INTEGRATING LOGICS AND PROCESS CALCULI FOR CRYPTOGRAPHIC PROTOCOL ANALYSIS

Mauricio Papa *, Oliver Bremer, John Hale, Sujeet Shenoi
University of Tulsa, Center for Information Security, Computer Science Dept.
{mauricio-papa, oliver-bremer, john-hale, sujeet}@utulsa.edu

Abstract This paper describes a formalism for cryptographic protocol simulation and analysis that integrates logic and process calculus components. Novel features include the comprehensive modeling of encrypted and unencrypted messages, an expressive message passing semantics and sophisticated constructs for modeling principals. Moreover, the seamless integration of inference rules for communication, reduction and information analysis supports formal proofs about the knowledge and behavior of principals, and about the properties of protocols.

Keywords: Formal methods, authentication, encryption, process calculi, cryptographic protocols

1. Introduction

The comprehensive analysis of cryptographic protocols requires modeling the messages exchanged by principals, the knowledge held by principals and the behavior of principals [4,10,14,15]. Logic-based techniques, e.g., BAN logic [5,6] and its derivatives [7,16], model messages and the beliefs of principals. However, they do not provide mechanisms for capturing and reasoning about the behavior of principals. Also, protocols must be idealized, i.e., abstracted, before analysis. Idealizing a protocol is difficult and prone to misinterpretation and errors.

Process calculus techniques, e.g., Spi calculus [2,3], on the other hand, focus on the behavior of principals (agents in a process calculus [1,13]). But they lack constructs for modeling the knowledge held by principals and reasoning about this knowledge. Thus, only limited properties can be proven about protocols.

*To whom correspondence should be addressed (email: *mauricio-papa@utulsa.edu*).
Research supported by MPO Contracts MDA904-96-1-0114, MDA904-96-1-0115 and MDA904-98-C-A900.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35691-4_52

D. Gritzalis et al. (eds.), *Security and Privacy in the Age of Uncertainty*

The integrated approach described in this paper draws features from process calculus and logic based approaches. Process calculus components are used to formally model the behavior and evolution of principals using concurrent communicating agents. Logic components are employed to simulate protocols and reason about the knowledge and behavior of agents (principals), and about the properties of protocols. The integrated approach can prove that protocols are vulnerable to attacks, thereby complementing state enumeration techniques [11,12] used to identify protocol vulnerabilities.

The following section defines communication syntax and semantics. Next, principals are formally modeled as concurrent agents, as in process calculi. Sections 4 and 5 present inference rules for agent and protocol analysis. In Section 6, the Needham-Schroeder Protocol [8,9,15] and a "middle person" attack are used to illustrate the power of the formalism. Finally, Section 7 presents our conclusions.

2. Modeling Communication

Communication between principals in protocols is modeled as synchronous message passing between agents. The sender outputs a "message" that matches a "pattern" exposed by the receiving agent. This section describes the syntax and semantics of agent communication. The synchronous model can express asynchronous communications and complex, encrypted messages.

Definition 2.1: A *key* ($key \in KEY$) is a public/private key (K_n/K_n^{-1}), a shared or secret key (K_n^s), the concatenation of two keys ($K_n{:}\ K_m$), a placeholder for a key that is not yet known ($key?$), or nil, for an unencrypted message:

$$key \quad ::= \quad K_n \mid K_n^{-1} \mid K_n^s \mid K_n{:}\ K_m \mid key? \mid nil$$

We assume the existence of a basic type $NAME$ comprising an infinite set of *names*. This basic type is used to create unique keys, and data for messages and patterns. For example, n and m in Definition 2.1 are of type $NAME$ ($n, m \in NAME$). Note that the concatenation of keys can be used to model k of n protocols, e.g., Clipper Key Escrow.

Definition 2.2: The key matching operator ($\overset{K}{\sim}$) is defined by the following rules:

$K_a \overset{K}{\sim} K_b^{-1}$ iff $a = b$ $\qquad$ $K_a^s \overset{K}{\sim} K_b^s$ iff $a = b$ $\qquad$ $nil \overset{K}{\sim} nil.$

Messages are defined as nested tuples of "values" encrypted under a key. Values can be keys, messages, names (representing data) or "fresh names" (representing nonces and time stamps).

$$
\begin{array}{rcl}
message & ::= & lstv_{key} \\
pattern & ::= & lstp_{key} \mid key \mid n? \mid n \\
lstv & ::= & \{\, value^* \,\} \\
lstp & ::= & \{\, pattern^* \,\} \\
value & ::= & key \mid message \mid n \mid \#n \\
key & ::= & K_n \mid K_n^{-1} \mid K_n^s \mid K_n : K_m \mid \\
 & & key? \mid nil
\end{array}
$$

