocols, the operational semantics should reflect attack specifications, both interception and impersonation should be possible. Running attack-specifications requires that the message flow is modified.

4.1 Rules for executing attack protocols

Malicious agents can do whatever good agents can and in addition fake and intercept messages. This means that the protocol machine must be extended with the corresponding rules for malicious protocol behaviour. Suppose that μ^A is an attack protocol, and i denote an agent (to be thought of

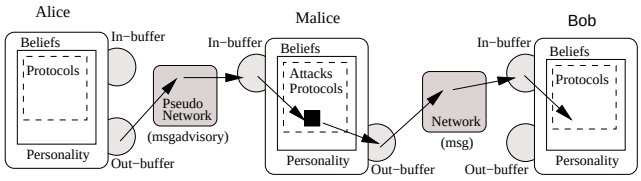

Figure 4. Compromised communication.

as the intruder). Then impersonating by sender is given by the rule:

$$\begin{array}{l}\prec i \mid \mathsf{bel} \cup \{\mathbb{E}(i, \mathsf{protocol}[\mu^{\mathcal{A}}, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \\ \qquad\qquad \mathsf{Transmit}(\mathbb{im}(i,a), b, \varphi)\, \mathscr{B}\, [\Phi]\,])\}, \mathsf{out} \succ \\ \longrightarrow \\ \prec i \mid \mathsf{bel} \cup \{\mathbb{E}(i, \mathsf{protocol}[\mu^{\mathcal{A}}, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \Phi\,])\}, \\ \qquad\qquad \mathsf{out} \cup \{\mathsf{Transmit}(\mathbb{im}(i,a), b, \varphi)\} \succ\end{array}$$

Interception of messages reads:

$$\begin{array}{l}\prec i \mid \mathsf{bel} \cup \{\mathbb{E}(i, \mathsf{protocol}[\mu^{\mathcal{A}}, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \\ \mathsf{Transmit}(t_1, \mathbb{im}(i,a), \varphi)\, \mathscr{B}\, [\Phi]\,])\}, \mathsf{in} \cup \{\mathsf{Transmit}(t_2, a, \varphi')\} \succ \\ \longrightarrow \\ \prec i \mid \mathsf{bel} \cup \{\mathbb{E}(i, \mathsf{protocol}[\mu^{\mathcal{A}}, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \\ \underline{\mathrm{sub}}(\mathbb{S}(\varphi', \varphi) \cup \{\langle t_2, t_1 \rangle\}, \Phi)\,])\}, \mathsf{in} \succ \\ \qquad\qquad \text{if } \mathbb{M}(\varphi', \varphi) \wedge \mathbb{M}(t_2, t_1)\end{array}$$

4.2 Message flow in the Dolev-Yao model

A standard approach in protocol analysis, is to let the attacker *control* the entire network. In an agent-centric approach like the one advocated in this paper the intruder is placed as malicious router that inspects and manipulates every message put into the network. Hence the network is split into two, first the *advisory* network and then the standard normal network. In Figure 4, the revised message flow according to the Dolev-Yao interpretation is depicted.

Agents sending messages acting as good agents: The good agents transmits messages, by putting the message into the channel **msgadvisory**, in which a message is denoted $\underline{\mathsf{msg}}$. Note that Malice is not permitted to use the rule, since Malice is intercepting every message, and then would intercept herself. Hence the communication rule for good agents, presented previously, is replaced by the following rule:

$$\begin{array}{l}\prec a \mid \mathsf{bel}, \mathsf{out} \cup \{\mathsf{Transmit}(a, b, \varphi)\} \succ \\ \longrightarrow \\ \prec a \mid \mathsf{bel} \cup \{\mathsf{Transmit}(a, b, \varphi)\}, \mathsf{out} \succ \parallel \underline{\mathsf{msg}}\ \varphi\ \ \mathsf{from}\ \ a\ \ \mathsf{to}\ \ b \\ \qquad\qquad \text{if } \mathsf{Honest}\ (a) \text{ and } \varphi \in \mathsf{bel} \text{ and } \neg\ \mathsf{Malicious}\ (a)\end{array}$$

Malice receives the message: A message in the **msgadvisory** is received by Malice. Malice's role in the interception is captured by the impersonation construct, $\mathsf{Transmit}(a, \mathbb{im}(i,b), \varphi)$.

$$\begin{array}{l}\prec i \mid \mathsf{bel}, \mathsf{in} \succ \parallel \underline{\mathsf{msg}}\ \varphi\ \ \mathsf{from}\ \ a\ \ \mathsf{to}\ \ b \ \longrightarrow \\ \prec i \mid \mathsf{bel} \cup \{\mathsf{Transmit}(a, \mathbb{im}(i,b), \varphi)\}, \\ \qquad\qquad \mathsf{in} \cup \{\mathsf{Transmit}(a, \mathbb{im}(i,b), \varphi)\} \succ \ \text{if } \mathsf{Malicious}\ (i)\end{array}$$

Malice sending messages as normal participant: In order to avoid that Malice intercepts his own messages, the honest transmissions by Malice is placed directly into the network:

$$\begin{array}{l}\prec i \mid \mathsf{bel}, \mathsf{out} \cup \{\mathsf{Transmit}(i, b, \varphi)\} \succ \ \longrightarrow \\ \prec a \mid \mathsf{bel} \cup \{\mathsf{Transmit}(i, b, \varphi)\}, \mathsf{out} \succ \parallel \mathsf{msg}\ \ \varphi\ \ \mathsf{from}\ \ i\ \ \mathsf{to}\ \ b \\ \qquad\qquad \text{if } \mathsf{Malicious}\ (i)\end{array}$$

