CHAPTER 8
SIMULATION AND ANALYSIS OF CRYPTOGRAPHIC PROTOCOLS

M. Papa, O. Bremer, S. Magill, J. Hale and S. Shenoi

Abstract This paper integrates logic and process calculus components to permit the comprehensive simulation and analysis of cryptographic protocols. The approach permits proofs about the information transmitted in protocols as well as the behavior of participating principals.

Keywords: Cryptographic protocols, simulation, verification, logics, process calculi

1. INTRODUCTION

Cryptographic protocols are unambiguous, mutually subscribed series of steps, involving the exchange of messages between communicating agents (principals). They are indispensable to distributed computing applications ranging from electronic commerce to national defense [14,15].

Most protocols fail due to design flaws, not because of weaknesses in encryption [3,8,9,14]. A subtle flaw in the Needham-Schroeder protocol [8] went undiscovered for nine years, potentially allowing malicious entities to steal keys, corrupt sensitive data, even sabotage military systems.

Higher order logics [3,5,15] are used to reason about protocols. BAN logic [3] is a many-sorted modal logic for analyzing authentication protocols. It assumes that authentication is a function of integrity and freshness of messages. Inference rules are specified for tracing these attributes through all the steps of a protocol. BAN also defines inference rules for reasoning about the information held by principals ("beliefs"). BAN has detected flaws and exposed redundancies in several protocols, including Needham-Schroeder, Yahalom and Kerberos protocols [3,14].

Despite their success, BAN logic and its derivatives (e.g., GNY [5] and SVO logics [15]) have two major limitations. First, protocols must be idealized prior to analysis. The translation of a real protocol to its idealized counterpart is a difficult process, susceptible to misinterpretation and errors [5,14]. Moreover, a single protocol may have multiple idealized representations, each with slightly different properties. The second limitation arises from the logics' inability to express the actions and evolution of principals. Message passing is modeled as an atomic event; the exchange of a message only affects the information held by the principals. When the actions and evolution of principals are ignored, it is not possible to make formal guarantees about their behavior.

Another promising approach to protocol verification involves modeling principals as agents using a process calculus [1,2,10-12]. An axiomatization of the process calculus is then used to obtain proofs about agent behavior. The **π-calculus** [11,12] exemplifies this approach. It models distributed systems as primitive concurrent agents with complex message passing. Computation is simulated by agent communication: agents exchange messages and evolve to simpler forms, until all communication ceases. A deep embedding of **π-calculus** using Higher Order Logic (HOL) [6] permits proofs about agents, distributed systems, and the **π-calculus** itself [10]. The ROC process calculus [7] extends the **π-calculus** for distributed objects with complex message passing. The *Spi*-calculus [2] augments the **π-calculus** with cryptographic primitives, permitting encrypted messages.

Process calculi give the ability to comprehensively model and reason about the behavior of principals (agents) without protocol idealization. However, process calculi lack constructs for reasoning about messages. While the *Spi*-calculus can model principals exchanging encrypted messages, it can neither model nor reason about the information held by principals. Thus, only limited properties can be proven about protocols.

This paper integrates logic and process calculus components, combining their strengths to permit the comprehensive simulation and analysis of cryptographic protocols. Protocols need not be idealized prior to modeling and verification. Moreover, it is possible to prove properties about individual messages, principals, and the protocol itself.

2. MESSAGE MODELING

Following the style of process calculi [7,11,12], we model message passing using synchronous communication. Asynchronous communication protocols can be constructed with synchronously communicating agents.

Communication occurs when a message output by one agent matches a pattern exposed by another. This section describes the syntax of messages and patterns, and the pattern-matching conventions.

Definition 2.1: A *key* ($k \in key$) is a public/private key (K_n/K_n^{-1}), a shared or secret key (K_n^s), the concatenation of two keys ($k_1 : k_2$), a placeholder for a key that is not yet known ($k?$), or *nokey*, corresponding to a cleartext message:

$$k \ ::= \ K_n \mid K_n^{-1} \mid K_n^s \mid k_1 : k_2 \mid k? \mid nokey$$

We assume the existence of an infinite set of *names* ($n \in name$) as a basic type. It is used to create unique keys, and data items for messages and patterns.