Figure 1. Message and pattern syntax

Definition 2.3: A *message* ($message \in MSG$) is a list of values $\{v_1, v_2, ..., v_j\}$ encrypted under *key* and denoted by $\{v_1, v_2, ..., v_j\}_{key}$. A *value* ($value \in VAL$) is a key ($key$), a message ($message$), a name ($n \in NAME$) or a fresh name ($\#n$ where $n \in NAME$):

$message ::= \{v_1, v_2, ..., v_j\}_{key} \qquad value ::= key \mid message \mid n \mid \#n$

Patterns permit the capture of messages and (possibly) their contents. A pattern exposed by a receiver expresses the knowledge it possesses about the format and content of the message.

Definition 2.4: A *pattern* ($pattern \in PAT$) is a list of patterns $\{p_1, p_2, ..., p_j\}$ encrypted under *key* ($\{p_1, p_2, ..., p_j\}_{key}$), a key ($key$), a wildcard or placeholder ($n?$) for capturing values, or a name (n):

$$pattern ::= \{p_1, p_2, ..., p_j\}_{key} \mid key \mid n? \mid n$$

A BNF syntax for messages and patterns is presented in Figure 1. We now define the matching of messages and patterns, fundamental to agent communication.

Definition 2.5: The *message-pattern matching operator* ($\sim$) is defined by the following rules ($v, v_i \in VAL$, $p, p_i \in PAT$, $key, key_i \in KEY$, $m \in MSG$ and $n \in NAME$):

$$
\begin{array}{rcl}
\{v_1, v_2, ..., v_j\}_{key_1} & \sim & \{p_1, p_2, ..., p_j\}_{key_2} \text{ iff } key_1 \overset{K}{\sim} key_2 \wedge v_i \sim p_i \\
m & \sim & n? \\
v & \sim & n? \\
v & \sim & n \text{ iff } v = n \\
key & \sim & key? \\
key_1 & \sim & key_2 \text{ iff } key_1 = key_2
\end{array}
$$

3. Modeling Principals

Modeling the behavior of principals is required for the comprehensive analysis of protocols. Most conventional techniques [5,6,7,16] concentrate on messages exchanged in protocols, but do not model the behavior of principals. Our approach, with its richer semantics, models messages and the behavior of principals. A "system" is defined as the concurrent composition of communicating "agents" that represent principals.

Definition 3.1: A *system* is defined as zero or more concurrently executing agents (& denotes the concurrency operator):

$$system \quad ::= \quad agent \,\&\, system \mid nil$$

Definition 3.2: An *agent* is characterized by its id, $id \in Name$, a list of values, $[lstv]$, representing its knowledge, and a list of concurrent communication sequences, $(cseq)$:

$$agent \quad ::= \quad id[lstv](cseq)$$

An agent engages in concurrent "communication sequences." Each sequence, called an "annotated sequence," represents a distinct protocol run. An annotated sequence comprises a "sequence" of message outputs and/or pattern exposures and an "annotation" that contains the fresh names (nonces or time stamps) associated with the particular sequence.

Definition 3.3: Let $m \in MSG$, $p \in PAT$ and $lstn$: List of $n \in name$, then a *concurrent sequence* $(cseq)$, *annotated sequence* $(aseq)$ and *sequence* (seq) of messages and/or patterns are defined by:

$$\begin{aligned} cseq &::= aseq \diamond cseq \mid nil \\ aseq &::= [seq].[lstn] \mid [seq]_\infty \\ seq &::= \hat{}m\ seq \mid\!\rightarrow p\ seq \mid nil \end{aligned}$$

where $\diamond$ is the commutative concurrency operator for sequences.

The definition above defines a sequence (seq) as empty (nil) or a sequence with an output message $(\hat{}mseq)$ or an exposed pattern $(\rightarrow pseq)$. An annotated sequence $(aseq)$ is a sequence of messages and/or patterns followed by a list of names $([seq].[lstn])$; the list $[lstn]$ stores fresh names for the corresponding sequence $([seq])$. The term $[seq]_\infty$ for an annotated sequence denotes a sequence that has to be executed repeatedly, e.g., to model a server. Note that when unfolding an infinite sequence, an unmodified copy is always maintained, i.e., $[seq]_\infty \equiv [seq].[nil] \diamond [seq]_\infty$. A concurrent sequence $(cseq)$ is the concurrent composition $(\diamond)$ of an annotated sequence and a concurrent sequence. The syntax for specifying systems of concurrent, communicating agents is presented in Figure 2.