Malice sending messages by impersonation: The rule is straightforward, if Malice intends to send a message by impersonation, then this fact is stored in the belief-set of Malice and the message is put on the network as message transmitted from the agent a.

$$\begin{array}{l}\prec i \mid \mathsf{bel}, \mathsf{out} \cup \{\mathsf{Transmit}(\mathbb{im}(i,a), b, \varphi)\} \succ \ \longrightarrow \\ \prec a \mid \mathsf{bel} \cup \{\mathsf{Transmit}(i, b, \varphi)\}, \mathsf{out} \succ \parallel \mathsf{msg}\ \ \varphi\ \ \mathsf{from}\ \ a\ \ \mathsf{to}\ \ b \\ \qquad\qquad \text{if } \mathsf{Malicious}\ (i)\end{array}$$

Any agent (good or bad) receiving messages: The rule for receiving messages is the same for Malice as it is for any other good agent. Hence to conclude: Executing the attack protocols require few changes into the protocol machine. The communication model involve the introduction of one extra state that the messages must pass through, the Dolev-Yao attacker.

4.3 Executing the protocol and the attack

Simulations of the Needham-Schroeder Public Key protocol were performed successfully in PROSA, with an initial configuration containing only good agents. The specification of Lowe's attack on the Needham-Schroeder was executed in the simulator using 6179 rewrite steps. In the next session we shall analyze the result of the simulation based on the final configuration in the protocol session.

5 Formalizing protocol goals

In specifications languages like CAPSL [4] and Casper [10], and HLPSL [1], the protocol specifier is invited to express security goals of the protocols. The main security properties that play a role in authentication protocols are *secrecy* and *authenticity*. Since our language is based on epistemic logic, the natural semantics to reformulate is Kripke models.

5.1 Denotational semantics

The operational semantics can be embedded in a denotational semantics, suitable for expressing security properties about a given configuration. The *denotational semantics* for the language can be given by connecting the transition relation from the operational semantics. First we interpret the single transition arrow; $\mathscr{C} \longrightarrow \mathscr{C}'$ iff $\mathscr{C} \longrightarrow^1 \mathscr{C}'$. The arrow $\xrightarrow{+}$ is the transitive closure of the one step arrow

$\longrightarrow^1$, that is $\mathscr{C} \xrightarrow{+} \mathscr{C}'$ iff $\mathscr{C} \longrightarrow^1 \mathscr{C}' \vee \exists \mathscr{C}''(\mathscr{C} \longrightarrow^1 \mathscr{C}'' \wedge \mathscr{C}'' \xrightarrow{+} \mathscr{C}')$. A *contextual model* is a pair $\langle \mathcal{C}, \longrightarrow^* \rangle$, where $\mathcal{C}$ is a set of configurations, and $\xrightarrow{+} \subseteq \mathcal{C} \times \mathcal{C}$.

Definition 2 *The truth of a formula $\phi \in \mathscr{L}_S$ in a configuration in a contextual model $\mathscr{M} = \langle \mathcal{C}, \xrightarrow{+} \rangle$, denoted $\mathscr{M} \models_{\mathscr{C}} \phi$, is defined by:*

$$
\begin{array}{lll}
\mathscr{M} \models_{\mathscr{C}} a = b & \textit{iff} & a^{\mathscr{M}} =^{\mathscr{M}} b^{\mathscr{M}} \\
\mathscr{M} \models_{\mathscr{C}} \mathsf{Agent}(a) & \textit{iff} & \prec a \,|\, \mathsf{bel} \succ \in \mathscr{C} \\
\mathscr{M} \models_{\mathscr{C}} \mathsf{Transmit}(a,b,F) & \textit{iff} & (\mathsf{msg}\ F\ \mathsf{from}\ a\ \mathsf{to}\ b) \in \mathscr{C} \\
\mathscr{M} \models_{\mathscr{C}} \mathsf{Bel}_a(\varphi) & \textit{iff} & \prec a \,|\, \mathsf{bel} \cup \{\varphi\} \succ \in \mathscr{C} \\
\mathscr{M} \models_{\mathscr{C}} \neg\varphi & \textit{iff} & \mathscr{M} \not\models_{\mathscr{C}} \varphi \\
\mathscr{M} \models_{\mathscr{C}} \varphi_1 \rightarrow \varphi_2 & \textit{iff} & \mathscr{M} \models_{\mathscr{C}} \varphi_1 \implies \mathscr{M} \models_{\mathscr{C}} \varphi_2 \\
\mathscr{M} \models_{\mathscr{C}} \forall x \in \mathsf{Ag}\, \phi(x) & \textit{iff} & \\
\multicolumn{3}{r}{\forall a\ (\prec a \,|\, \mathsf{bel} \succ \in \mathscr{C} \implies \mathscr{M} \models_{\mathscr{C}} \phi(a))} \\
\mathscr{M} \models_{\mathscr{C}} \varphi_1 \mathcal{U} \varphi_2 & \textit{iff} & \exists \mathscr{C}'(\mathscr{C} \xrightarrow{+} \mathscr{C}' \wedge \mathscr{M} \models_{\mathscr{C}'} \psi \\
\multicolumn{3}{r}{\wedge \forall \mathscr{C}''(\mathscr{C} \xrightarrow{+} \mathscr{C}'' \wedge \mathscr{C}'' \xrightarrow{+} \mathscr{C}' \implies \mathscr{M} \models_{\mathscr{C}''} \varphi))} \\
\mathscr{M} \models_{\mathscr{C}} \forall X\, \varphi(X) & \textit{iff} & \forall \psi \in \mathscr{L}_S\ \mathscr{M} \models_{\mathscr{C}} \varphi(\psi)
\end{array}
$$