Definition 2.2: A *message* ($m \in message$) is a tuple $\{v_1, v_2, ..., v_j\}_k$ encrypted under key k. A *value* ($v \in value$) is a key (k), a message (m), a name (n) or a fresh name ($\#n$):

$$\begin{aligned} m \ &::= \ \{v_1, v_2, ..., v_j\}_k \\ v \ &::= \ k \mid m \mid n \mid \#n \end{aligned}$$

To permit complex structures, messages are defined as nested tuples of "values" ($v \in value$). Note that fresh names (nonces and time stamps) are also incorporated in the BAN and GNY logics [3,5]. A newly generated value is considered to be fresh throughout a protocol run. A list of fresh names associated with each run must be maintained to permit reasoning about freshness.

Patterns permit the capture of messages and their contents. A pattern exposed by a receiver expresses the information it has about message format and content.

Definition 2.3: A *pattern* ($p \in pattern$) is a tuple $\{p_1, p_2, ..., p_j\}_k$ encrypted under key k, a key (k), a wildcard ($n?$) for capturing values, or a datum (n):

$$p \ ::= \ \{p_1, p_2, ..., p_j\}_k \mid k \mid n? \mid n$$

Figure 1 illustrates the modeling of messages and patterns. In the example, the Law Enforcement Agency Field (LEAF) for a Clipper transmission [14,15] is output by *Phone* as $\{\{K_s^s\}_{K_{c_1}^s : K_{c_2}^s}, ID, AUTH\}_{K_f^s}$, where $\{K_s^s\}_{K_{c_1}^s : K_{c_2}^s}$ is the session key K_s^s encrypted under the Clipper chip's unit key $K_{c_1}^s : K_{c_2}^s$, ID is the Clipper serial number, and AUTH is the escrow authenticator. All these items are encrypted under the Clipper family key K_f^s.

Since the law enforcement agency knows the LEAF format and the family key K_f^s, it can expose the pattern $\{ekey?, id?, auth?\}_{K_f^s}$ to receive

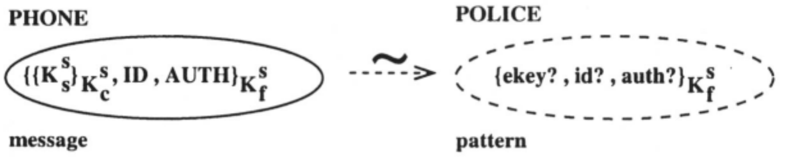
PHONE
$\{\{K^s_s\}_{K^s_c}, ID, AUTH\}_{K^s_f}$
message
POLICE
$\{ekey?, id?, auth?\}_{K^s_f}$
pattern

Figure 1. Modeling message passing.

the message as input. The wildcards *ekey*?, *id*? and *auth*? are placeholders for receiving the individual LEAF items. Communication occurs when the *message* output by *Phone* in Figure 1 matches the *pattern* exposed by *Police.*

The following rules define the matching of messages and patterns. First, we define the matching of keys. Intuitively, the key matching operator establishes that an agent can decrypt an encrypted message if it possesses the right key.

Definition 2.4: Key matching ($\overset{K}{\sim}$) is defined by the following rules:

$$\begin{array}{rcl} K_a & \overset{K}{\sim} & K_b^{-1} \text{ iff } a = b \\ K_a^s & \overset{K}{\sim} & K_b^s \text{ iff } a = b \\ nokey & \overset{K}{\sim} & nokey \end{array}$$

Definition 2.5: The matching of messages and patterns ($\sim$) is defined by the following rules ($v \in value$, $p \in pattern$, $k \in key$, $m \in message$ and $n \in name$):

$$\begin{array}{rcl} \{v_1, v_2, ..., v_j\}_{k_1} & \sim & \{p_1, p_2, ..., p_j\}_{k_2} \text{ iff } k_1 \overset{K}{\sim} k_2 \wedge v_i \sim p_i \ \forall\, i = 1..j \\ m & \sim & n? \\ v & \sim & n? \\ v & \sim & n \text{ iff } v = n \\ k & \sim & k? \\ k_1 & \sim & k_2 \text{ iff } k_1 = k_2 \end{array}$$

Communication occurs only when a message is matched by an exposed pattern. Free occurrences of a wildcard n? in the pattern p are replaced with the corresponding matching value v throughout the pattern. For example, in Figure 1, the message $\{\{K^s_s\}_{K^s_c}, ID, AUTH\}_{K^s_f}$ is matched by the pattern $\{ekey?, id?, auth?\}_{K^s_f}$ exposed by *Police*, causing $\{K^s_s\}_{K^s_c}$, *ID* and *AUTH* to be bound to *ekey*, *id* and *auth*, respectively.