To illustrate the expressive power of the approach, we formally model the Needham-Schroeder Protocol [5,15] shown in Figure 3. Note that the conventional notation only describes the contents of messages and indicates the message senders and intended receivers. It does not specify the behavior of principals as the protocol progresses.

Figure 4 presents formal definitions of the three participating agents. The *system* is defined as the concurrent composition (&) of agents a, b and s. Note that corresponding principals and agent ids are written in upper case, i.e., A, B and S, respectively.

Agent a is defined by its id A, its list of values $[lstv_A]$ and a single concurrent sequence comprising five actions and a null list of fresh names $(lstn)$: $([\hat{}\{\cdot\} \rightarrow \{\cdot\}\hat{}\{\cdot\} \rightarrow \{\cdot\}\hat{}\{\cdot\}].[\,])$ The five actions are: (i) an output to send $\{A, B\}$ to S (Step 1 in Figure 3), (ii) a pattern exposure to

$$
\begin{array}{rcl}
system & ::= & agent \;\&\; system \mid nil \\
agent & ::= & id[lstv](cseq) \\
cseq & ::= & aseq \diamond cseq \mid nil \\
aseq & ::= & [\,seq\,].[\,lstn\,] \mid [\,seq\,]_{\infty} \\
seq & ::= & \hat{}\, message\ seq \mid \rightarrow pattern\ seq \mid nil \\
lstn & ::= & \{\, n^{*} \,\}
\end{array}
$$

Figure 2. Agent system syntax

$$
\begin{array}{lcll}
1. & A & \longrightarrow & S: \{A, B\} \\
2. & S & \longrightarrow & A: \{K_b, B\}_{K_s^{-1}} \\
3. & A & \longrightarrow & B: \{N_a, A\}_{K_b} \\
4. & B & \longrightarrow & S: \{B, A\} \\
5. & S & \longrightarrow & B: \{K_a, A\}_{K_s^{-1}} \\
6. & B & \longrightarrow & A: \{N_a, N_b\}_{K_a} \\
7. & A & \longrightarrow & B: \{N_b\}_{K_b}
\end{array}
$$

Figure 3. Needham-Schroeder Protocol

receive $\{K_b, B\}_{K_s^{-1}}$ from S (Step 2), (iii) an output to send $\{\#N_a, A\}_{K_b}$ to B (Step 3), (iv) a pattern exposure to receive $\{N_a, \#N_b\}_{K_a}$ from B (Step 6), and (v) an output to send $\{N_b\}_{K_b}$ to B (Step 7). Agents b and s are defined similarly.

Note that as the agents evolve with each communication, their $lstv$'s and $lstn$'s are updated accordingly. For example, $lstv_A$, representing the knowledge held by agent a, initially contains nil key (for unencrypted messages), K_a^{-1} (a's private key), and K_s (s's public key), i.e., $lstv_A = \{nil, K_a^{-1}, K_s\}$. After Step 2, $lstv_A$ is updated with K_b. Likewise, agent a's $lstn_A$ is originally empty. It is updated with the fresh name (nonce) N_a after Step 3. Finally, note that bindings to wildcard variables occur when agents communicate. For example, when agent a sends the message $\{A, B\}$ to server s in Step 1, s's wildcards $id1?$ and $id2?$ are bound to the agent ids A and B, respectively.

4. Inference Rules

Inference rules specify a formal semantics of agent behavior. Three groups of inference rules are specified: (i) agent communication and reduction rules, (ii) agent knowledge rules, and (iii) system development rules.

Definition 4.1: Agent communication and reduction are defined by the following inference rules:

$$\textbf{In}: \frac{m \sim p}{id[lstv]([\rightarrow p\ seq].[lstn] \diamond cseq) \xrightarrow{m} id[lstv \cup m]([seq].[lstn] \diamond cseq\{p/m\})}$$

$$\textbf{Out}: \frac{}{id[lstv]([\hat{}m\ seq].[lstn] \diamond cseq) \xrightarrow{\bar{m}} id[lstv \cup m \cup fresh(m)]([seq].[lstn \cup fresh(m)] \diamond cseq)}$$

$$\textbf{Comm}: \frac{a_1 \xrightarrow{\bar{m}} a_1',\ a_2 \xrightarrow{m} a_2'}{a_1\ \&\ a_2 \Longrightarrow a_1'\ \&\ a_2'}$$

$$
\begin{array}{lcl}
system & ::= & a \;\&\; b \;\&\; s \\
a & ::= & A[lstv_A]([\hat{}\{A, B\} \rightarrow \{keyB?, B\}_{K_s} \hat{}\{\#N_a, A\}_{keyB} \rightarrow \{N_a, nonceB?\}_{K_a^{-1}} \hat{} \\
 & & \{nonceB\}_{keyB}].[\,]) \\
b & ::= & B[lstv_B]([\rightarrow \{nonce?, id?\}_{K_b^{-1}} \hat{}\{B, id\} \rightarrow \{key?, id\}_{K_s} \hat{}\{nonce, \#N_b\}_{key} \rightarrow \\
 & & \{N_b\}_{K_b^{-1}}].[\,]) \\
s & ::= & S[lstv_S]([\rightarrow \{id1?, id2?\} \hat{}\{K_{id2}, id2\}_{K_s^{-1}} \rightarrow \{id2, id1\} \hat{}\{key_{id1}, id1\}_{K_s^{-1}}]\infty)
\end{array}
$$

