A Semantic Model for Authentication Protocols

Thomas Y.C. Woo* Simon S. Lam†
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

Abstract

We specify authentication protocols as formal objects with precise syntax and semantics, and define a semantic model that characterizes protocol executions. We have identified two basic types of correctness properties, namely, *correspondence* and *secrecy*, that underlie the correctness concerns of authentication protocols. We define *assertions* for specifying these properties, and a formal semantics for their satisfaction in the semantic model. The Otway-Rees protocol is used to illustrate the semantic model and the basic correctness properties.

1 Introduction

Authentication is a fundamental concern in the design of secure distributed systems [14, 25]. In distributed systems, authentication is typically carried out by protocols, called *authentication protocols*. The primary goal of an authentication protocol is to establish the identities of the parties (referred to as *principals* in the security literature) who participate in the protocol. Many authentication protocols, however, also accomplish a secondary goal, namely, to distribute a new secret session key for further communication among the principals (see discussion in Section 2).

*Research supported by NSA Computer Security University Research Program under contract no. MDA 904-91-C7046.

†Research supported by National Science Foundation grant no. NCR-9004464.

The design of authentication protocols is notoriously error-prone. Many authentication protocols have been published and later found to contain subtle weaknesses or flaws [3, 6, 7, 17, 19, 20]. Two factors contribute to this: (i) the lack of well-established guiding principles for protocol design, and (ii) the use of informal operational reasoning for protocol analysis. Recently, a great deal of research has been directed to remedy these two problems. With respect to (i), basic design principles corresponding to symmetric and asymmetric cryptosystems are discussed in [25], and the design of a family of authentication protocols is systematically demonstrated in [5]. With respect to (ii), many formal approaches have been proposed for the analysis of authentication protocols [4, 6, 8, 10, 11, 12, 17, 23].

There are, however, two shortcomings in existing formal analysis approaches. First, there is a significant gap between the formal and intuitive notions of correctness. The formal notion of correctness is often highly specialized and does not always capture the entire intuitive notion of correctness an authentication protocol designer has in mind. For example, some approaches [8, 12] adopt *secrecy* as their main correctness criterion, and do not address concerns not directly related to secrecy; while others [6, 10, 11] specify correctness in terms of *state of belief* and make implicit secrecy concerns. This gap can lead to potential misinterpretations.

Second, the "formal" notions of correctness adopted in most existing approaches are not sufficiently formalized. Such imprecision can lead to difficulties in protocol analysis. In the extreme

case, it can allow an intuitively flawed authentication protocol to be proved "correct." A good example of this is the BAN logic construct "$P \overset{k}{\leftrightarrow} Q$" whose meaning is only informally explained in [6]. The informal explanation was widely misunderstood and led directly to the criticism by Nessett in [21]. In fact, the precise meaning of "$P \overset{k}{\leftrightarrow} Q$" only became clear with the publication of a new semantics for BAN logic in [2].

Furthermore, with an insufficiently formalized notion of correctness, it is difficult to judge the generality of a proposed analysis approach. In particular, the questions of *soundness* and *completeness* cannot be answered.

The goal of our research is to address these two shortcomings. To precisely define any notion of correctness, we need a semantic model that characterizes authentication protocol executions. We define such a semantic model for authentication protocols by specifying them as formal objects with rigorous syntax and semantics. We propose a formalization of correctness based on two types of correctness properties, namely, *correspondence* and *secrecy*. Correspondence properties address authentication concerns, while secrecy properties address secrecy concerns. We define the syntax of assertions for specifying both kinds of properties, and a formal semantics for their satisfaction in terms of the semantic model.

We maintain a clean separation between the underlying semantic model and the correctness properties being defined. Thus if new correctness properties (other than correspondence and secrecy) are needed for a particular application, they can be easily defined using the semantic model.

The balance of this paper is organized as follows. In Section 2, we introduce informally the correspondence and secrecy properties. In Section 3, we present our semantic model for authentication protocols. In Section 4, we introduce the syntax of correspondence and secrecy assertions and define their semantics in terms of our semantic model. In Section 5, we compare our approach with several other formal analysis approaches. In Section 6, we discuss some design decisions in formulating our approach and discuss work for further research.

2 Correctness Properties

In this section, we take an informal look at various correctness properties for authentication protocols. We guide our discussion by examining several toy authentication protocols as well as a realistic authentication protocol proposed by Otway and Rees [22]. The discussion here would hopefully provide sufficient motivation for our formalization to be presented in Sections 3 and 4.

2.1 Two Basic Properties

Correctness is no more than satisfying a set of specified goals. Thus to understand correctness, we begin by looking at the goals an authentication protocol is supposed to accomplish. As mentioned in the Introduction section, there are two such goals:

(G1) *Authentication* — The primary goal of an authentication protocol is for an authenticating principal to be ascertained of the identity of an authenticated principal.[1] In other words, upon the successful termination of protocol execution, an authenticating principal should be assured that it is "talking" to the principal it has in mind.

(G2) *Key Distribution* — While (G1) assures that the "right" principals are present during an authentication exchange, it provides no guarantee of the authenticity and privacy of any ensuing communication. To achieve this, a secret session key can be distributed during the authentication exchange. This secret session key can then be used to ensure the authenticity and privacy of future communications.

[1] In a two-party mutual authentication protocol, each principal assumes the role of an authenticating principal as well as an authenticated principal.

(G1) and (G2) are high level goals. They represent the expected end results from running a particular authentication protocol. For protocol analysis, these high level goals should be translated into more primitive correctness properties. In our research, we have identified two such basic types of correctness properties, namely, *correspondence* and *secrecy*.

Informally, correspondence means that the execution of the different principals in an authentication protocol proceeds in a locked-step fashion. In particular, when an authenticating principal finishes its part of the protocol, the authenticated principal must have been present and participated in its part of the protocol. Correspondence addresses (G1). Secrecy specifies that certain information should not be accessible to an intruder. Secrecy addresses (G2).

Correspondence and secrecy are two distinct properties, as can be illustrated by the following examples. Before we proceed, we first review the notation generally used for the specification of authentication protocols.

Notation. Principals are denoted by upper case letters, e.g. P, Q. The shared key between P and Q is denoted by k_{PQ}. The public and private keys of P are denoted respectively by k_P and k_P^{-1}. The concatenation of messages m and m' is denoted by m, m'. Encryption of a message m by a key k is denoted by $\{m\}_k$. A protocol is presented as a sequence of *protocol steps*. Each protocol step is written in the form "$P \rightarrow Q : m$" which represents the communication of message m from P to Q. □

The following one-way[2] authentication protocol Π_1 is supposed to authenticate Q to P and to distribute a new session key from P to Q:

$$\begin{array}{lll} (1) & P \rightarrow Q : & P, \{k\}_{k_{PQ}} \\ (2) & Q \rightarrow P : & Q, \{k\}_{k_{PQ}} \end{array}$$

[2]The example can be easily extended to mutual authentication. But it is sufficient here to look at one-way authentication protocols.