3. PROTOCOL MODELING

Protocols are modeled as sequences of messages exchanged by agents. We model the messages exchanged and the behavior of the agents.

The starting point is a sequence of messages or patterns that are output ($\hat{}$) or input ($\rightarrow$) by an agent. Note that the term $a \equiv m\hat{}seq$ denotes that agent a outputs the message m. Likewise, $a \equiv m\hat{}p \rightarrow seq$ denotes that a first outputs the message m and then exposes the pattern p for input.

Definition 3.1: Let $m \in message$, $p \in pattern$ and *Istn* : List of $n \in name$. Then, *sequences (seq)*, *annotated sequences* (*aseq*) and *concurrent sequences* (*cseq*) of messages and/or patterns are defined by:

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

where *nil* is the empty list, $[seq]_\infty$ is an infinite sequence, and $\diamond$ is the commutative concurrency operator for sequences.

A sequence (*seq*) is defined as a sequence with an output message ($m\hat{}seq$) or an input pattern ($p \rightarrow seq$) or empty (*nil*). An annotated sequence (*aseq*) is a sequence of messages and/or patterns followed by a list of names ([*seg*].[*lstn*]); the list [*lstn*] stores fresh names for the corresponding sequence ([*seg*]). The term $[seq]_\infty$ for an annotated sequence denotes a sequence that has to be executed repeatedly, e.g., to model a server. A concurrent sequence (*cseq*), the concurrent composition ($\diamond$) of an annotated sequence and a concurrent sequence, models threads of execution for a single agent. It can express multiple runs of one protocol or parallel runs of different protocols.

Definition 3.2: The following property holds for an infinite sequence $[seq]_\infty$:

$$[seq]_\infty \equiv [seq].[nil] \diamond [seq]_\infty$$

Definition 3.3: An agent a is defined by:

$$a ::= c(id, lstv)$$

where $c \in cseq$, $id \in name$ and *lstv* : List of $v \in value$.

The term *id* identifies the agent *a*, *lstv* is a list of values representing the information possessed by the agent. As in the BAN and GNY logics

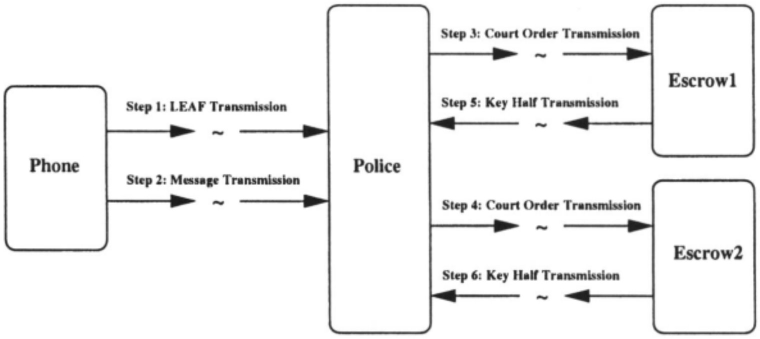
Phone
Police
Escrow1
Escrow2
Step 1: LEAF Transmission
Step 2: Message Transmission
Step 3: Court Order Transmission
Step 4: Court Order Transmission
Step 5: Key Half Transmission
Step 6: Key Half Transmission

Figure 2. Simplified Clipper protocol.

[3,5], maintaining the list *lstv* permits reasoning about agents, especially about what they know, say and have been told.

The modeling of cryptographic protocols with this syntax is highlighted using the simplified Clipper protocol in Figure 2.

Four agents participate in the protocol: a Clipper telephone (*Phone*), a law enforcement agent (*Police*) and two key escrow agents (*Escrow*1 and *Escrow*2). The protocol has six steps. *Police* exposes patterns to capture the LEAF (Step 1) and the encrypted message (Step 2) from *Phone*. Next, *Police* transmits *Phone*'s identification and a court order to *Escrow*1 and *Escrow*2 (Steps 3 and 4), each of which hold one-half of *Phone's* unit key. In Steps 5 and 6, *Escrow*1 and *Escrow*2 transmit one-half of *Phone*'s unit key to *Police*.

