Verifying Authentication Protocols: Methodology and Example*

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

Abstract

We present a new approach to the analysis of authentication protocols. The approach consists of several elements: a specification language for formally specifying authentication protocols, a semantic model for characterizing protocol executions, an assertion language for stating *secrecy* and *correspondence* properties, and procedures for verifying these properties. The main emphasis of this paper is on the assertion language, its semantics, and verification procedures. In particular, we present a set of *proof rules*. We also present an example to illustrate our approach.

1 Introduction

Authentication plays a foundational role in the security of distributed systems. In a distributed environment, authentication is typically carried out using *authentication protocols*. The primary goal of an authentication protocol is to establish identities of the entities participating in the protocol. Typical entities include users, workstations, processes and so on. We refer to these entities as *principals*. Many authentication protocols, however, also accomplish a secondary goal, namely, to distribute a new secret session key for future private communication among the principals.

The design of authentication protocols is highly error-prone, even more so than communication protocols. Indeed, many authentication protocols have been proposed and later found to contain subtle weaknesses or flaws [3, 6, 8, 12, 13]. Various formal methods have been proposed and applied to the analysis of communication protocols. Unfortunately, these methods cannot be directly applied to authentication protocols. This is because authentication protocols must operate correctly in a hostile environment where all kinds of attacks may be perpetuated by intruders (e.g., masquerade and replay attacks).

Our research is aimed at developing formal techniques that can be used to ascertain the "security" of an authentication protocol. This entails first understanding precisely what it means for an authentication protocol to be secure, and then developing a verification methodology based on this understanding. In our view, security is no more than a specialized notion of correctness, and verification of security is simply verification of this specialized notion of correctness.

Problems with Existing Approaches

Many approaches have been proposed for the analysis of authentication protocols [4, 6, 9, 10, 11, 12, 15]. However, they are not satisfactory for several reasons. First, most approaches do not clearly separate correctness and verification concerns. That is, the precise notion of correctness is not independently stated but rather is built into the verification procedure. This can hamper understanding as the correctness criteria must be extracted out from the verification procedure. Applicability is also restricted to cases where the built-in correctness notion coincides with the intended one. We believe that there should be a clear distinction between the two. In particular, correctness is a specification problem while verification is a procedure problem.

Second, the notion of correctness adopted is often limited and does not adequately capture the entire intuitive notion of correctness an authentication protocol designer has in mind. For example, the proposals in [9, 11] adopt *secrecy* as their main correctness criterion; while others [6, 10] specify correctness in terms of *state of belief* and make implicit the secrecy concerns.

*Research supported in part by NSA Computer Security University Research Program under contracts no. MDA 904-91-C7046 and MDA 904-93-C4089, and in part by National Science Foundation grant no. NCR-9004464.

0-8186-3670-X/93 $03.00 © 1993 IEEE

This discrepancy can lead to potential misinterpretations.

Third, the supposedly "formal" notion of correctness is often 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 [5]. The informal explanation was widely misunderstood and led directly to the criticism by Nessett in [14]. 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].

Fourth, the analysis in certain approaches is conducted at a very high level of abstraction. The results obtained from such analysis can be easily misinterpreted when trying to relate them to protocol executions. Examples include most of the modal logical approaches [4, 6, 10, 15]. We stress, however, that these approaches are certainly useful in their own right. But we also believe that it is important to be able to analyze a protocol at different levels of abstraction. Therefore, concrete and operational approaches should also be devised to complement these high-level approaches.

An Overview of our Approach

Our approach addresses these inadequacies. We view correctness and verification as two separate but related problems: Correctness is established by verification, while verification requires a precise notion of correctness. This separation of concerns allows us to first tackle the correctness problem and then focus on the verification problem.

To tackle the correctness problem, we first identify the basic security properties of an authentication protocol. Our research has thus far identified two such classes of properties, namely, *secrecy* and *correspondence*. These two classes of properties are distinct[1] and are sufficient for formalizing the typical high-level correctness concerns of authentication and key distribution. We have defined a formal assertion language for stating these two classes of properties. The formal semantics of this assertion language is specified using a semantic model we have developed for authentication protocols [17]. Roughly speaking, the model defines an operational semantics for authentication protocols. In particular, it characterizes all their executions.

For the verification problem, we investigate procedures for proving the two classes of properties. It turns out that secrecy properties can be simply viewed as invariance properties. Thus the standard invariance verification procedure can be used. On the other hand, the verification of correspondence properties is more involved; we make use of an axiomatic proof system. Our proof procedure has two steps: (1) Identification of a set of *primitive* correspondence properties.[2] These primitive correspondence properties must be justified either by the protocol specification or by appealing directly to the semantic model. (2) Application of a sequence of proof rules to infer the desired conclusion. Thus the main task in solving the verification problem is to identify an appropriate set of proof rules.

The major merit of our approach is its formality and concreteness. Our semantic model is based on states and transitions, which can be easily related to protocol implementations. Such an approach also resembles techniques used for analyzing communication protocols. Our assertion language allows users to specify correctness properties tailored to a specific protocol. The semantics of these correctness properties are completely formalized in the semantic model. Therefore, there is no ambiguity in interpreting the precise meaning of security. We believe that our work represents the first attempt to identify various classes of basic security properties, and to advocate a language-based approach to define the operational semantics of authentication protocols. Our proof rules are formal, and can be proved to be sound with respect to our semantic model.