A sentence φ is *valid* in a model $\mathscr{M} = \langle \mathcal{C}, \xrightarrow{+} \rangle$, denoted $\mathscr{M} \models \varphi$, iff for every $\mathscr{C} \in \mathcal{C}$, $\mathscr{M} \models_{\mathscr{C}} \varphi$. The Kripke semantics gives the toolbox to express and reason about the security goals of secrecy and authenticity with respect to the execution of a particular protocol session or a certain state in the session.

5.2 Secrecy

One of the goals of an authentication protocol is to keep certain data *secret*. A fact φ is *contingent secret* for a group of agents G, written $\mathrm{CSecret}(G, \varphi)$, iff every agent not in the group does not possess φ, formally:

$$\mathrm{CSecret}(G, \varphi) \xleftrightarrow{\text{def}} \forall x(\mathsf{Agent}(x) \wedge x \notin G \rightarrow \neg \mathsf{Bel}_x(\varphi)).$$

In case of the Needham-Schroeder protocol, the two nonces is intended to be secret after the end of the session:

$$\mathscr{M} \models_{\mathscr{C}_e} \mathrm{CSecret}(\{\mathsf{Alice}, \mathsf{Bob}\}, \\ \mathsf{isNonce}(\mathsf{n}(N_1, \mathsf{Alice})) \wedge \mathsf{isNonce}(\mathsf{n}(N_2, \mathsf{Bob})))$$

for given nonces $\mathsf{n}(N_1, \mathrm{Alice})$ and $\mathsf{n}(N_2, \mathrm{Bob})$.

Yet contingent secrecy does not reveal much of the true nature of confidentiality, a sentence φ might be contingent secret by chance. A stronger notion is obtained by requiring that every configuration remain contingent secret in the future: We define the operator *always* in terms of the until operator by $\boxed{\mathsf{F}}\ \varphi = \neg(\top \mathcal{U} \neg\varphi)$, and derive the *now and always in the future* operator by $\boxdot_\varphi = \varphi \wedge \boxed{\mathsf{F}}\ \varphi$.

$$\mathrm{Secret}(G, \varphi) \xleftrightarrow{\text{def}} \boxdot \forall x(\mathsf{Agent}(x) \wedge x \notin G \rightarrow \neg \mathsf{Bel}_x(\varphi)).$$

Both local and global notions are expressible within the same language. The conceptual gap between local and global concepts also holds for the two concepts of secrecy: The sentence $\mathrm{CSecret}(G, \varphi)$ is verified by examining the entire configuration and collecting the beliefs of every agent. Suppose that $G = \{a_1, \ldots, a_n\}$ is a given group of agents. By Definition 2, the meaning of $\mathrm{CSecret}(G, \varphi)$ is given by:

$$\mathscr{M} \models_{\mathscr{C}} \mathrm{CSecret}(G, \varphi) \text{ iff } \forall a\ (\prec a \,|\, \mathsf{bel} \succ \in \mathscr{C} \text{ and } \\ \mathscr{M} \models_{\mathscr{C}} a \neq a_1 \wedge \ldots \wedge a \neq a_n \implies \varphi \notin \mathsf{bel})$$

The sentence $\mathsf{Bel}_a(\mathrm{CSecret}(G, \varphi))$ expresses that the agent a believes that φ is secret for the group G, hence

$$\mathscr{M} \models_{\mathscr{C}} \mathsf{Bel}_a(\mathrm{CSecret}(G, \varphi)) \text{ iff } \\ \prec a \,|\, \mathsf{bel} \cup \{\forall x(\mathsf{Agent}(x) \wedge x \notin G \rightarrow \neg\, \mathsf{Bel}_x(\varphi))\} \succ \in \mathscr{C}$$

The latter distinction indicates a distinction between agents beliefs that corresponds and do not corresponds to the environment.