Figure 3 shows the formal model of the protocol. *Phone* is defined as: *cseq*(*PhoneId*, *lstv*). *cseq* is composed of two output messages, the LEAF $(\{\{K_s^s\}_{K_{c_1}^s:K_{c_2}^s}, ID, AUTH\}_{K_f^s})$ and a message *M* encrypted under session key K_s^s $(\{M\}_{K_s^s})$. No fresh names are used in this particular protocol, thus *lstn* = [*nil*]. List *lstv* = $[K_s^s, K_{c_1}^s : K_{c_2}^s, K_f^s]$ contains all the the keys required for *Phone* to implement Clipper transmissions. Note that the concatenation of the key halves $K_{c_1}^s : K_{c_2}^s$ yields K_c^s, *Phone's* unit key.

Police is defined by *cseq*(*PoliceId*, *lstv*). Its *lstv* only contains the Clipper family key (K_f^s) and two keys: $K_{e_1}^s$ and $K_{e_2}^s$ for communicating with *Escrow*1 and *Escrow*2, respectively. *Police*'s *cseq* contains two patterns: $\{ekey?, id?, auth?\}_{K_f^s}$ to capture the LEAF and *emsg*? to capture *Phone*'s encrypted message $(\{M\}_{K_s^s})$. The *cseq* also contains two more concurrent sequences, each with a message output followed by a pattern exposure. The first $\{id, auth, CO\}_{K_{e_1}^s}\hat{}\{id, kc_1?\}_{K_{e_1}^s} \rightarrow$ is directed at *Escrow*1 to send *Phone's ID* and *AUTH* (which were previously bound to the wildcards *id*? and *auth*?) and the court order (*CO*)

$$\begin{array}{rcl}
Phone & \equiv & [\{\{K^s_s\}_{K^s_{c_1}:K^s_{c_2}}, ID, AUTH\}_{K^s_f}\hat{}\{M\}_{K^s_s}\hat{}].[nil](PhoneId,[K^s_s, K^s_{c_1} : K^s_{c_2}, K^s_f]) \\
 & & \\
Police & \equiv & [\{ekey?, id?, auth?\}_{K^s_f} \rightarrow emsg? \rightarrow].[nil] \diamond \\
 & & [\{id, auth, CO\}_{K^s_{e_1}}\hat{}\{id, kc_1?\}_{K^s_{e_1}} \rightarrow].[nil] \diamond \\
 & & [\{id, auth, CO\}_{K^s_{e_2}}\hat{}\{id, kc_2?\}_{K^s_{e_2}} \rightarrow].[nil](PoliceId,[K^s_f, K^s_{e_1}, K^s_{e_2}]) \\
 & & \\
Escrow1 & \equiv & [\{id?, auth?, CO\}_{K^s_{e_1}} \rightarrow \{id, K^s_{c_1}\}_{K^s_{e_1}}\hat{}].[nil](E1Id,[K^s_{e_1}]) \\
 & & \\
Escrow2 & \equiv & [\{id?, auth?, CO\}_{K^s_{e_2}} \rightarrow \{id, K^s_{c_2}\}_{K^s_{e_2}}\hat{}].[nil](E2Id,[K^s_{e_2}])
\end{array}$$

Figure 3. Formal model of Clipper protocol.

and to receive the first half of *Phone*'s unit key (stored in the wildcard $kc_1?$). Similarly, the second sequence is directed at *Escrow*2 to receive the second half of *Phone*'s unit key (stored in $kc_2?$) Since no fresh names are used, *lstn* = [*nil*] for the current sequence.

The definitions of *Escrow*1 and *Escrow*2 are similar to each other. *Escrow*1 is defined by *cseq*(*E*1*Id*, $[K^s_{e_1}]$) where $K^s_{e_1}$ is used to communicate with *Police*. *Escrow*1's *cseq* contains $\{id?, auth?, CO\}_{K^s_{e_1}}$ to receive information from *Police* and the message $\{id, K^s_{c_1}\}_{K^s_{e_1}}$ to send one-half of Phone's unit key to *Police*. The pattern $\{id?, auth?, CO\}_{K^s_{e_1}}$ exposed by *Escrow*1 ensures that *Police* submits a verifiable (id, escrow authenticator, court order) tuple corresponding to *Phone*.