Figure 4. Agent description (Needham-Schroeder Protocol).

The following notation is used. Agent communication offers are denoted by $\xrightarrow{\alpha}$, where α is m (for a pattern exposure) or $\bar{m}$ (for a message output). Single- and multi-step reductions are written as $\Longrightarrow$ and $\overset{*}{\Longrightarrow}$, respectively. The symbol $\equiv$ denotes agent equivalence.

The **In** rule defines the behavior of an agent that exposes a pattern and receives a message. The precondition $m \sim p$ requires the message m to match the pattern p exposed by the agent $id[lstv](\cdot)$. The postcondition states that, after the message is accepted, the annotated sequence $([\rightarrow p\ seq].[lstn]$ of the agent becomes $[seq].[lstn]$ and all free occurrences of wildcards in p in the remainder of the agent's concurrent sequences are replaced by m. Moreover, since the agent has gained new knowledge m, $lstv$ is augmented with m ($\cup$ denotes concatenation).

The **Out** rule specifies the behavior of an agent outputting a message m. On offering message m, the annotated sequence $([\hat{}m seq].[lstn] \diamond cseq)$ of agent $id[lstv]$ reduces to $([seq].[lstn \cup fresh(m)] \diamond cseq)$, where all fresh names in m are added to $lstn$. Similarly, the list of values $lstv$ held by the agent is updated with the offered message to produce $lstv \cup m \cup fresh(m)$. Note that the output message m and the fresh names contained in this message constitute potentially new knowledge that must be added to the agent's $lstv$.

The **Comm** rule defines reduction for a pair of communicating agents. Two preconditions must be satisfied for a_1 & a_2 to reduce to a_1' & a_2' after communicating message m. First, agent a_1 must be able to reduce to agent a_1' with output message $\bar{m}$. Second, agent a_2 must be able to reduce to agent a_2' with input pattern m.

Definition 4.2: The inference rules **Knows**, **Extract** and **Construct** are defined by:

$$\textbf{Knows}: \frac{(id[lstv](cseq) \equiv id[lstv'](cseq),\ id[lstv'](cseq) \text{ knows } v) \vee (v \in lstv)}{id[lstv](cseq) \text{ knows } v}$$

$$\textbf{Extract}: \frac{id[lstv](cseq) \text{ knows } \{v_1, \ldots, v_n\}_{k1},\ id[lstv](cseq) \text{ knows } k_2,\ k_1 \sim k_2}{id[lstv](cseq) \equiv id[lstv \cup v_1 \cup \ldots \cup v_n](cseq)}$$

$$\textbf{Construct}: \frac{id[lstv](cseq) \text{ knows } v_1, \ldots, v_n,\ id[lstv](cseq) \text{ knows } k}{id[lstv](cseq) \equiv id[lstv \cup \{v_1, \ldots, v_n\}_k](cseq)}$$

The **Knows** rule expresses predicates of the form "agent $id[lstv](cseq)$ knows value v." The precondition states that an agent knows v if it can be transformed into an equivalent agent that knows v or if v is in the agent's $lstv$.

The **Extract** rule extracts values from inside a message held by an agent in its $lstv$. The precondition requires the agent to hold a matching key to access the message components. The agent's $lstv$ is augmented with the extracted components as specified in the rule conclusion.

The **Construct** rule creates new values from values that are known to an agent. The newly constructed values are concatenated to the agent's $lstv$. Note that changes to $lstv$ due to the **Extract** and **Construct** rules do not create new agents; they only change the representations of what the agents know. The remaining two inference rules, **Chain** and **ChainBase**, define reductions for systems of concurrent agents with multiple communication steps.