Protocol Π_1 is incorrect because the following scenario is possible: (Z denotes an intruder. The principal enclosed in brackets is the intended sender or receiver.)

$$\begin{array}{lll} P \rightarrow (Q)Z & : & P, \{k\}_{k_{PQ}} \\ (Q)Z \rightarrow P & : & Q, \{k\}_{k_{PQ}} \end{array}$$

Specifically, P succeeds in finishing its execution of Π_1 even without the participation of Q. Thus correspondence fails. However, Π_1 does preserve the secrecy of the session key k from intruders. That is, an intruder would not be able to obtain the key distributed in any run of Π_1. Of course, this secrecy property is not too interesting in light of the correspondence failure.

Now consider the following one-way authentication protocol Π_2. Like Π_1, Π_2 is also intended to authenticate Q to P and to distribute a new session key from P to Q.

$$\begin{array}{lll} (1) & P \rightarrow Q : & P, \{k, P\}_{k_P^{-1}} \\ (2) & Q \rightarrow P : & Q, \{k, Q\}_{k_Q^{-1}} \end{array}$$

Protocol Π_2 is incorrect as the session key being distributed can be easily discovered by an intruder Z as shown below:

$$\begin{array}{lll} P \rightarrow (Q)Z & : & P, \{k, P\}_{k_P^{-1}} \\ Z & : & \text{recover } k \text{ by encrypting with } k_P \\ (P)Z \rightarrow Q & : & P, \{k, P\}_{k_P^{-1}} \\ Q \rightarrow P & : & Q, \{k, Q\}_{k_Q^{-1}} \end{array}$$

However, Π_2 does satisfy correspondence. That is, Q must participate in order for P to finish its execution of the protocol. Thus, secrecy fails while correspondence is satisfied in this case.

2.2 Where is Timeliness?

A technique often used to compromise authentication protocols is the so called *replay* attack. This refers to the playback of previously recorded message communications by an intruder in an attempt to sabotage an ongoing exchange. In fact, the first well-publicized authentication protocol

failure is concerned with a combination of replay attack and compromise of old keys [7]. In light of this, it seems that the timeliness of an authentication exchange ought to be included as a basic correctness property. For example, BAN logic includes a special *freshness* construct to distinguish information that belongs to the current session.

In our approach, this is not necessary. In fact, in our model, a timeliness failure would manifest itself in the form of a correspondence failure or a secrecy failure. This is possible in part because of our correspondence definition and the introduction of an explicit action **GetSecret** to model the compromise of old secrets (see discussion in Section 3). For example, the failure reported in [7] is in fact a correspondence failure due to the inability of one of the principals to ensure timeliness. Thus in our approach, we do not have timeliness as a basic correctness property. Instead, it is indirectly taken care of by ensuring both correspondence and secrecy.

2.3 A Realistic Protocol

We have introduced two basic types of correctness properties by looking at several flawed toy authentication protocols. We can gain additional insights by examining a more realistic one. We choose the Otway-Rees protocol [22] for this purpose.

The protocol has a standard cast, consisting of two mutually authenticating principals and a third commonly trusted principal. The protocol is intended to achieve the two goals described in Section 2.1: mutual authentication and key distribution. Here is the Otway-Rees protocol as it appears in [22] (with minor notational changes):

$$
\begin{array}{lll}
(1) & P \rightarrow Q: & n, P, Q, \{n_P, n, P, Q\}_{k_{PS}} \\
(2) & Q \rightarrow S: & n, P, Q, \{n_P, n, P, Q\}_{k_{PS}}, \\
 & \vdots & \quad \{n_Q, n, P, Q\}_{k_{QS}} \\
(3) & S \rightarrow Q: & n, \{n_P, k\}_{k_{PS}}, \{n_Q, k\}_{k_{QS}} \\
(4) & Q \rightarrow P: & n, \{n_P, k\}_{k_{PS}}
\end{array}
$$

P and Q are the principals authenticating each other. S is an *authentication server* trusted by both P and Q, and is also responsible for the generation of the new session key k. The protocol makes use of several nonces,[3] namely n, n_P and n_Q. As is the case with most authentication protocols, there is an *initiator*, P in this case, who initiates the authentication exchange, and a *responder*, Q in this case, who responds to a request for authentication. There exist authentication protocols with multiple initiators[4]; they are in general uninteresting and we do not consider them in our model.

Before we can proceed to prove correspondence and secrecy properties for the protocol, we found that various subtle assumptions are necessary:

- The shared keys of each principal are different. At the least, we must have for all principal P, $k_{PS} \neq k_{ZS}$ where Z is the intruder. Otherwise, the protocol can fail trivially.

  In practice, it is still safe when some keys are the same because the principals may not be aware of the fact. This is similar to the situation in which multiple people can pick the same password without compromising security. However, unless a formal approach can explicitly model awareness, this assumption is necessary.[5]

- Encryption is *ideal*. That is, it is not possible to produce $\{m\}_k$ without knowing m and k, except by eavesdropping. And to recover m from $\{m\}_k$, one must know k^{-1}. In addition, we have the following:

$$
\begin{aligned}
\{m\}_{k_1} = \{m\}_{k_2} &\Rightarrow k_1 = k_2 \\
\{m_1\}_k = \{m_2\}_k &\Rightarrow m_1 = m_2
\end{aligned}
$$

[3] They are called *challenges* in [22].

[4] Here is an example:

$$
\begin{array}{lll}
(1a) & P \rightarrow S: & P, Q, \{n_P, P, Q\}_{k_{PS}} \\
(1b) & Q \rightarrow S: & Q, P, \{n_Q, Q, P\}_{k_{QS}} \\
(2a) & S \rightarrow P: & \{n_P, k\}_{k_{PS}} \\
(2b) & S \rightarrow Q: & \{n_Q, k\}_{k_{QS}}
\end{array}
$$

[5] This assumption is in the same spirit as the so called *Unique Names Assumption* often used in logical reasoning.

which can in turn be replaced by an even stronger assumption:

$$\{m_1\}_{k_1} = \{m_2\}_{k_2} \Rightarrow m_1 = m_2 \wedge k_1 = k_2 \quad \text{(H)}$$

(H) resembles an equality axiom often used for axiomatizing *Herbrand* models and unification. Note that these assumptions are not satisfied by realistic (hence finite) cryptosystems.

Assumptions such as the ones above can be made explicit using a semantic model.

3 A Semantic Model

The first step toward defining a rigorous semantic model for authentication protocols is to precisely characterize authentication protocols themselves. Although the notation used in Section 2.1 is intuitive and has been adopted in the general literature, it lacks the formality needed for our purpose. Referring to the Otway-Rees protocol specification given in Section 2.3, the following observations can be made:

- The specification should be viewed as a protocol *schema*. Specifically, the symbols P, Q, n, n_P, n_Q and k should be viewed as *parameters* that can be instantiated with specific values for different executions of the protocol. Note however that S does represent a fixed principal that does not vary across different executions. Such use of variables and constants should be made explicit.

- The specification in fact describes three distinct *local* protocols, each of which specifies the actions of one of the participating principals. The "global" specification as shown in Section 2.3 can be misleading. For example, in step (1) of the protocol, a more accurate description of the receive action of Q should be n, P, Q, m (where m is a variable) instead of $n, P, Q, \{n_P, n, P, Q\}_{k_{PS}}$. This is because Q is not in a position to determine if m is indeed $\{n_P, n, P, Q\}_{k_{PS}}$ due to its lack of knowledge of k_{PS}. We believe that it is more appropriate to specify the individual local protocols, and we pursue this in our approach.