4. PROTOCOL SIMULATION

Most techniques for formally modeling protocols only consider the messages exchanged by principals. We adopt an integrated approach that models messages and principal behavior. Principals are formally modeled as concurrent agents that exchange complex messages and reduce to simpler forms. A virtual machine has been designed to simulate protocols by applying inference rules defined for agent communication and reduction [7]. This permits the comprehensive simulation of protocols – essential to their analysis and verification

4.1. INFERENCE RULES FOR AGENT BEHAVIOR

Inference rules governing agent communication and reduction provide an unambiguous semantics for simulating agent behavior. Actions defined for agents include communication, reduction and binding.

Definition 4.1: Agent communication and reduction $(\Longrightarrow)$ are defined by the following inference rules. Communication offers are denoted by $\xrightarrow{\alpha}$, where α is m (input) or $\bar{m}$ (output):

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

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

$$\textbf{Comm}: \frac{a \xrightarrow{\bar{m}} a',\ b \xrightarrow{m} b'}{a \ \&\ b \Longrightarrow a' \ \&\ b'}$$

The **In** rule defines the semantics for the receipt of a message using a pattern. The precondition $m \sim p$ specifies that the message m must match the pattern p exposed by agent $(id, lstv)$ with communication sequence $[p \rightarrow seg].[lstn] \diamond cseq(id, lstv)$. The postcondition states that after the message is accepted, the agent reduces to $[seq].[lstn] \diamond cseq\{m/p\}(id, lstv \cup m)$. All free ocurrences of wildcards in p are replaced by the corresponding values in m in the remaining sequences of messages and/or patterns; this is specified by $[seg].[lstn] \diamond cseq\{m/p\}$. The agent's *lstv* is updated with the message that has just been accepted. The new list of values is $lstv \cup m$ ($\cup$ denotes concatenation).

The **Out** rule has no preconditions. On offering message m, the communication sequence $[m\hat{\ }seq].[lstn] \diamond cseq$ of agent (id, *lstv*) reduces to $[seq].[lstn \cup fresh(m)] \diamond cseq$. The list of names *lstn* associated with the remaining sequence of messages and/or patterns *seq* is updated with the set of fresh names in m ($fresh(m)$); this is expressed by $lstn \cup fresh(m)$. Similarly, the list of values *lstv* held by the agent is updated with the offered message to produce $lstv \cup m$.

Comm governs agent reduction. Two preconditions must hold for a & b, the concurrent composition of agents a and b, to reduce to a' & b' after communicating message m. First, agent a must be able to reduce to agent a' with output $\bar{m}$. Second, agent 6 must be able to reduce to agent b' with input m.

4.2. CLIPPER PROTOCOL SIMULATION

We illustrate the simulation of the Clipper protocol in Figure 2 by presenting the agent reductions for the first step. The agent definitions in Figure 3 serve as the starting point for the simulation.

Step 1 implements the LEAF Transmission. *Phone* outputs the LEAF as: $\{\{K_s^s\}_{K_{c_1}^s : K_{c_2}^s}, ID, AUTH\}_{K_f^s}$. *Police* exposes $\{ekey?, id?, auth?\}_{K_f^s}$.

Note that *Police* knows the Clipper family key K_f^s used to encrypt the LEAF, i.e., $K_f^s \in lstv$ of *Police*. The message and pattern match according to Definitions 2.4 and 2.5. Using **Comm** and chaining to **Out** for *Phone* and **In** for *Police*, the agents reduce to:

$$\begin{aligned}
Phone &\equiv [\{M\}_{K_s^s}\hat{}\,].[nil](PhoneId, [K_s^s, K_{c_1}^s : K_{c_2}^s, K_f^s, \{\{K_s^s\}_{K_{c_1}^s : K_{c_2}^s}, ID, AUTH\}_{K_f^s}]) \\
Police &\equiv [emsg? \rightarrow].[nil] \diamond [\{ID, AUTH, CO\}_{K_{e_1}^s}\hat{}\{ID, kc_1?\}_{K_{e_1}^s} \rightarrow].[nil] \diamond \\
&\quad [\{ID, AUTH, CO\}_{K_{e_2}^s}\hat{}\{ID, kc_2?\}_{K_{e_2}^s} \rightarrow].[nil] \\
&\quad (PoliceId, [K_f^s, K_{e_1}^s, K_{e_2}^s, \{\{K_s^s\}_{K_{c_1}^s : K_{c_2}^s}, ID, AUTH\}_{K_f^s}])
\end{aligned}$$

Note that $\{K_s^s\}_{K_{c_1}^s : K_{c_2}^s}$, *ID* and *AUTH* are bound to *Police*'s wildcards *ekey*?, *id*? and *auth*?, respectively. Furthermore, the lists *lstv* of *Phone* and *Police* are updated to include $\{\{K_s^s\}_{K_{c_1}^s : K_{c_2}^s}, ID, AUTH\}_{K_f^s}$.