Definition 4.3: The inference rules **Chain** and **ChainBase** are defined by:

$$\textbf{Chain}: \frac{a_1 \;\&\; a_2 \Longrightarrow a_1' \;\&\; a_2',\ a_1' \;\&\; a_2' \;\&\; a_3 \;\&\; \ldots \;\&\; a_n \stackrel{*}{\Longrightarrow} a_1'' \;\&\; a_2'' \;\&\; a_3 \;\&\; \ldots \;\&\; a_n}{a_1 \;\&\; a_2 \;\&\; a_3 \;\&\; \ldots \;\&\; a_n \stackrel{*}{\Longrightarrow} a_1'' \;\&\; a_2'' \;\&\; a_3 \;\&\; \ldots \;\&\; a_n}$$

$$\textbf{ChainBase}: \frac{a_1 \equiv a_1'', a_2 \equiv a_2'', \ldots, a_n \equiv a_n''}{a_1 \;\&\; a_2 \;\&\; \ldots \;\&\; a_n \stackrel{*}{\Longrightarrow} a_1'' \;\&\; a_2'' \;\&\; \ldots \;\&\; a_n''}$$

The **Chain** rule models pseudotransitivity, i.e., $X \Longrightarrow Y \wedge YW \stackrel{*}{\Longrightarrow} Z \models XW \stackrel{*}{\Longrightarrow} Z$. The **ChainBase** rule serves as a base case for the recursive application of the **Chain** rule. It simply states that two systems containing equivalent agents can reduce to each other.

5. Protocol Analysis

The formal analysis of protocols involves modeling principals using the agent syntax, and applying the inference rules, **In**, **Out**, **Comm**, **Chain** and **ChainBase**, to simulate agent behavior and the inference rules, **Knows**, **Extract** and **Construct**, to reason about the knowledge held by agents as the protocol progresses.

As an example, we model and analyze Step 1 of the Needham-Schroeder Protocol where agent a sends agent s the unencrypted message $\{A, B\}$ (Figure 3). The first part of the analysis of Step 1 is to show that the concurrent system $a \;\&\; s$ evolves to the reduced system $a^1 \;\&\; s^1$ after a single communication. Then, we prove that s^1 knows the individual message values, i.e., "s^1 knows A, B."

We use the **Comm** rule to show that "$a \;\&\; s \Longrightarrow a^1 \;\&\; s^1$." According to this rule, the postcondition $a \;\&\; s \Longrightarrow a^1 \;\&\; s^1$ holds if $a \xrightarrow{\bar{m}} a_1$ and $s \xrightarrow{m} s_1$ are true. Thus, it has to be proven that agent a reduces to a^1 by outputting the message $m = \{A, B\}$ and agent s reduces to s^1 after receiving m. Figure 5 shows the sequence of rule applications.

On instantiating the **Out** rule for agent a, we obtain:

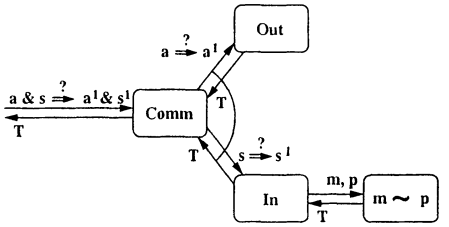

Figure 5. "$a \;\&\; s \Longrightarrow a^1 \;\&\; s^1$."

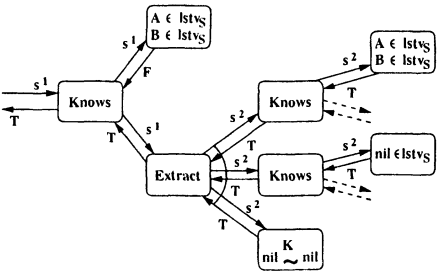

Figure 6. "s^1 knows A, B."

$$\textbf{Out}: \frac{}{A[lstv_A]([\ulcorner\{A,B\}\ldots].[]) \xrightarrow{\overline{\{A,B\}}} A[lstv_A \cup \{A,B\}]([\ldots].[])}$$

Thus, the reduced agent a^1 is given by $a^1 = A[lstv_A \cup \{A,B\}]([\ldots].[])$. The corresponding instantiation of the **In** rule for agent s is:

$$\textbf{In}: \frac{\{A,B\} \sim \{id1?, id2?\}}{S[lstv_S]([\rightarrow \{id1?, id2?\} \ldots]_\infty) \xrightarrow{\{A,B\}} S[lstv_S \cup \{A,B\}]([\ldots]_\infty\{\{id1?, id2?\}/\{A,B\}\})}$$

According to the **In** rule's precondition, agent s must expose a pattern $p = \{id1?, id2?\}$ that matches $m = \{A,B\}$. Since $m \sim p$ according to Definition 2.5, the postcondition holds and agent s reduces to $s^1 = S[lstv_S \cup \{A,B\}]([\ldots]_\infty\{\{id1?, id2?\}/\{A,B\}\})$.

Since the **Out** and **In** rules evaluate to true, **Comm** also returns true. Thus, "$a \;\&\; s \Longrightarrow a^1 \;\&\; s^1$."