5.3 Correspondence

Agents might have false beliefs, this is particularly the case in computer security. The concept *correspondence*, captures the connection between the local perspective of an agent with the objective global facts of the configuration. Notions of correspondence can be distinguished with respect to their strength If every belief that an agent a posses is correct in a configuration $\mathscr{C}$, then we say that the agent *corresponds* to the configuration, denoted $\mathrm{Corr}(a, \mathscr{C})$. Correspondence in security is rare. In the cases of attacks on protocols involving good agents, the good agents possess non-corresponding sentences, as well as corresponding sentences. We call the latter notion *weak correspondence*, denoted $\mathrm{WCorr}(a, \mathscr{C}, \varphi)$. If an agent always possess the correct beliefs about its environment, then we say that the agent has *strong correspondence* with its environment, denoted $\mathrm{SCorr}(a, \mathscr{C})$. The formal definitions of the notions of correspondence is given below:

$$
\begin{array}{ll}
\mathrm{WCorr}(a, \mathscr{C}, \varphi): & \mathscr{M} \models_{\mathscr{C}} \mathsf{Bel}_a(\varphi) \text{ and } \mathscr{M} \models_{\mathscr{C}} \varphi \\
\mathrm{Corr}(a, \mathscr{C}): & \text{For every } \varphi \in \mathscr{L}_S, \\
 & \quad \text{if } \mathscr{M} \models_{\mathscr{C}} \mathsf{Bel}_a(\varphi), \text{ then } \mathscr{M} \models_{\mathscr{C}} \varphi \\
\mathrm{SCorr}(a, \mathscr{C}): & \boxdot \mathrm{Corr}(a, \mathscr{C})
\end{array}
$$

Weak correspondence expresses that one particular belief of the agent a in a particular configuration $\mathscr{C}$ corresponds to reality (the global view of the model). Correspondence states that at a given point in execution, in the configuration $\mathscr{C}$, agent a has a correct view on its environment. Strong correspondence states that from an initial configuration $\mathscr{C}$, a always corresponds, hence no faulty beliefs will ever appear in a's mind.

(A_1) $\mathsf{Bel}_{\mathrm{Alice}}(\mathsf{Done}(\mathrm{NSPK}, 1, \mathsf{role}(\mathrm{Alice}) \wedge \mathsf{role}(\mathrm{Malice})))$
(A_2) $\mathsf{Bel}_{\mathrm{Bob}}(\mathsf{Done}(\mathrm{NSPK}, 1, \mathsf{role}(\mathrm{Alice}) \wedge \mathsf{role}(\mathrm{Bob})))$
(A_3) $\mathsf{Bel}_{\mathrm{Malice}}(\mathsf{Done}(\mathrm{NSPK}_{\mathrm{ATTACK}}, 1, \mathsf{role}(\mathrm{Alice}) \wedge \mathsf{role}(\mathrm{Bob}) \wedge \mathsf{role}(\mathrm{Malice})))$
(A_4) $\neg\, \mathsf{Bel}_{\mathrm{Alice}}(\mathsf{Done}(\mathrm{NSPK}, \mathsf{role}(\mathrm{Alice}) \wedge \mathsf{role}(\mathrm{Bob})))$

Figure 5. Protocol sessions finished.

5.4 Authenticity

The agent's view on their final configuration denoted $\mathscr{C}_{\mathrm{e}}$ is described in Figure 5. Each of the agents are done with a protocol sessions: The clause A_1 means that Alice believes that she has successfully run a NSPK session with Malice without being aware that Malice is an intruder. From what Alice knows Malice is authenticated to her, although the nonce from Malice is actually compromised. The clause A_2 states that Bob believes that he has be running a successful session of NSPK with Alice. From the external point of view both A_1 and A_2 are faulty. The final statement A_3 expresses that Malice has successfully run an attack on the Needham-Schroeder Public Key protocol, with the good agents Alice and Bob.

Lowe's attack is an attack on authenticity. The correspondence property might be lifted to include the finalization predicate:

$$\text{If } \mathscr{M} \models_{\mathscr{C}} \mathsf{Bel}_a(\mathsf{Done}(\mu, N, \xi)), \text{ then } \mathscr{M} \models_{\mathscr{C}} \mathsf{Done}(\mu, N, \xi)$$

where ξ denote a conjunction of roles. The session number might be suppressed, by writing $\mathsf{Done}(\mu, \bigwedge_{i=1}^{n} \mathsf{role}(a_i))$ as an abbreviation for $\exists N\ \mathsf{Done}\ (\mu, N, \bigwedge_{i=1}^{n} \mathsf{role}(a_i))$. By avoiding the synchronization problem concerning session numbers, we can define *weak finalization* by the following:

$$\mathscr{M} \models_{\mathscr{C}} \mathsf{Done}(\mu, \bigwedge_{i=1}^{n} \mathsf{role}(a_i)) \text{ iff } \mathscr{M} \models_{\mathscr{C}} \bigwedge_{i=1}^{n} (\mathsf{Bel}_{a_i}(\mathsf{Done}(\mu, \bigwedge_{i=1}^{n} \mathsf{role}(a_i))))$$

Weak finalization gives one definition of protocol termination. The correspondence related to weak finalization yields:

$$\text{If } \mathscr{M} \models_{\mathscr{C}} \mathsf{Bel}_a(\mathsf{Done}(\mu, \xi)), \text{ then } \mathscr{M} \models_{\mathscr{C}} \mathsf{Done}(\mu, \xi)$$

Observation 1 *The simulation result specified in Figure 5, NSPK fails to correspond for the participating agents.*