The remaining steps proceed similarly. Two communicating agents always reduce according to the **Comm** rule which requires **Out** to be applied to the sender and **In** to be applied to the receiver. When no further communication is possible the protocol run is complete.

5. PROTOCOL ANALYSIS

The agent inference rules in Section 4 formalize agent behavior, i.e., how agents exchange messages and evolve. To analyze protocols, it is also necessary to specify how agents can infer new information from exchanged messages. Inference rules are required for reasoning about the information held by agents, the freshness of the information, and whether or not agents can communicate. This section specifies inference rules for messages and illustrates their application in the Clipper example.

5.1. INFERENCE RULES FOR AGENT KNOWLEDGE

The rules in Definition 5.1 are used to determine: (i) what an agent knows, (ii) what it can infer, and (iii) what messages it can produce.

Definition 5.1: The inference rules **Knows, Extract** and **Construct** are defined by:

$$\textbf{Knows} : \frac{v \in lstv}{Name(cseq(id, lstv)) \text{ knows } v}$$

$$\textbf{Extract} : \frac{\{v_1, \ldots, v_j\}_{k1} \in lstv,\ \ k_2 \in lstv,\ \ k_1 \overset{K}{\sim} k_2}{cseq(id, lstv) \equiv cseq(id, lstv \cup v_1 \ldots \cup v_j)}$$

$$\textbf{Construct}: \frac{v_i(i=1..j) \in lstv,\ \ k \in lstv}{cseq(id, lstv) \equiv cseq(id, lstv \cup \{v_1, ..., v_j\}_k)}$$

The **Knows** rule expresses predicates of the form "agent *id* knows value *v*;" this is written as "*id* knows *v*." This predicate is true only if $v \in lstv$ or if *v* can be derived using the **Extract** and **Construct** rules. The function *Name*(*A*) returns the *id* of an agent *A* from the agent description, i.e., *Name*(*cseq*(*id,lstv*)) = *id.*

The **Extract** rule helps extract the components of a message held by an agent in its *lstv*. The list *lstv* is augmented with the extracted components according to the rule conclusion. For example, if an agent holds the encrypted message $\{X\}_{K^s_w}$ and the key K^s_w in *lstv*, then the agent can obtain *X* and, therefore, *X* is added to *lstv*. Note that $\cup$ denotes the concatenation of a value list and a value.

Construct creates new values from values held in *lstv*; these new values are added to *lstv*. For example, if an agent's *lstv* holds the values *X, Y* and Z, and the key K^s_w, then it can construct the encrypted message $\{X, Y, Z\}_{K^s_w}$; this encrypted message is added to the agent's *lstv*. Altering *lstv* using **Extract** and **Construct** does not change agent behavior, only what the agent knows.

Variations of **Extract**, **Construct** and **Knows** are used in the GNY logic [5] as the "Being-Told" and "Possession" rules.

5.2. CLIPPER PROTOCOL ANALYSIS

The Clipper protocol in Figures 2 and 3 is used to illustrate the application of the message inference rules. We prove that at the end of the protocol, the *Police* agent "knows" the message *M* that is encrypted as $\{M\}_{K^s_s}$ and output by *Phone* in Step 2. The configuration of the *Police* agent after the protocol is completed serves as the starting point for the analysis. Since the *Police*'s knowledge (*lstv*) is of special importance, the analysis focuses on it.

$$[\underbrace{K^s_f}_{(4)}, \underbrace{K^s_{e_1}}_{(7)}, \underbrace{K^s_{e_2}}_{(8)}, \underbrace{\{\underbrace{\{K^s_s\}_{K^s_{c_1}:K^s_{c_2}}}_{(2)}, ID, AUTH\}_{K^s_f}}_{(3)}, \underbrace{\{M\}_{K^s_s}}_{(1)}, \{ID, AUTH, CO\}_{K^s_{e_1}},$$
$$\{ID, AUTH, CO\}_{K^s_{e_2}}, \underbrace{\{ID, K^s_{c_1}\}_{K^s_{e_1}}}_{(5)}, \underbrace{\{ID, K^s_{c_2}\}_{K^s_{e_2}}}_{(6)}]$$

To **Extract** *M* from (1), *Police* must know K^s_s which can be **Extracted** from (2). To do so, *Police* must **Extract** (2) from (3) requiring knowledge of K^s_f (4), $K^s_{c_1}$ and $K^s_{c_2}$. The latter two must be **Extract**ed from (5) and (6), requiring *Police* to know $K^s_{e_1}$ (7) and $K^s_{e_2}$ (8).