- The specification includes only communication actions. The assumption is that possible *internal* actions performed by a principal can be abstracted away. For example, the generation of the two nonces n and n_P by P is implicit and so is the checking by S to see that the principals named in the message sent by Q in step (2) match. We find certain abstractions to be beneficial (e.g. the checking by S can be left implicit as long as a consistent pattern matching rule is assumed); but certain other abstractions (e.g. nonce and key generation) should be made explicit to allow a clean and simple semantic model.

Our approach addresses all these points. We start by presenting a formal syntax for specifying authentication protocols. This characterizes the class of authentication protocols of interest to us. The semantic model is defined following the standard state-transition approach. The execution of an authentication protocol is formalized as runs in a state-transition system.

3.1 Syntax

We assume a given set of constant symbols, variable symbols and function symbols. Typical constant symbols include names of principals, nonces and keys[6], while the set of function symbols includes the following: the concatenation function $[\cdot, \ldots, \cdot]$, the set construction function $\{\cdot, \ldots, \cdot\}$, the encryption function $\{\cdot\}_\cdot$, the shared key function $key(\cdot, \cdot)$ and so on.[7]

[6] In other words, there should be a constant symbol for each principal in the system and for each nonce or key that will ever appear.

[7] Note that the more conventional way of denoting these functions are $concat(\cdot, \ldots, \cdot)$, $set(\cdot, \ldots, \cdot)$ and $encrypt(\cdot, \cdot)$. We chose our notation for brevity. Also, for simplicity, we take $[\cdot, \ldots, \cdot]$ and $\{\cdot, \ldots, \cdot\}$ to be variable arity functions.

A *term* is either a variable symbol, a constant symbol, or a function symbol applied to a list of terms. A term is *ground* if it does not contain any variable. A *primitive* term is a ground term that is a constant symbol.

Notation. We typeset constant symbols in Sans Serif font, e.g. P, N. The set of names for all principals in the system is denoted by SYS. Note that all such names are primitive terms. In particular, SYS contains the distinguished constant symbol Z, which denotes the intruder. In the Otway-Rees example, SYS also contains another distinguished constant symbol S, which denotes the trusted server.

In our protocol specification language, variables are typeset in two ways. Lower case variables stand for primitive terms, while upper case bold face variables stand for arbitrary terms. We assume that variables x, y range over SYS, variables i, r range over SYS $-$ {S, Z} and variable p ranges over SYS $-$ {Z}. The same convention is followed in the assertion language to be introduced in Section 4. □

We also assume a given set of *labels*. A *protocol statement* (*statement* in short) is of the form ℓ : *act* where ℓ is a label and *act* is an instance of one of the following *action* schema (where $i, r, n, p, \mathbf{M}, \mathbf{N}, \mathbf{S}$ are terms):

BeginInit (r)	NewNonce (n)
EndInit (r)	NewSecret $(\mathbf{S}, n)$
BeginRespond (i)	Send $(p, \mathbf{M})$
EndRespond (i)	Receive $(p, \mathbf{M})$
Accept $(\mathbf{N})$	GetSecret (n)

Thus each action consists of a keyword (e.g. BeginInit, EndInit, and so on), called the *command* of the action and a list of terms, called the *arguments* of the action.

The informal meaning of each action should be intuitive from its command, perhaps with the exception of Accept and GetSecret. For example, BeginInit(r) and EndInit(r) mark the beginning and end of an initiator protocol with r denoting the responder. Similarly, BeginRespond and EndRespond mark the beginning and end of a responder protocol with i denoting the initiator. NewNonce(n) returns a new nonce n; while NewSecret($\mathbf{S}, n$) returns a new secret n to be shared among the principals named in the set S. Secrets are mainly used for keys. Accept(N) is intended to make explicit the acceptance of a new session key N being distributed. A key becomes *old* as soon as all of the principals that are intended to share the key have accepted the key. An old key can be compromised (e.g. by using off-line cryptanalysis) and this is modeled by the GetSecret statement, which returns an old key. The precise semantics of each action is defined formally in Section 3.2.

A statement is *ground* if all arguments of its action are ground. A *local protocol* (*protocol* in short) is a *finite* sequence of statements that does not include a GetSecret statement. A protocol is *ground* if it contains only ground statements.

An *initial condition* is a first order formula constructed using two binary predicates, namely, "has" for possession and "=" for equality, and standard logical connectives. Informally, x has $\mathbf{M}$ holds if the term $\mathbf{M}$ is in the possession of principal x (see Section 3.2 for a more precise definition). An initial condition specifies the set of terms each principal has before protocol execution begins.

Definition. A *protocol specification* is an ordered pair $(Init, ProSet)$ where *Init* is a finite set of initial conditions and *ProSet* a finite nonempty set of protocols such that all statement labels appearing in protocols of *ProSet* are distinct. □

As an example, Figure 1 shows the Otway-Rees protocol formally specified using our syntax. It consists of two initial conditions and three separate protocols, one for each of the participating principals. Thus formally, the protocol specification for the Otway-Rees protocol is

({(1), (2)}, {Initiator(i), Responder(r), Server(S)})