Now we show that "s^1 knows A, B" – equivalent to proving that "s^1 knows A" and "s^1 knows B." Figure 6 illustrates the proof for both parts. Note that proving "s^1 knows A" (resp. "s^1 knows B") requires testing the condition $A \in lstv_S$ (resp. $B \in lstv_S$).

First, the **Knows** rule is instantiated to show that "s^1 knows A."

$$\textbf{Knows}: \frac{(S[lstv_S \cup \{A,B\}]([\ldots]_\infty) \equiv S[lstv'_S]([\ldots]_\infty),\ S[lstv'_S]([\ldots]_\infty) \text{ knows } A) \vee (A \in lstv_S \cup \{A,B\})}{S[lstv_S \cup \{A,B\}]([\ldots]_\infty) \text{ knows } A}$$

The preconditions state that either $A \in lstv_S \cup \{A,B\}$ (A is already in the knowledge base of s^1) or s^1 is equivalent to some other agent s^2 which in turn knows A ($s^1 \equiv s^2 \wedge s^2 \text{knows} A$). The first precondition does not hold because $lstv_S \cup \{A,B\}$ contains $\{A,B\}$, and not A (and B). Therefore, the **Extract** rule is used to obtain A (and B) from $\{A,B\}$.

$$\textbf{Extract}: \frac{S[lstv_S \cup \{A,B\}]([\ldots]_\infty) \text{ knows } \{A,B\},\ S[lstv_S \cup \{A,B\}]([\ldots]_\infty) \text{ knows } nil,\ nil \overset{K}{\sim} nil}{S[lstv_S \cup \{A,B\}]([\ldots]_\infty) \equiv S[lstv_S \cup \{A,B\} \cup A \cup B]([\ldots]_\infty)}$$

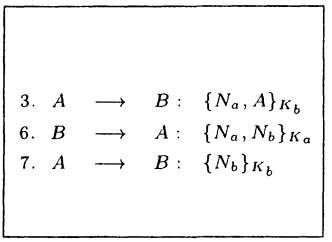

Figure 7. Simplified Protocol

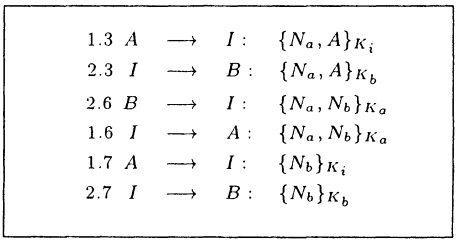

Figure 8. Middle person attack.

Three preconditions must hold for **Extract**: the equivalent agent s^2 (i) must know the message, (ii) must know a decryption key and (iii) this decryption key must match the encryption key for the message. Obviously, s^2 knows the (unencrypted) message $\{A, B\}$. Furthermore, every agent, including s^2, knows the matching key (*nil* in Definition 2.2) needed to unlock unencrypted messages.

Extract creates the equivalent agent s^2 whose knowledge base ($lstv'_S = lstv_S \cup \{A, B\} \cup A \cup B$) contains $\{A, B\}$ as well as A (and B), i.e., $s^2 = S[lstv_S \cup \{A, B\} \cup A \cup B]([\ldots]_\infty)$.

Since "s^2 knows A (and B)" and $s^2 \equiv s^1$, it follows that "s^1 knows A (and B)."

6. Analysis of the Needham-Schroeder Protocol

This section analyzes the well-known "middle person attack" on the Needham-Schroeder Protocol [4,5,8]. The vulnerability in the original protocol is expressed in terms of a message secrecy property. Next, the corrected protocol is formally specified and verified.

The original Needham-Schroeder Protocol can be viewed as two separate, interleaved protocols [8]. The first is used to obtain the other principal's public key from the server (Steps 1 and 2 for K_b and Steps 4 and 5 for K_a). The second, comprising Steps 3, 6 and 7, handles the actual communication between A and B and is vulnerable to the middle person attack. To analyze the vulnerability and the correction, we concentrate on the second interleaved protocol (Figure 7).

6.1 Middle Person Attack

The middle person attack involves an intruder (I) who participates in two interleaved runs of the protocol, the first involving communication between I and A and the second between I and B [8]. Figure 8 labels the steps in the first (resp. second) interleaved protocol as 1.3, 1.6 and 1.7 (resp. 2.3, 2.6 and 2.7).