Since *Police* **Knows** (4), (7) and (8) $(K_f^s, K_{e_1}^s, K_{e_2}^s \in lstv_{Police})$, the **Extract**-rule can be applied several times and finally $M \in lstv_{Police}$. This completes the proof of "*PoliceId* knows *M*."

6. AUTOMATING PROOFS

The inference rules defined in the previous sections provide the foundation for verification. A significant advantage is that the rules governing agent communication and reduction (used for simulation) can be integrated with the inference rules for messages (used for analysis) to reason about the properties of protocols and participating agents.

Proofs are automated using a translation (mechanization) [4] of the agent reduction and message inference rules into Higher Order Logic (HOL) [6], an interactive theorem proving environment. A HOL session results in a *theory* – an object containing sets of types, constants, definitions, axioms and theorems (logical consequences of the definitions and axioms). As new theorems are proved, they are added to the theory and may be used to develop additional results. Only well-formed theories can be constructed because formal proofs are required for all new theorems.

Automating proofs in HOL involves incorporating type definitions, function definitions, inference rules, tactics, and conversions.

Four types are defined in the HOL Protocol Theory: *key*, *value*, *message* and *pattern*, the last three being mutually recursive.

Then, functions are implemented to manipulate the specified types. E.g., functions for matching keys and matching messages and patterns, and the *Name* function for obtaining an agent's *id*.

Next, inference rules are defined for reasoning about agent behavior and knowledge (**In**, **Out**, **Comm**, **Knows**, etc.). This implements a HOL theory supporting basic proofs via manual rule application.

Tactics and conversions are required to automate proofs. A tactic tranforms a goal into an equivalent goal that is easier to prove. Tactics are introduced to selectively apply the inference rules governing agent behavior and knowledge to facilitate theorem proving.

The final step is to create conversions for transforming propositions into theorems. A conversion must establish the truth value for every expression of the form it is designed to handle. Although it is not feasible to develop conversions for all types of propositions, their introduction, even to a limited degree, can facilitate the automation of proofs.

7. CONCLUSIONS

The integration of logic and process calculus provides a powerful framework for simulating and analyzing cryptographic protocols. The approach advances logic techniques by not requiring protocols to be idealized before analysis. It improves on process calculi by permitting the exhaustive modeling of messages and principals. Novel features include an expressive message passing semantics, sophisticated modeling of concurrency, and seamless integration of inference rules for agent behavior and knowledge. Furthermore, no assumptions are made about the honesty of communicating agents. This facilitates the analysis of cryptographic protocols in open, potentially hostile environments.

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 D. (1997) Reasoning about cryptographic protocols in the Spi calculus. *Proceedings of the Fourth ACM Conference on Computer and Communications Security*,, 36-47.

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

[4] Galiasso, P. (1998) *Mechanization of ROC in Higher Order Logic.* M.S. Thesis, Computer Science Department, University of Tulsa, Tulsa, Oklahoma.

[5] 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.

[6] Gordon, M. and Melham, T. (1993) *Introduction to Higher Order Logic (HOL).* Cambridge University Press, Cambridge, U.K.

[7] Hale, J., Threet, J. and Shenoi, S. (1997) A framework for high assurance security of distributed objects, in *Database* Security, *X: Status and Prospects* (eds. P. Samarati and R. Sandhu), Chapman and Hall, London, 101-119.

[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] Melham, T. (1992) A mechanized theory of the π-calculus in HOL. Technical Report 244, University of Cambridge Computer Laboratory, Cambridge, U.K.

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

[12] Milner, R., Farrow, J. and Walker, D. (1989) A calculus of mobile processes. Report ECS-LFCS-89-85&86, University of Edinburgh, Edinburgh, U.K.

[13] Pfleeger, C. (1997) *Security in Computing.* Prentice Hall, Upper Saddle River, New Jersey.

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

[15] 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