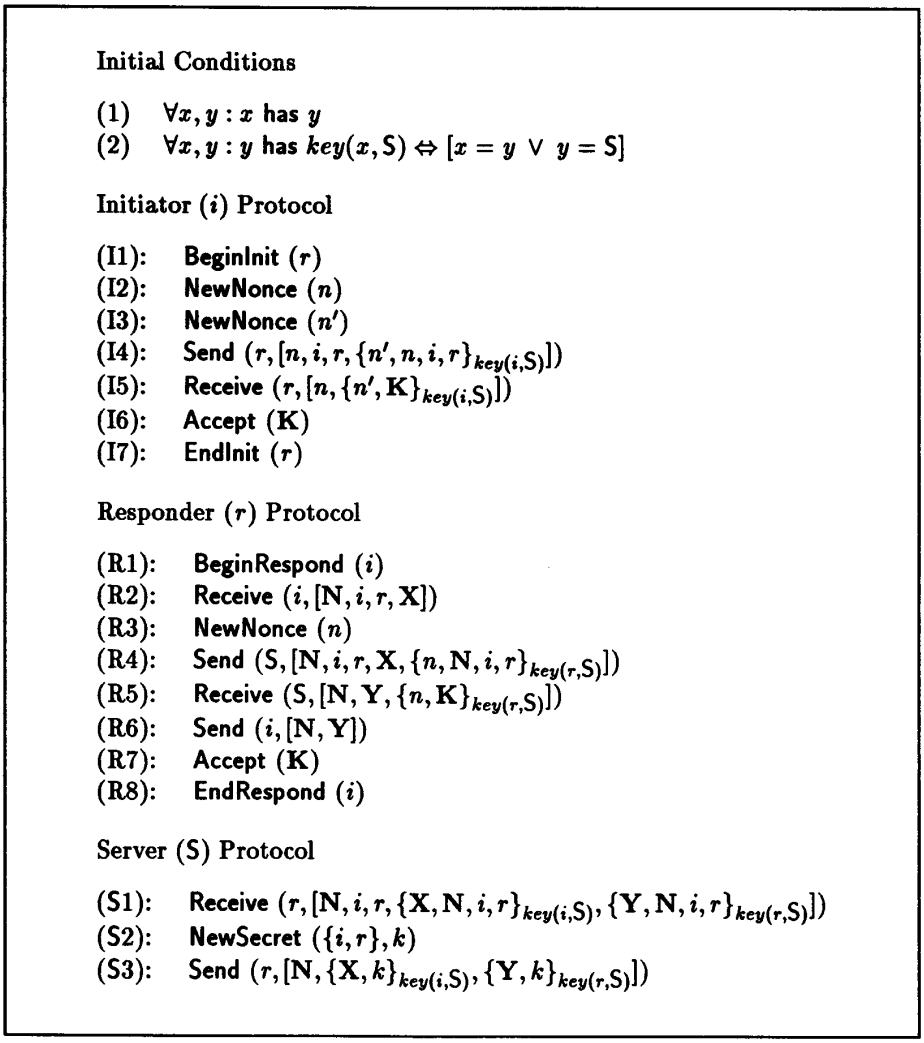

Figure 1: Otway-Rees Protocol Specification

(1) specifies that each principal in the system knows the names of all other principals in the system, whereas (2) specifies that the secret key $key(x, \mathsf{S})$ of principal x is shared only between itself and S. This condition implies that the secret key of each principal is distinct, i.e. $\forall x, y : x \neq y \Rightarrow key(x, \mathsf{S}) \neq key(y, \mathsf{S})$.

Note that both the initiator and responder protocols can be instantiated by arbitrary principals, and are called *instantiable* protocols. The server protocol, on the other hand, can only be performed by the constant principal S (a trusted server in the system) and is called a *fixed* protocol, with S as its *fixed* principal.

3.2 Semantics

The semantics of a protocol specification is given by the set of its possible executions. From a semantic viewpoint, a protocol specification can be viewed as a shorthand representation of the set

of executions. In our research, we characterize these executions using a state-transition based approach.

To simplify the presentation, we assume throughout this subsection a given protocol specification $\Pi = (Init, \{\mathcal{P}_1(x_1), \ldots, \mathcal{P}_L(x_L)\})$ where *Init* is a set of initial conditions and each $\mathcal{P}_i(x_i)$ is a protocol where x_i stands for a variable or constant principal.

A *substitution* is a *nonempty* set of pairs of the form t/x where t is a term and x a variable, with the restriction that it does not contain two pairs t/x and t'/x where t differs from t'. A substitution can be applied to a term, a statement or a protocol. Let *obj* be an object to which a substitution θ is to be applied. Then the application of θ to *obj*, denoted by $obj : \theta$, is defined as the concurrent replacement of all occurrences of x in *obj* by t, for every pair t/x in θ. We denote by $domain(\theta)$ the set $\{x \mid t/x \in \theta\}$ and $range(\theta)$ the set $\{t \mid t/x \in \theta\}$. We say a substitution θ is *over* the set $domain(\theta)$ of variables. A substitution θ is *ground* if every term in $range(\theta)$ is ground. For example, consider the statement $stmt$ = (I4) $\mathsf{Send}(r, [n, i, r, \{n', n, i, r\}_{key(i,S)}])$ and the ground substitution $\theta = \{\mathsf{P}/i, \mathsf{Q}/r, \mathsf{N}/n, \mathsf{N}'/n'\}$. We have $domain(\theta) = \{i, r, n, n'\}$, $range(\theta) = \{\mathsf{P}, \mathsf{Q}, \mathsf{N}, \mathsf{N}'\}$ and $stmt : \theta$ is (I4) $\mathsf{Send}(\mathsf{Q}, [\mathsf{N}, \mathsf{P}, \mathsf{Q}, \{\mathsf{N}', \mathsf{N}, \mathsf{P}, \mathsf{Q}\}_{key(\mathsf{P},\mathsf{S})}])$.

A *step* is of the form $(id, stmt)$ where id is a session id and $stmt$ a ground statement. Let ξ be a sequence of steps. ξ is said to be a *partial instance* of a protocol $\mathcal{P}_i$ if all steps of ξ contain the same session id and there exists a substitution θ such that $\mathcal{P}_i(x_i) : \theta$ is ground and the sequence of statements appearing in ξ is a prefix of $\mathcal{P}_i(x_i) : \theta$. For example, the following is a partial instance of the Initiator protocol of Otway-Rees:

(5, (I1): **BeginInit** (Q))
(5, (I2): **NewNonce** (N))
(5, (I3): **NewNonce** (N'))
(5, (I4): **Send** (Q, [N, P, Q, {N', N, P, Q}$_{key(\mathsf{P},\mathsf{S})}$]))

Intuitively, a partial instance models the partial execution of an instance of a protocol. Note that a partial instance of a protocol always starts from the beginning of a protocol, and the empty sequence is a partial instance of any protocol.

Let ξ be a sequence of steps. We denote by ξ_{id} the subsequence of ξ consisting of all steps whose session id is id. Let X be a principal. If $X \neq \mathsf{Z}$, we say ξ is a *fragment* of X if for all session id id, ξ_{id} is a partial instance of some protocol $\mathcal{P}_i(X)$.[8] If $X = \mathsf{Z}$, any arbitrary sequence of steps is a fragment of Z. Thus a fragment of a principal X ($X \neq \mathsf{Z}$) is simply an interleaving of partial executions of protocols that X can perform. In the case of Z, any sequence of steps is a fragment as Z is not required to follow any protocol. Note also that by definition a protocol does not include a **GetSecret** statement, thus no fragment of a principal X ($X \neq \mathsf{Z}$) can contain a **GetSecret** step. In other words, only Z can perform a **GetSecret** action.

A *local state* of X is an ordered pair $(frag, info)$ where $frag$ is a fragment of X and $info$ a set of ground terms. A *local transition* is simply a ground statement. For brevity, we sometimes identify a local transition simply by its action. For example, we may refer to the local transition (I1):**BeginInit**(Q) as a **BeginInit**(Q) transition or even simply as a **BeginInit** transition when the argument Q is not relevant. Let α be a local transition. Then α is *enabled at* a local state $(frag, info)$ of X if:

- there exists a session id id such that $frag@(id, \alpha)$ is a fragment of X, where @ denotes the concatenation operation, and
- if α is a $\mathsf{Send}(Y, M)$ transition, then $M \in info$. This stipulates that a principal can only send messages that it possesses.

Notation. In a meta-level discussion such as the above, we use upper case italic letters to represent ground terms. For example, $\mathsf{Send}(Y, M)$

[8] Thus if $\mathcal{P}$ is a fixed protocol (e.g. the server protocol of Otway-Rees), then its partial instance can only appear in the fragments of its fixed principal (e.g. S in the case of the server protocol of Otway-Rees).