The middle person attack occurs when A initiates a communication with I. Intruder I then initiates a communication with B with the goal

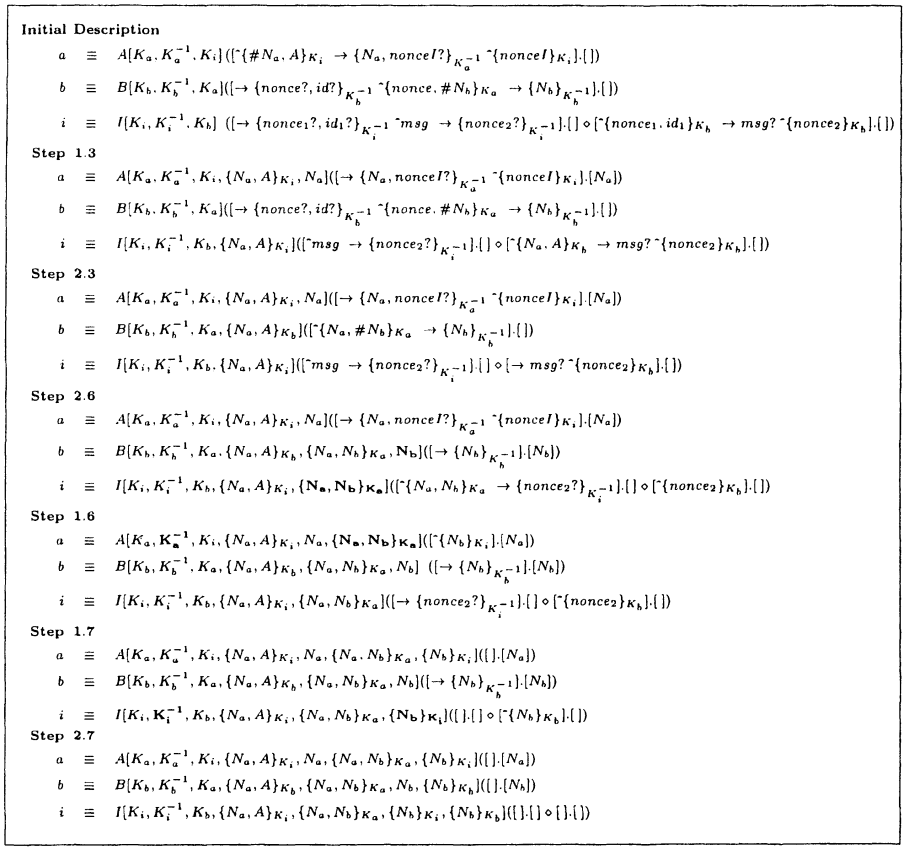

Figure 9. Evolution of agents during attack (original protocol).

of impersonating A. In Step 1.6, I forwards the message $\{N_a, N_b\}_{K_a}$ received from B to principal A. At this point, I does not have access to the message components because it does not know A's private key (K_a^{-1}). However, in Step 1.7, A is unknowingly used as an oracle by I. It decrypts the message $\{N_a, N_b\}_{K_a}$ to obtain the components. Then, it encrypts N_b under I's public key (K_i) and sends $\{N_b\}_{K_i}$ to I. In Step 2.7, I is able to impersonate A in the communication with B because it knows the formerly secret nonce N_b.

Successful authentication in the Needham-Schroeder Protocol relies on the secrecy of the nonce N_b. The middle person attack occurs because N_b is known by I.

Definition 5.1: The Needham-Schroeder Protocol is *vulnerable* to the middle person attack if nonce N_b is known to the intruder i, i.e., i knows N_b is true.

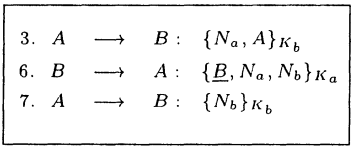

Figure 10. Revised protocol

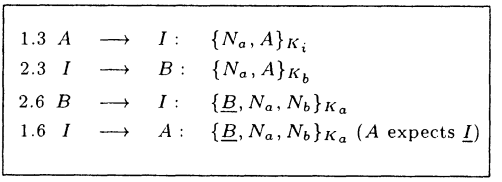

Figure 11. Attempted attack.

Starting with the initial definitions of agents a, b and i in Figure 9, we prove that the Needham-Schroeder Protocol is vulnerable. Figure 9 shows the evolution of agents as the protocol progresses. The inference rules **In**, **Out**, **Comm**, **Chain** and **ChainBase** are applied, as illustrated in Section 5, to reduce the agents. Note that Figure 9 gives the agent descriptions after each communication step.

Examination of Figure 9 reveals that the nonce N_b (in boldface) is created by agent b in Step 2.6 and is stored in its *lstv*. After Step 2.6, agent i knows $\{N_a, N_b\}_{K_a}$ (boldface), but it does not know N_b because it does not know K_a^{-1}. These results are proved, as in Section 5, using the **Knows** and **Extract** rules.