Proof: Recall that the final state $\mathscr{C}_{\mathrm{e}}$ (specified in Figure 5): Thus we have $\mathscr{M} \models_{\mathscr{C}_{\mathrm{e}}} A_1 \wedge A_2 \wedge A_3$. Suppose for contradiction that Bob's view on the execution of NSPK is correct. This means that Bob's belief about the termination should correspond: Assume therefore that $\mathscr{M} \models_{\mathscr{C}_{\mathrm{e}}} \mathsf{Bel}_{\mathrm{Bob}}(\mathsf{Done}(\mathrm{NSPK}, \mathsf{role}(\mathrm{Alice}) \wedge \mathsf{role}(\mathrm{Bob})))$. Since Bob's beliefs corresponds with his environment $\mathscr{C}_{\mathrm{e}}$: $\mathscr{M} \models_{\mathscr{C}_{\mathrm{e}}} \mathsf{Done}(\mathrm{NSPK}, \mathsf{role}(\mathrm{Alice}) \wedge \mathsf{role}(\mathrm{Bob}))$. By weak finalization, we can infer that both $\mathscr{M} \models_{\mathscr{C}_{\mathrm{e}}} \mathsf{Bel}_{\mathrm{Alice}}(\mathsf{Done}(\mathrm{NSPK}, \mathsf{role}(\mathrm{Alice}) \wedge \mathsf{role}(\mathrm{Bob})))$, and $\mathscr{M} \models_{\mathscr{C}_{\mathrm{e}}} \mathsf{Bel}_{\mathrm{Bob}}(\mathsf{Done}(\mathrm{NSPK}, \mathsf{role}(\mathrm{Alice}) \wedge \mathsf{role}(\mathrm{Bob})))$. Yet Alice did not initiate a run with Bob; $\mathscr{M} \models_{\mathscr{C}_{\mathrm{e}}} \neg\, \mathsf{Bel}_{\mathrm{Alice}}\ (\mathsf{Done}(\mathrm{NSPK}, \mathsf{role}(\mathrm{Alice}) \wedge \mathsf{role}(\mathrm{Bob})))$, which gives the contradiction. ■

5.5 Formalizing authentication

Gavin Lowe has given a taxonomy of various authentication properties in [9]. In this section we shall see how Lowe's notions might be embedded in our framework. Note that we interpret the concepts 'initiator' and 'responder' solely by the authentication process. The NSPK protocol is an example of a protocol promising mutual authentication.

Definition 3 *(Lowe 2.1) A protocol μ guarantees to an initiator A* aliveness *of another agent B if whenever A (acting as the initiator) completes a session of the protocol, apparently with responder B, then B has previously been running the protocol as the responder.*

We interpret Lowe's definition of aliveness by the following formalization in $\mathscr{L}_P$:

$$\text{For every } \mathscr{C}, \text{ if } \mathscr{M} \models_{\mathscr{C}} \mathsf{Bel}_A(\mathsf{Done}(\mu, \mathsf{role}(A) \wedge \mathsf{role}(B))), \text{ then } \mathscr{M} \models_{\mathscr{C}} \exists x(\mathsf{Agent}(x) \wedge \mathsf{Bel}_B(\mathsf{Done}(\mu, \mathsf{role}(x) \wedge \mathsf{role}(B)))).$$

Note that our formalization of Definition 3, is slightly stronger than Lowe's definition, since Lowe only require that B has previously been running the protocol. The latter can be made explicit by:

$$\text{For every } \mathscr{C}, \text{ if } \mathscr{M} \models_{\mathscr{C}} \mathsf{Bel}_A(\mathsf{Done}(\mu, \mathsf{role}(A) \wedge \mathsf{role}(B))), \text{ then } \mathscr{M} \models_{\mathscr{C}} \exists x(\mathsf{Agent}(x) \wedge (\mathsf{Bel}_B(\mathsf{Done}(\mu, \mathsf{role}(x) \wedge \mathsf{role}(B))) \vee \mathsf{Bel}_B\ (\mathsf{Done}(\mu, \mathsf{role}(B) \wedge \mathsf{role}(x))))).$$

Definition 4 *(Lowe 2.2) A protocol μ guarantees to an initiator A* weak agreement *of another agent B if whenever A (acting as the initiator) completes a session of the protocol, apparently with responder B, then B has previously been running the protocol, apparently with A.*

$$\textit{For every } \mathscr{C}, \textit{ if } \mathscr{M} \models_{\mathscr{C}} \mathsf{Bel}_A(\mathsf{Done}(\mu, \mathsf{role}(A) \wedge \mathsf{role}(B))), \textit{ then } \mathscr{M} \models_{\mathscr{C}} \mathsf{Bel}_B(\mathsf{Done}(\mu, \mathsf{role}(A) \wedge \mathsf{role}(B)))$$

As observed by Lowe ([9]), NSPK fails to satisfy weak agreement:

Observation 2 *NSPK fails to satisfy weak agreement.*

Proof: From figure 5, clause (A_2) we have directly that $\mathscr{M} \models_{\mathscr{C}_{\mathrm{e}}} \mathsf{Bel}_{\mathrm{Bob}}(\mathsf{Done}(\mathrm{NSPK}, \mathsf{role}(\mathrm{Alice}) \wedge \mathsf{role}(\mathrm{Bob})))$, yet we also have by clause (A_4) that $\neg\mathscr{M} \models_{\mathscr{C}_{\mathrm{e}}} \mathsf{Bel}_{\mathrm{Alice}}(\mathsf{Done}(\mathrm{NSPK}, \mathsf{role}(\mathrm{Alice}) \wedge \mathsf{role}(\mathrm{Bob})))$. ■

$$
\begin{array}{l}
\mathsf{protocol}[\mathrm{SRA}, 0, \mathsf{role}(A) \wedge \mathsf{role}(B), \mathsf{role}(A) \wedge \mathsf{role}(B), \mathsf{role}(A), \\
\quad \mathsf{Transmit}(A, B, E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{MSG}, A)]) \qquad (R_1) \\
\mathscr{B}\ \mathsf{Transmit}\,(B, A, E[\mathsf{key}(\mathsf{a}, \mathsf{u}, B) : E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{MSG}, A)]]) \quad (R_2) \\
\mathscr{B}\ \mathsf{Transmit}\,(A, B, E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{MSG}, A)]) \qquad (R_3) \\
\mathscr{B}\,\varepsilon]
\end{array}
$$