(U1) $u \in U \Rightarrow u \in update(U,t)$
(U2) $t \in update(U,t)$
(U3) $[u_1,\ldots,u_n] \in update(U,t) \Leftrightarrow u_1 \in update(U,t) \wedge \ldots \wedge u_n \in update(U,t)$
(U4) $\{u\}_v \in update(U,t) \wedge v^{-1} \in update(U,t) \Rightarrow u \in update(U,t)$
(U5) $u \in update(U,t) \wedge v \in update(U,t) \Rightarrow \{u\}_v \in update(U,t)$
(U6) $\{\{u\}_v\}_{v^{-1}} \in update(U,t) \Rightarrow u \in update(U,t)$
(U7) $\{\{u\}_{v^{-1}}\}_v \in update(U,t) \Rightarrow u \in update(U,t)$

Table 1: An axiomatization of *update*

above stands for a specific transition, with Y denoting the constant name of a principal and M denoting a ground message term. □

Let $(frag, info)$ and $(frag', info')$ be two local states of X, and α a local transition. Then α *takes* $(frag, info)$ *to* $(frag', info')$, denoted by

$$(frag, info) \xrightarrow{\alpha} (frag', info')$$

if

- α is enabled at $(frag, info)$,
- $frag' = frag@(id, \alpha)$ for some session id id,
- if α is a **BeginInit**, **EndInit**, **BeginRespond**, **EndRespond**, **Accept** or **Send** transition, then $info' = info$,
- if α is a **Receive**(Y, M) transition, then $info' = update(info, M)$,
- if α is a **NewNonce**(N), **NewSecret**(S, N) or **GetSecret**(N) transition, then $info' = update(info, N)$

where $update(U,t)$ denotes a procedure that returns a new set of terms and whose inputs are a set U of terms and a term t. It models the update of the state of knowledge of a principal after some new information has been obtained. The precise definition of *update* is determined by the computation power and knowledge of a principal. For example, a reasonable definition of *update* is the smallest set of terms that satisfies the axiomatization in Table 1.

Note that (U3)–(U5) imply unlimited computation power, while (U6)–(U7) encode information about the underlying cryptosystem. Additional axioms can be included when more knowledge about the cryptosystem is available. It is also possible to customize the definition of *update* for each principal to reflect the particular reasoning power each principal has. This can be achieved by defining a different $update_X$ for each principal X.

A *global state* is a tuple of *local states*, one for each principal in the system. Given a global state s, we denote the local state of principal X in s by $s.X$. A *global transition* can be: (i) an *internal* transition of the form $(X.\ell, act)$ where X is a principal and $\ell : act$ a local transition other than **Send** or **Receive**, or (ii) a *communication* transition of the form $(X.\ell, Y.\ell', \mathsf{Comm}(M))$ where X, Y are principals, ℓ, ℓ' labels and M a ground term.

Let s be a global state and β a global transition. β is *enabled at* s if:

- $\beta = (X.\ell, act)$ is an internal transition and the local transition $\ell : act$ is enabled at $s.X$,
- $\beta = (X.\ell, Y.\ell', \mathsf{Comm}(M))$ is a communication transition and there exists principals S, T, U, V (not all distinct) such that the following three conditions hold:
  - the local transition ℓ : **Send**(T, M) is enabled at $s.S$,
  - the local transition ℓ' : **Receive**(V, M) is enabled at $s.U$,
  - one of the following holds: Case (i) if $X \neq \mathsf{Z}$ and $Y \neq \mathsf{Z}$, then $S = V = X$

and $T = U = Y$. Case (ii) if $X = \mathsf{Z}$ and $Y \neq \mathsf{Z}$, then $S = \mathsf{Z}$ and $T = U = Y$. Case (iii) if $X \neq \mathsf{Z}$ and $Y = \mathsf{Z}$, then $S = V = X$ and $U = \mathsf{Z}$.

Note that the case of $X = \mathsf{Z}$ and $Y = \mathsf{Z}$ is not possible.

We call $(S.\ell, \mathsf{Send}(T, M))$ and $(U.\ell', \mathsf{Receive}(V, M))$ the *send* and *receive components* of β respectively.

Let s and s' be two global states and β a global transition. We say β *takes* s *to* s', denoted by $s \xrightarrow{\beta} s'$ if:

- β is enabled at s,
- if $\beta = (X.\ell, act)$ is an internal transition, then $s.X \xrightarrow{\alpha} s'.X$ holds where $\alpha = \ell : act$, and for all $Y \neq X$, $s'.Y = s.Y$,
- if $\beta = (X.\ell, Y.\ell', \mathsf{Comm}(M))$ is a communication transition and $(S.\ell, \mathsf{Send}(T, M))$ and $(U.\ell', \mathsf{Receive}(V, M))$ are respectively the send and receive components of β, then $s.S \xrightarrow{\alpha} s'.S$, $s.U \xrightarrow{\alpha'} s'.U$ where $\alpha = \ell : \mathsf{Send}(T, M)$ and $\alpha' = \ell' : \mathsf{Receive}(V, M)$, and for all W such that $W \neq S$ and $W \neq U$, $s'.W = s.W$.

An *initial state* is a global state that satisfies the set of initial conditions. The precise satisfaction relation, denoted by $\models$, is similar to that of standard first order logic and can be defined via recursion on the syntax of the formulas. We omit the definition here except to note that for a formula of the form X **has** M and a global state s, $s \models X$ **has** M if $s.X = (frag, info)$ and $M \in info$.

Definition. An *execution* is a *finite* alternate sequence of global states and transitions, of the form $s_1\beta_1 s_2\beta_2 \ldots s_n$, such that:

(i) s_1 is an initial state

(ii) for all $1 \leq i < n$, $s_i \xrightarrow{\beta_i} s_{i+1}$ holds

(iii) if $\beta_i = (X.\ell, \mathsf{GetSecret}(N))$, then there exists $1 \leq j < i$, $Y \neq \mathsf{Z}$ and S such that (a) $\beta_j = (Y.\ell', \mathsf{NewSecret}(S, N))$, and (b) $S = \{U \mid j < k < i \wedge \beta_k = (U.\ell'', \mathsf{Accept}(N))\}$

(iv) (Uniqueness of Secrets)

if $\beta_i = (X.\ell, \mathsf{NewSecret}(S, N))$, then N must be a new primitive term.

(v) (Uniqueness of Nonces)

if $\beta_i = (X.\ell, \mathsf{NewNonce}(N))$ then N must be a new primitive term.

(iii) specifies that a GetSecret transition can only return old keys, and not one whose distribution is still ongoing. (iii)(a) ensures that the secret must have been generated earlier in the execution; while (iii)(b) ensures that all principals for whom the secret is intended have accepted the secret, and hence its distribution has been completed. (iv) specifies that secrets are always new and primitive. This ensures that principals cannot stumble onto a secret (hence violating its secrecy), thus avoiding the problem of "accidental" discovery of a secret by another principal due to reuse. (v) specifies a condition similar to (iv) for nonces. It ensures the freshness or "used only once" property for nonces. It avoids the problem of "chance" equalities between a nonce and an arbitrary message term. We assume nonces to be globally fresh, and they are secret. If nonces are only locally fresh[9], then (v) can be replaced by the following:

if $\beta_i = (X.\ell, \mathsf{NewNonce}(N))$, then N must be a primitive term and that $N \notin \{N' \mid 1 \leq j < i \wedge \beta_j = (X.\ell', \mathsf{NewNonce}(N'))\}$.