After executing Step 1.6, the *lstv* of agent a contains $\{N_a, N_b\}_{K_a}$ (boldface). Since a knows K_a^{-1} (boldface), it also knows N_b; this is proved using the **Knows** and **Extract** rules.

However, after Step 1.7, agent i has $\{N_b\}_{K_i}$ (boldface) in its *lstv*. Since agent i already knows K_i^{-1} (boldface), on applying the **Knows** and **Extract** rules, we can show that it also knows N_b. Thus, "i knows N_b" is true, and the Needham-Schroeder Protocol is vulnerable to the middle person attack according to Definition 5.1. In Step 2.7, agent i exploits the vulnerability by sending $\{N_b\}_{K_b}$ to agent b, thereby impersonating a.

6.2 Revised Protocol

To prevent the middle person attack, nonce N_b must not be captured by intruder I. Therefore, the revised protocol [4,8,9] requires the identity of the contacted principal (B) to be included in the message sent in Step 6.

Note that the initiating principal A expects the identity of the principal to be included in the message it receives. If the identity that is received differs from what is expected, A rejects the message. Figure 10 shows the revised protocol. Note that the new information added to the protocol is underlined in Step 6.

Figure 11 shows the attempted attack by intruder I. Steps 1.3 and 2.3 proceed as in the original protocol. However, in Step 2.6, B sends its identity B embedded in the encrypted message $\{\underline{B}, N_a, N_b\}_{K_a}$ to I. Since intruder I does not know K_a^{-1}, it cannot unlock the message and insert its own identity $\underline{I}$ in place of $\underline{B}$. All it can do is replay $\{\underline{B}, N_a, N_b\}_{K_a}$ to

A. Since A expects $\{\underline{I}, N_a, N_i\}_{K_a}$, it rejects the message sent by I and the attack fails.

7. Conclusions

The integrated approach described in this paper supports formal, yet natural translations of cryptographic protocols, eliminating the need for protocol idealization as in BAN logic. It also improves on process calculus methodologies (e.g., Spi calculus) by permitting the exhaustive modeling of messages and principals, thereby supporting comprehensive protocol analysis and verification.

Novel features of the approach include an expressive message passing semantics, sophisticated concurrency constructs and seamless integration of inference rules for agent communication, reduction and information analysis. The approach facilitates the analysis of cryptographic protocols in open, potentially hostile environments by making no assumptions about the honesty of principals.

8. References

[1] Abadi, M. and Cardelli, L. (1995) An imperative object calculus. *Proceedings of the Conference on Theory and Practice of Software*, 471-485.

[2] Abadi, M. and Gordon, A. (1997) A calculus for cryptographic protocols: The Spi calculus. *Proceedings of the Fourth ACM Conference on Computer and Communications Security.*, 36-47.

[3] Abadi, M. and Gordon, A. (1997) Reasoning about cryptographic protocols in the Spi calculus. *CONCUR'97: Concurrency Theory, Springer LNCS, Vol. 1243*, Amsterdam, The Netherlands, 59-73.

[4] Anderson, R. and Needham, R. (1995) Programming Satan's computer. *Computer Science Today*, 426-440.

[5] Burrows, M., Abadi, M. and Needham, R. (1989) A logic of authentication. *ACM Operating Systems Review; Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, **23(5)**, 34-37.

[6] Burrows, M., Abadi, M. and Needham, R. (1990) A logic of authentication. *ACM Transactions on Computer Systems*, **8(1)**, 18-36.

[7] Gong, L., Needham, R. and Yahalom, R. (1990) Reasoning about belief in cryptographic protocols. *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 234-248.

[8] Lowe, G. (1995) An attack on the Needham-Schroeder public key authentication protocol. *Information Processing Letters*, **56(3)**, 131-133.

[9] Lowe, G. (1996) Some new attacks upon security protocols. *Proceedings of the Ninth IEEE Computer Security Foundations Workshop.*

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

[11] Meadows, C. (1999) Analysis of the Internet key exchange protocol using the NRL Protocol Analyzer. *Proceedings of the IEEE Symposium on Security and Privacy.*

[12] Meadows, C. (2000) Open issues in formal methods for cryptographic protocol analysis. *Proceedings of the DISCEX 2000 Conference*, 237-250.

[13] Milner, R. (1989) *Communication and Concurrency*. Prentice-Hall, New York.

[14] Neuman, C. and Ts'o, T. (1994) Kerberos: An authentication service for computer networks, *IEEE Communications*, **32(9)**, 33-38.

[15] Schneier, B. (1996) *Applied Cryptography*. John Wiley, New York.

[16] Syverson, P. and van Oorschot, P. (1994) On unifying some cryptographic protocol logics, *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 165-177.