Figure 6. Shamir Rivest Adelman in $\mathscr{L}_P$.

$$
\begin{array}{l}
\mathsf{protocol}[\mathrm{SRAattack2}, 0, \\
\quad \mathsf{role}(A) \wedge \mathsf{role}(B) \wedge \mathsf{role}(I), \mathsf{role}(I), \mathsf{role}(A), \\
\quad \mathsf{Transmit}(A, \mathbb{im}(I, B), E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{MSG}, A)]) \quad (R.1.1) \\
\mathscr{B}\ \mathsf{Transmit}\,(\mathbb{im}(I, B), A, E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{MSG}, A)]) \quad (R.2.1) \\
\mathscr{B}\ \mathsf{Transmit}\,(A, \mathbb{im}(I, B), \mathsf{Text}(\mathrm{MSG}, A)) \qquad (R.2.2) \\
\mathscr{B}\ \mathsf{Transmit}\,(\mathbb{im}(I, B), A, \mathsf{Text}(\mathrm{Bogus}, I)) \qquad (R.1.2) \\
\mathscr{B}\ \mathsf{Transmit}\,(A, \mathbb{im}(I, B), E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{Bogus}, I)]) \quad (R.1.3) \\
\mathscr{B}\,\varepsilon]
\end{array}
$$

Figure 7. Attack on Shamir Rivest Adelman.

$$
\begin{array}{l}
\quad \mathsf{protocol}\ [\mathrm{SRA\ attack2}, 0, \\
\quad\quad \mathsf{role}(A) \wedge \mathsf{role}(B) \wedge \mathsf{role}(I),\ \mathsf{role}(I),\ \mathsf{role}(A), \\
\mathsf{Enforce}_A(\mathsf{Bel}_A(\mathsf{newText}(\mathrm{MSG}, A))) \qquad (1) \\
\mathscr{B}\ \mathsf{Bel}_A\ (\mathsf{isKey}(\mathsf{key}(\mathsf{a}, \mathsf{u}, A))) \qquad (2) \\
\mathscr{B}\ \mathsf{Enforce}_A\ (\mathsf{Bel}_A(E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{MSG}, A)])) \qquad (3) \\
\mathscr{B}\ \mathsf{Bel}_A\ (\mathsf{Agent}(B)) \qquad (4) \\
\mathscr{B}\ \mathsf{Transmit}\ (A, \mathbb{im}(I, B), E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{MSG}, A)]) \qquad (5) \\
\mathscr{B}\ \mathsf{Bel}_I\ (\mathsf{Agent}(A)) \qquad (6) \\
\mathscr{B}\ \mathsf{Enforce}_I\ (\mathsf{Bel}_I(\mathrm{Trust}(I, A, E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{MSG}, A)]))) \qquad (7) \\
\mathscr{B}\ \mathsf{Bel}_I\ (E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{MSG}, A)]) \qquad (8) \\
\mathscr{B}\ \mathsf{Transmit}\ (\mathbb{im}(I, B), A, E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{MSG}, A)]) \qquad (9) \\
\mathscr{B}\ \mathsf{Enforce}_A\ (\mathsf{Bel}_A(\mathrm{Trust}(A, B, \\
\quad\quad E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{MSG}, A)]))) \qquad (10) \\
\mathscr{B}\ \mathsf{Bel}_A\ (\mathsf{isKey}(\mathsf{key}(\mathsf{a}, \mathsf{i}, A))) \qquad (11) \\
\mathscr{B}\ \mathsf{Enforce}_A\ (\mathsf{Bel}_A(\\
\quad\quad D[\mathsf{key}(\mathsf{a}, \mathsf{i}, A) : E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{MSG}, A)]])) \qquad (12) \\
\mathscr{B}\ \mathsf{Transmit}\ (A, \mathbb{im}(I, B), \mathsf{Text}(\mathrm{MSG}, A)) \qquad (13) \\
\mathscr{B}\ \mathsf{Enforce}_I\ (\mathsf{Bel}_I(\mathrm{Trust}(I, A, \mathsf{Text}(\mathrm{MSG}, A)))) \qquad (14) \\
\mathscr{B}\ \mathsf{Bel}_I\ (\mathsf{Text}(\mathrm{MSG}, A)) \qquad (15) \\
\mathscr{B}\ \mathsf{Enforce}_I\ (\mathsf{Bel}_I(\mathsf{newText}(\mathrm{Bogus}, I))) \qquad (16) \\
\mathscr{B}\ \mathsf{Transmit}\ (\mathbb{im}(I, B), A, \mathsf{Text}(\mathrm{Bogus}, I)) \qquad (17) \\
\mathscr{B}\ \mathsf{Enforce}_A\ (\mathsf{Bel}_A(\mathrm{Trust}(A, B, \mathsf{Text}(\mathrm{Bogus}, I)))) \qquad (18) \\
\mathscr{B}\ \mathsf{Bel}_A\ (\mathsf{Text}(\mathrm{Bogus}, I)) \qquad (19) \\
\mathscr{B}\ \mathsf{Enforce}_A\ (\mathsf{Bel}_A(E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{Bogus}, I)])) \qquad (20) \\
\mathscr{B}\ \mathsf{Transmit}\ (A, \mathbb{im}(I, B), E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{Bogus}, I)]) \qquad (21) \\
\mathscr{B}\ \mathsf{Enforce}_I\ (\mathsf{Bel}_I(\mathrm{Trust}(I, A, E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{Bogus}, I)]))) \qquad (22) \\
\mathscr{B}\ \mathsf{Bel}_I\ (E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{Bogus}, I)]) \qquad (23) \\
\mathscr{B}\,\varepsilon]
\end{array}
$$