That is, relative to each principal, a nonce cannot be generated twice. However, the same nonce can be generated by two different principals independently. □

[9] For example, when nonces are implemented by a timestamp or a counter. Note that a locally fresh nonce is by definition not secret.

Definition. Let Π be a protocol specification. The *semantics* of Π is the set of all executions. □

In what follows, we use Π to refer also to the semantics of Π when the context is clear. We observe the following important property of our semantics.

Proposition. The semantics of a protocol specification is *prefix closed.* That is, let Π be a protocol specification and $\xi \in \Pi$. Then any prefix of ξ that ends in a state also belongs to Π.

Proof. Immediate from the definition of execution. □

This prefix closure property is crucial from the point of view of analysis. It provides the basis for *inductive* techniques such as *invariance* verification.

4 Formalizing Correctness

Given the semantic model in Section 3, we are now ready to examine how correctness can be formalized. In particular, we present a formalization of correspondence and secrecy properties introduced informally in Section 2.1. A complete presentation, with extended syntax and additional assertion types, can be found in [26]. The following presentation is intended to illustrate the essential ideas and general flavor of our approach.

4.1 Correspondence

Correspondence properties are specified using *correspondence assertions.* We first consider the ground case, and then extend our definition to the general case. A *ground correspondence assertion* is a formula of the form:

$$\gamma_1 \mid \ldots \mid \gamma_k \hookrightarrow \mu_1 \mid \ldots \mid \mu_\ell$$

where γ_i's and μ_i's are global transitions. The meaning of the assertion is defined with respect to an execution. Informally, an execution *satisfies* the assertion if for each appearance of a global transition γ_i $(1 \leq i \leq k)$ in the execution, there exists an appearance of a global transition in the set $\{\mu_1, \ldots, \mu_\ell\}$ that strictly precedes γ_i in the execution. The operator "|" can be interpreted as "or".

For example, the following are two ground correspondence assertions appropriate for the Otway-Rees protocol: Let P,Q be two principals in SYS distinct from Z and S:[10]

$$(\mathsf{P}.(\mathrm{I7}), \mathsf{EndInit}(\mathsf{Q})) \hookrightarrow (\mathsf{Q}.(\mathrm{R1}), \mathsf{BeginRespond}(\mathsf{P})) \quad (1)$$

$$(\mathsf{Q}.(\mathrm{R8}), \mathsf{EndRespond}(\mathsf{P})) \hookrightarrow (\mathsf{P}.(\mathrm{I1}), \mathsf{BeginInit}(\mathsf{Q})) \quad (2)$$

(1) specifies that every time P takes a EndInit(Q) transition, Q must have taken a distinct BeginResponse(P) transition earlier. That is, whenever P finishes executing an authentication exchange with Q, Q must have taken at least its first step in such an exchange. The distinctness guarantees that there are at least as many BeginResponse(P) transitions by Q as there are EndInit(Q) by P. (2) specifies a similar property, but from Q's perspective. (1) and (2) together guarantee that authentication exchanges between P and Q proceed in a "locked-step" fashion, thus addressing the authentication concerns of Section 2.1.

The labels in a correspondence assertion can be omitted if the actions uniquely identify the labels. For example, this is the case for (1) and (2).

Clearly, assertions similar to (1) and (2) should be stated for all principals in the system (except Z and S). This is achieved by using the following general correspondence assertions:

$$(i, \mathsf{EndInit}(r)) \hookrightarrow (r, \mathsf{BeginRespond}(i)) \quad (\mathrm{A1})$$

$$(r, \mathsf{EndRespond}(i)) \hookrightarrow (i, \mathsf{BeginInit}(r)) \quad (\mathrm{A2})$$

[10] Strictly speaking, S can also initiate or respond to authentication exchanges. Its behavior would however be no different from an ordinary principal. Thus for simplicity, we only consider S for its special role as a server and do not consider its own authentications.

where i and r are variables. (Recall that i and r range over $\mathsf{SYS} - \{\mathsf{S}, \mathsf{Z}\}$.) A *general correspondence assertion* is similar to a ground correspondence assertion except that some of its component terms may not be ground. A general correspondence assertion can be viewed as a shorthand for the set of ground correspondence assertions that can be obtained from it using some ground substitution.[11] For example, (A1) subsumes (1) and (A2) subsumes (2) with the substitution $\{\mathsf{P}/i, \mathsf{Q}/r\}$. The semantics described for the ground case can thus be naturally extended to the general case.

We now define a formal *satisfaction* relation between an execution and a ground correspondence assertion. Let $\xi = s_1\beta_1 s_2\beta_2 \ldots s_n$ be an execution and $\rho = \gamma_1 \mid \ldots \mid \gamma_k \hookrightarrow \mu_1 \mid \ldots \mid \mu_\ell$ be a ground correspondence assertion. Define

$$I = \{i \mid \beta_i \in \{\gamma_1, \ldots, \gamma_k\}\}$$

Then ξ *satisfies* ρ if there exists a function f from I to the set of natural numbers such that

- $f(i) < i$, and
- $\beta_{f(i)} \in \{\mu_1, \ldots, \mu_\ell\}$, and
- f is one-to-one.

Definition. Let Π be a protocol specification and ρ a correspondence assertion. Then Π *satisfies* ρ if for all execution $\xi \in \Pi$, ξ satisfies ρ. □

4.2 Secrecy

The secrecy specification of an authentication protocol can be divided into two parts: a general part and a specific part. The general part applies to all protocol specifications while the specific part is customized for each authentication protocol.

[11] Strictly speaking, this is true only for correspondence assertions whose left and right hand sides contain the same set of variables. The semantics is different if this is not the case. See [26] for details.

The general part consists of a *general secrecy condition*, saying that the intruder Z cannot discover any secret not intended for it other than those it has successfully compromised using GetSecret. Formally, the condition is defined as follows: Let $\xi = s_1\beta_1 s_2\beta_2 \ldots s_n$ be an execution. Define:

$$NS_\xi = \left\{ N \;\middle|\; \begin{array}{l} \exists i : \beta_i = (X.\ell, \mathsf{NewSecret}(S, N)) \\ \wedge\, X \neq \mathsf{Z} \wedge \mathsf{Z} \notin S \\ \wedge\, \forall Y \in S : \\ \exists j : \beta_j = (Y.\ell', \mathsf{Accept}(N)) \end{array} \right\}$$

$$GS_\xi = \{N \mid \exists i : \beta_i = (\mathsf{Z}.\ell, \mathsf{GetSecret}(N))\}$$

Now let $s_n.\mathsf{Z} = (frag, info)$. Then ξ satisfies the *general secrecy condition* if

$$(NS_\xi - GS_\xi) \cap info = \emptyset$$

Note that NS_ξ contains all secrets that have been generated by a principal other than Z in the course of protocol execution and are not intended for Z, while GS_ξ contains all secrets that Z has successfully compromised using GetSecret. Thus $(NS_\xi - GS_\xi)$ represents the set of secrets that should not be accessible to Z.

The specific part consists of *secrecy assertions*. A secrecy assertion is specified using formulas similar to those used for initial conditions. The formulas however are interpreted in the *final* state of an execution, as opposed to the initial state for initial conditions. That is, an execution $\xi = s_1\beta_1 s_2\beta_2 \ldots s_n$ *satisfies* a secrecy assertion f if $s_n \models f$. As an example, the following secrecy assertion is appropriate for the Otway-Rees protocol: (Recall that p ranges over $\mathsf{SYS} - \{\mathsf{Z}\}$.)

$$\forall x, p : x \text{ has } key(p, \mathsf{S}) \Leftrightarrow [x = p \vee x = \mathsf{S}] \quad \text{(S)}$$

Note that (S) is similar to the initial condition (2) specified in Figure 1, which says that the secret key of each principal should be shared only between itself and S. The main difference is that (S) does not specify that $key(\mathsf{Z}, \mathsf{S})$ remains secret because Z can always choose to reveal its secret key to others (as it is not bound by any protocol), and this should not constitute a violation of secrecy.

Definition. Let Π be a protocol specification. Then Π *satisfies* the general secrecy condition if for all execution $\xi \in \Pi$, ξ satisfies the general secrecy condition. □

Definition. Let Π be a protocol specification and f a secrecy assertion. Then Π *satisfies* f if for all execution $\xi \in \Pi$, ξ satisfies f. □

4.3 Timeliness

We mention in Section 2.2 that timeliness concerns can be taken care of by correspondence and secrecy properties. We illustrate this here using the Otway-Rees protocol as an example.

Informally, timeliness means that a principal should be able to distinguish messages of a current authentication exchange from those of earlier exchanges. In particular, a principal should only accept keys being distributed in a current authentication exchange, and not old keys from earlier exchanges that might have been compromised.

The requirement that a principal cannot accept compromised old keys is part of the general secrecy condition. The following assertion specifies the requirement that a principal can only accept keys previously generated by the trusted server S:

$$(r, \mathsf{Accept}(n)) \hookrightarrow \begin{array}{l}(\mathsf{S}, \mathsf{NewSecret}(\{i, r\}, n)) \\ \mid (\mathsf{S}, \mathsf{NewSecret}(\{r, i\}, n))\end{array} \quad \text{(T)}$$

We can now make precise our statement that timeliness concerns are taken care of by correspondence and secrecy properties. Formally, this means if the Otway-Rees protocol satisfies correspondence assertions (A1) and (A2), the secrecy assertion (S) and the general secrecy condition, then it also satisfies (T). Of course, this statement remains to be proved.

4.4 Correctness Specification

A *correctness specification* is an order pair $(\mathcal{C}, \mathcal{S})$ where $\mathcal{C}$ is a set of correspondence assertions and $\mathcal{S}$ a set of secrecy assertions.

Definition. Let Π be a protocol specification and $\Phi = (\mathcal{C}, \mathcal{S})$ a correctness specification. Then Π is *correct with respect to* Φ if

- for all correspondence assertion $f \in \mathcal{C}$, Π satisfies f
- for all secrecy assertion $f \in \mathcal{S}$, Π satisfies f
- Π satisfies the general secrecy condition

□

5 Comparison with Other Work

Many approaches have been proposed for the formal analysis of authentication protocols. For brevity, we will only attempt to compare our approach with those that are closely related to ours.

Dolev and Yao [8] proposed the first model for formalizing protocol correctness.[12] The model they proposed is algebraic in nature. The correctness concern is the secrecy of some specific initial message terms, and their focus is on finding efficient algorithms for deciding correctness. The class of protocols we consider is more general than theirs; thus most of their formalization and techniques are not directly applicable. However, our approach does rely on an underlying algebraic foundation similar to theirs, and so do several other approaches, including the work by Meadows [17].

The work of Meadows [17] is similar in spirit to ours. Her model is an adaptation of the Dolev and Yao model, and her interest is in protocols similar to ours. In her approach, a protocol is specified by a set of formal rules for generating words in a term-rewriting language. Correctness concerns (e.g. authentication and secrecy) are formulated as questions on the *unobtainability* of certain words in a formal language. Such questions are in turn answered by showing that certain states are *unreachable*. However, except

[12] The protocols they considered were called *ping pong* protocols as they involve only two parties. These protocols are mainly intended for distributing secrets rather than for mutual authentication.

for the underlying algebraic model, Meadows's approach and focus are different from ours. Our protocol specifications are designed to resemble programs, and our semantic model characterizes all possible protocol executions. Meadows' protocols, on the other hand, are specified from the viewpoint of the intruder and her model is concerned mainly with the evolution of the intruder's state. In her approach, high level correctness concerns (e.g. authentication) are reduced to low level questions of unobtainability, while our approach reduces the high level concerns to two basic types of correctness properties. Our secrecy assertions can be used to express unobtainability, while our correspondence assertions are flexible and can be customized for individual protocols.

Another major class of formal analysis approaches is based on the use of formal logic. This includes BAN logic [2, 6] and its extensions [10, 11], Syverson's KPL [23] and Bieber's logic [4]. Kemmerer's approach [12] can also be included here as it is formalized using a first order logic. Most of these approaches (except Kemmerer's) are based on the use of some sort of *modal logic of belief or knowledge* with a *possible worlds* semantics.

The BAN logic approach is probably the most widely studied one because of its intuitive appeal. The BAN logic approach assumes a much higher level of abstraction than ours. For example, a belief of the form P **believes** $P \overset{k}{\leftrightarrow} Q$ does not directly reduce to any combination of correspondence and secrecy properties. (However, we conjecture that a protocol satisfying an appropriate set of correspondence and secrecy properties would also satisfy some of the goals suggested in [6, p. 12].) In general, the constructs in BAN logic and our assertions are not comparable. For example, the **control** construct is not expressible in our model, as our model does not address *trust* issues. Another major difference between our approach and the BAN logic approach is in the formal representation of protocols. In BAN logic, an authentication protocol is specified by a sequence of formulas in the logic. This specification can only be obtained by an informal *idealization* process that requires a thorough understanding of the protocol. We find idealization undesirable because of the potentially large semantic gap that exists between the original protocol and the idealized version (see for example the idealization of the Yahalom protocol in [6, pp. 29–30]). Our specifications are concrete algebraic programs. Thus we trade abstraction for concreteness. We gain by being able to focus more concretely on some specialized properties, and we lose in expressive power and the ability to capture certain notions (e.g. trust). A formal semantic model was defined for BAN logic in [2]. However, the model is intended more for explaining the constructs in the logic than as a basis for a general formalization of correctness. Lastly, the primary focus of BAN logic is on authentication concerns while secrecy concerns are left implicit [6, p. 2]. Our approach treats both kinds of concerns explicitly.

The work in [23] and [4] are much more abstract and are more concerned with meta-level investigations than with the analysis of concrete authentication protocols. For example, one of the goals of Syverson's KPL is to clarify the two kinds of "*knowing*" that are possible, namely, knowing that a proposition is true and knowing (possessing) a concrete message term. Bieber does discuss both secrecy and authentication. However, his focus is on message authentication, rather than identity authentication. Their logics are general, and can easily express the kind of correctness properties we have defined. Such generality is however not necessary and our goal is to precisely identify and define the kinds of assertions needed to specify correctness.