Figure 8. Automated refinement of the attack on Shamir Rivest Adelman.

6 Shamir Rivest Adelman Three Pass

In this section we shall investigate another application of attack simulation: validation of attacks on authentication protocols. The Shamir Rivest Adelman protocol (SRA) [2, p. 64] assumes that encryption is commutative, which means $E[k_1 : E[k_2 : F]] = E[k_2 : E[k_1 : F]]$.

$$
\begin{array}{lcl}
(\mathrm{SRA}_1) & A \longrightarrow B & : \quad E(K_A : M) \\
(\mathrm{SRA}_2) & B \longrightarrow A & : \quad E(K_B : E(K_A : M)) \\
(\mathrm{SRA}_3) & A \longrightarrow B & : \quad E(K_B : M)
\end{array}
$$

The protocol is transfered to $\mathscr{L}_P$ by The second attack in Clark/Jacob involves two interleaving sessions.

$$
\begin{array}{lcl}
(R.1.1) & A \longrightarrow I(B) & : \quad E(K_A : M) \\
(R.2.1) & I(B) \longrightarrow A & : \quad E(K_A : M) \\
(R.2.2) & A \longrightarrow I(B) & : \quad M \\
(R.1.2) & I(B) \longrightarrow A & : \quad bogus \\
(R.1.3) & A \longrightarrow I(B) & : \quad E(K_A : bogus)
\end{array}
$$

Although this attack on the SRA protocol is erroneous, it is not possible to detect the attack through static validation as was reported in [7]. Fortunately attack simulation uncovers the two flaws in the attack. The second attack presented in the report contains two protocol jumps. The attack was specified in $\mathscr{L}_P$ as described in Figure 7. The attack includes three roles initiator A, responder B and the attacker I. The automated refinement of the previous attack specification is shown in Figure 8. When validating the refined attack specification, no error is reported. After configuring a scenario involving two good agents Alice and Bob possessing the SRA protocol, and an intruder agent Malice that possesses the attack protocol, the scenario is started with Alice as the initiator of the authentication. Let this initial configuration be denoted $\mathscr{C}_{\mathrm{init}}$. The simulation stops in a configuration $\mathscr{C}_{\mathrm{end}}$ with two unresolved execution predicates $\mathbb{E}$ inside Alice (recall Section 3.2), and one unresolved execution predicate inside Malice. Malice is waiting to intercept a message to be sent by Alice $(R.2.2)$, (line 13 in Figure 8):

$$
\begin{array}{l}
\mathscr{M} \models_{\mathscr{C}_{\mathrm{end}}} \mathsf{Bel}_{\mathsf{Malice}}(\mathbb{E}(\mathsf{Malice}, \mathsf{protocol}[\mathrm{SRA\ attack2}, 1, \\
\mathsf{role}(\mathsf{Alice}) \wedge \mathsf{role}(\mathsf{Bob}) \wedge \mathsf{role}(\mathsf{Malice}), \mathsf{role}(I), \mathsf{role}(A), \\
\quad\quad \mathsf{Transmit}(\mathsf{Alice}, \mathbb{im}(\mathsf{Malice}, \mathsf{Bob}), \\
\quad\quad\quad \mathsf{Text}(\mathrm{MSG}, \mathsf{Alice}))\,\mathscr{B}\,\Phi'])))
\end{array}
$$

where the protocol header includes the protocol name $\mathrm{SRA\ attack2}$, the session number 1, instantiated with the three agents Alice, Bob Malice, playing the roles A, B, I respectively. Malice can only play the attacker role I, and there is one start role A. Finally Φ' denotes the rest of the instantiated attack protocol. In the configuration $\mathscr{C}_{\mathrm{end}}$, the inbuffer of Malice contains the message:

$$
\begin{array}{c}
\mathsf{msg}\ E[\mathsf{key}(\mathsf{a}, \mathsf{u}, \mathsf{Alice}) : E[\mathsf{key}(\mathsf{a}, \mathsf{u}, \mathsf{Alice}) : \mathsf{Text}(\mathrm{MSG}, \mathsf{Alice})]] \\
\mathsf{from}\ \mathsf{Alice}\ \mathsf{to}\ \mathbb{im}(\mathrm{Malice}, \mathrm{Bob})
\end{array}
$$

which indicates that Alice has responded to the second session of the protocol, the request $(R.2.1)$, with the encryption $E(K_A : E(K_A : M))$, originally intended to be sent from Alice to Bob. Alice expected $E(K_B : M)$, yet she has no way of discovering that she was fooled and instead

got the cipher text $E(K_A : M)$. Hence we have both

$$\mathscr{M} \models_{\mathscr{C}_{\mathrm{end}}} \mathsf{Bel}_{\mathrm{Alice}}(\mathbb{E}(\mathrm{Alice}, \mathsf{protocol}[\mathrm{SRA}, 1, \ldots, \mathsf{Transmit}(\mathrm{Bob}, \mathrm{Alice}, E[\mathsf{key}(\mathsf{a}, \mathsf{u}, \mathrm{Alice}) : \mathsf{Text}(\mathrm{MSG}, \mathrm{Alice})])\, \mathscr{B}\, \Phi'']))$$

and

$$\mathscr{M} \models_{\mathscr{C}_{\mathrm{end}}} \mathsf{Bel}_{\mathrm{Alice}}(\mathbb{E}(\mathrm{Alice}, \mathsf{protocol}[\mathrm{SRA}, 2, \ldots, \mathsf{Transmit}(\mathrm{Bob}, \mathrm{Alice}, E[\mathsf{key}(\mathsf{a}, \mathsf{u}, B^{\%}) : E[\mathsf{key}(\mathsf{a}, \mathsf{u}, \mathrm{Alice}) : \mathsf{Text}(\mathrm{MSG}, \mathrm{Alice})]])\, \mathscr{B}\, \Phi''']))$$

where the header is supressed and where Φ'' and Φ''' denote the respective instantiated protocol tails. In the first session $(\mathrm{SRA}, 1)$, Alice plays the responder role B, while the second session $(\mathrm{SRA}, 2)$ she plays the initiator A. Thus in the first session Alice does never receive the appropriate final message instance from Bob or by Malice impersonating Bob. In the second session Alice is waiting for an instance of the second message (SRA_2), which never appear in Alice's inbuffer, Malice is blocked by the attack clause $(R.2.2)$. Reachability analysis of the attack shows that there are no more simulations than the one previously described. In other words, reachability analysis shows that

$$\mathscr{M} \models_{\mathscr{C}_{\mathrm{init}}} \boxdot \neg\, \mathsf{Bel}_{\mathrm{Malice}}\, (\mathsf{Done}(\mathrm{SRAattack2}, \mathsf{role}(\mathrm{Alice}) \wedge \mathsf{role}(\mathrm{Bob}) \wedge \mathsf{role}(\mathrm{Malice}))),$$

hence the attack can never succeed. The fact that Alice is not able to finalize any intended session is also a question of reachability, it can justified (by the model checker in Maude) that

$$\mathscr{M} \models_{\mathscr{C}_{\mathrm{init}}} \boxdot \neg\, \mathsf{Bel}_{\mathrm{Alice}}\, (\mathsf{Done}(\mathrm{SRA}, \mathsf{role}(\mathrm{Alice}) \wedge \mathsf{role}(\mathrm{Bob}))).$$

To conclude, the attack contains two severe errors, and there is no obvious way to repair the specification.

7 Conclusion

We have shown how a simulator for executing security protocols can be extended to simulate attacks on these protocols, in particular Lowe's attack on the Needham-Schroeder protocol. All the security properties used in standard protocol analysis can be expressed within our framework. The attack simulator has also proven be useful in checking the correctness of attacks: the second attack on the Shamir Rivest Adelman protocol could only be detected by simulation, not by static validation [7]. The reason is that some flaws in attacks arise from the fact that good agents do not follow the intended protocol. There is a connection between static validation and simulations of attack protocols: if the simulation of an attack specification P shows that the attack can be executed, then the validation algorithm will succeed for P too.

References

[1] David Basin, Sebastian Mödersheim, and Luca Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, June 2005.

[2] J. Clark and J. Jacob. A Survey of Authentication Protocol Literature, 1997. Version 1.0, Unpublished Report, University of York, `http://cs.york.ac.uk/~jac/papers/drareview.ps.gz`.

[3] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. Maude Manual Version 2.2. `http://maude.cs.uiuc.edu/maude2-manual/html/`, 2005.

[4] Millen J. Denker G. and Ruess H. The CAPSL integrated protocol environment. Technical Report SRI-CLS-2000-02, SRI, 2000.

[5] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge*. MIT Press, 2003.

[6] Anders Moen Hagalisletto. Automated Refinement of Security Protocols. In *Proceedings of 20th IEEE International Parallel and Distributed Processing Symposium 2006*, 2006. Workshop SSN 2006.

[7] Anders Moen Hagalisletto. Errors in Attacks on Authentication Protocols, 2007. Accepted for publication in IEEE The Second International Conference on Availability, Reliability and Security (ARES 2007) .

[8] Gavin Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In *Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems*, pages 147–166. Springer-Verlag, 1996.

[9] Gavin Lowe. A hierarchy of authentication specifications. In *PCSFW: Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.

[10] Gavin Lowe. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6:53–84, 1998.

[11] Wenbo Mao. *Modern Cryptography: Theory and Practice*. Prentice Hall, 2004.

[12] Roger M. Needham and Michael D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Commun. ACM*, 21(12):993–999, 1978.