The importance of semantics has also been emphasized by Syverson in [23, 24], but his focus is on logic rather than a general formalization of correctness. He gave similar observations on potential misinterpretations when correctness notions are insufficiently formalized.

Finally, we would like to mention the work by Millen et. al. [18] on the Interrogator protocol analyzer. In their model, a protocol is viewed

as a collection of communicating finite state machines. Each such machine is similar in concept to a local protocol, although a local protocol is potentially of infinite states. Their model is also state-transition based. Their *network global state* corresponds roughly to a global state in our model except that they use a *network buffer* to model messages in transit. Their *message history* corresponds roughly to our *executions* except that they include only communication transitions. Their main concern is on secrecy of particular message terms and their focus is on a software tool.

6 Discussion

Our approach is not unlike various approaches taken in the study of programming language semantics and program verification. In some sense, the syntax we have defined for authentication protocols can be viewed as a very small programming language. An authentication protocol corresponds to a program and its operational meaning is given by our semantic model. Of course, the technical machineries used in defining our semantics are quite different from those used for most programming languages. In particular, our semantics has to allow for an intruder Z who does not follow a specified protocol.

Our use of an explicit GetSecret transition to model the compromise of old keys is novel. This is a reason why we can eliminate timeliness as a separate correctness property. In fact, it is possible to define a more restrictive semantics for authentication protocols where GetSecret is not allowed in any execution. A protocol that does not ensure timeliness may be correct under this more restrictive semantics but not in the semantics we have defined.

Our current model focuses only on nonce-based authentication protocols. It should be possible to extend our model to handle timestamp-based protocols. The abstract properties of nonces and timestamps are similar, except that a *window of acceptance* is needed for timestamps but not for nonces.

In formulating our semantic model, we have avoided tackling the more philosophical issue of *awareness*. We make the implicit assumption that a principal is immediately aware of the discovery of a secret even if it just accidentally stumbles upon it. This is certainly pessimistic. To pursue the alternative approach of making a distinction between awareness and possession, a clear understanding of how to formalize awareness is required. Some initial attempt in adding awareness to an algebraic model is reported in [16]. General logical perspectives on formalizing awareness can be found in [9, 13, 15].

We have presented a semantic model for authentication protocols and a formalization of two basic types of correctness properties. We believe that these are necessary groundwork that must be in place before we can propose and study analysis techniques. In particular, given the semantic model and our correctness properties, the question of soundness and completeness of a particular analysis technique can be precisely answered.

We are currently studying analysis techniques for proving correspondence and secrecy properties.

Acknowledgements

The authors would like to thank G. Neelakantan Kartha and David Yau of the University of Texas at Austin, and the anonymous referees for their helpful comments on revising the paper.

References

[1] *The Computer Security Foundations Workshop*, Franconia, New Hampshire, June 12–14 1990. IEEE Computer Society Press.

[2] M. Abadi and M.R. Tuttle. A semantics for a logic of authentication (extended abstract). In *Proceedings of the 10th ACM Symposium on Principles of Distributed Computing*, pages 201–216, August 1991.

[3] R.K. Bauer, T.A. Berson, and R.J. Feiertag. A key distribution protocol using event

markers. *ACM Transactions on Computer Systems*, 1(3):249–255, August 1983.

[4] P. Bieber. A logic of communication in hostile environment. In *Proceedings of the The Computer Security Foundations Workshop III* [1], pages 14–22.

[5] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic design of a family of attack-resistant authentication protocols. Technical report, IBM Raleigh, Watson & Zurich Laboratories, April 1992.

[6] M. Burrows, M. Abadi, and R.M. Needham. A logic of authentication. Technical Report 39, Systems Research Center, Digital Equipment Corporation, February 28 1989. Revised February 22, 1990. An abbreviated version appears in *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.

[7] D.E. Denning and G.M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, August 1981.

[8] D. Dolev and A.C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(2):198–208, March 1983.

[9] R. Fagin and J.Y. Halpern. Belief, awareness, and limited reasoning. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 479–490, Los Angeles, California, 1985.

[10] L. Gong, R.M. Needham, and R. Yahalom. Reasoning about belief in cryptographic protocols. In *Proceedings of the 11th IEEE Symposium on Research in Security and Privacy*, pages 234–248, Oakland, California, May 7–9 1990. IEEE Computer Society Press.

[11] R. Kailar and V.D. Gilgor. On belief evolution in authentication protocols. In *Proceedings of the The Computer Security Foundations Workshop IV*, pages 103–116, Franconia, New Hampshire, June 18–20 1991. IEEE Computer Society Press.

[12] R.A. Kemmerer. Analyzing encryption protocols using formal techniques. *IEEE Journal on Selected Areas in Communications*, 7(4):448–457, May 1989.

[13] K. Konolige. What awareness isn't: A sentential view of implicit and explicit belief. In *Proceedings of the 1st Theoretical Aspects of Reasoning About Knowledge*, pages 241–250, Monterey, Californiaa, March 19–22 1986.

[14] B. Lampson, M. Abadi, M. Burrows, and T. Wobber. Authentication in distributed systems: Theory and practice. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, pages 165–182, Asilomar Conference Center, Pacific Grove, CA, October 13–16 1991.

[15] H.J. Levesque. A logic of implicit and explicit belief. In *Proceedings of the National Conference on Artificial Intelligence*, pages 198–202, Houston, TX, 1984.

[16] C. Meadows. Representing partial knowledge in an algebraic security model. In *Proceedings of the The Computer Security Foundations Workshop III* [1], pages 23–31.

[17] C. Meadows. Applying formal methods to the analysis of a key management protocol. *Journal of Computer Security*, 1(1):5–35, 1992.

[18] J.K. Millen, S.C. Clark, and S.B. Freedman. The Interrogator: Protocol security analysis. *IEEE Transactions on Software Engineering*, 13(2):274–288, February 1987.

[19] J.H. Moore. Protocol failures in cryptosystems. *Proceedings of the IEEE*, 76(5):594–602, May 1988.

[20] R.M. Needham and M.D. Schroeder. Authentication revisited. *ACM Operating Systems Review*, 21(1):7, January 1987.

[21] D. Nessett. A critique of the Burrows, Abadi and Needham logic. *ACM Operating Systems Review*, 24(2):35–38, April 1990.

[22] D. Otway and O. Rees. Efficient and timely mutual authentication. *ACM Operating Systems Review*, 21(1):8–10, January 1987.

[23] P. Syverson. Formal semantics for logics of cryptographic protocols. In *Proceedings of the The Computer Security Foundations Workshop III* [1], pages 32–41.

[24] P. Syverson. The use of logic in the analysis of cryptographic protocols. In *Proceedings of the 12th IEEE Symposium on Research in Security and Privacy*, pages 156–170, Oakland, California, May 20–22 1991. IEEE Computer Society Press.

[25] T.Y.C. Woo and S.S. Lam. Authentication for distributed systems. *Computer*, 25(1):39–52, January 1992.

[26] T.Y.C. Woo and S.S. Lam. A methodology for verifying authentication protocols. Technical report, Department of Computer Sciences, The University of Texas at Austin, 1993. In preparation.