The balance of this paper is organized as follows. In Section 2, we informally discuss the secrecy and correspondence properties. In Section 3, we present the syntax and semantics of our assertion language. The material here is a significant extension of a preliminary version given in [17]. For example, the language now includes infinitely disjunctive terms and two new kinds of assertions, namely, *restricted correspondence* and *equivalence*. This section makes heavy use of the semantic model developed in [17]. Due to length limitation, we are not able to include a summary of the model; we refer readers to that paper instead. In Section 4, we describe proof procedures for the correctness properties. In particular, we present a set of proof rules for correspondence properties. In Section 5, we work out an example in its entirety using our methodology. We start with a specification of the

[1] That is, an authentication protocol may satisfy a secrecy property while failing a correspondence property or vice versa.

[2] They play a role similar to that of axioms in a typical proof system.

protocol example; then we present a specification of correctness and finally the key steps in its verification. In Section 6, we conclude the paper with several observations. Proofs can be found in [16].

2 Two Basic Correctness Properties

In this section, we take an informal look at the two classes of basic correctness properties for authentication protocols. We guide our discussion with a simple authentication protocol. This protocol also serves as the example in Section 5. Due to space limitation, our discussion is necessarily brief. We encourage the readers to consult [17] for a more detailed discussion.

Before we proceed, we first review the notation generally used in the literature for specifying authentication protocols. This notation is sufficient for the informal discussion in this section. We adopt a more specialized notation with rigorous syntax in our approach, which was first introduced in [17].

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. □

Consider the following protocol, **P**: (n is a nonce)

(1) $P \rightarrow Q$: "I am P."
(2) $Q \rightarrow P$: n
(3) $P \rightarrow Q$: $\{P, Q, n\}_{k_{PA}}$
(4) $Q \rightarrow A$: $\{P, Q, n, \{P, Q, n\}_{k_{PA}}\}_{k_{QA}}$
(5) $A \rightarrow Q$: $\{P, Q, n\}_{k_{QA}}$

P is a one-way authentication protocol; it authenticates P to Q. It does not distribute any key.[3] P is the *initiator* of the protocol and also the *authenticated* principal, while Q is the *responder* of the protocol and also the *authenticating* principal. A is a distinguished principal who is universally trusted[4] by all principals in the system. In particular, it shares a secret key k_{PA} with each principal P in the system.

The central question we want to ask is whether **P** is correct. That is, does **P** perform its intended goal of authenticating P to Q? Informally, authentication can be understood as follows: Upon termination of protocol execution, the authenticating principal should be assured that it is "talking" to the principal it expects. In other words, the protocol should not succeed if the authenticated principal has not participated in the authentication exchange. We note that the notion of authenticating and authenticated principals are relative to a particular instance of authentication exchange. This is because multiple concurrent authentication exchanges are allowed, even between the same two principals. The steps in these different exchanges can be arbitrarily interleaved.

Clearly, the above understanding of correctness is highly imprecise. For example, the meaning of "termination" and "talking" is unclear. One way to clarify it is to use a *correspondence property*:[5]

> Every time Q receives a valid response in step (5), it must be the case that P has generated an authentication initiation request in step (1) earlier.

A correspondence property essentially relates the occurrence of a transition to an earlier occurrence of some other transition. The relation is *one-to-one*. Thus there is a correspondence between the occurrence of a transition and the occurrence of the transition it is related to.

In the above example, the correspondence is between the transition "Q receives a valid response in step (5)" and the transition "P sends an authentication initiation request in step (1)." Referring to the original informal statement of correctness, the correspondence property is basically refining "termination" to mean Q's receipt of a valid response in step (5), and "talking" to mean P's sending of an authentication initiation request.

Besides the obvious authentication concern, there is an often neglected secrecy concern that **P** should satisfy in order for it to be correct. Specifically, it must be the case that in executing **P**, the secret keys of each principal remain secret. This is a secrecy property.

[3] We have deliberately kept the example simple for ease of exposition. The example can be easily extended to include mutual authentication and key distribution. As we will see, secrecy is already a concern even in the absence of key distribution.

[4] The precise meaning of trust is difficult to formulate. In this paper, we take it to mean that A would faithfully follow its protocol and does not divulge any key to other principals.

[5] Since we have not yet introduced our assertion language, we use English here to state the property informally. The formal statement is given in Section 5.

Although this is somewhat trivial in the case of P, it must nevertheless be included as one of the correctness criteria.

Furthermore, for authentication protocols that perform key distribution, there is a secrecy property concerned with the secrecy of the keys being distributed. Essentially, the secrecy property specifies that keys destined to be shared among a set of principals should be known only to members of that set.

3 Assertion Language

We informally discussed the classes of correspondence and secrecy properties in Section 2. In this section, we present an assertion language for formally specifying these properties.

This section makes heavy use of the notation introduced in [17]. We provide a brief summary below.

Notation. Given a term γ, we denote by $var(\gamma)$ the set of variables that occur in γ. To make explicit the fact that a term contains certain variables, we sometimes denote a term as $\gamma(V)$, where V is a set of variables that occur in the term γ (among others). Thus, we always have $V \subseteq var(\gamma(V))$. When V is a singleton $\{x\}$, we simply write $\gamma(x)$.

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'. The application of a substitution θ to a term γ, denoted by $\gamma : \theta$, is defined as the concurrent replacement of all occurrences of x in γ 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.

An *execution* is a *finite* alternate sequence of global states and transitions. We generally write it in the form $s_1\beta_1 s_2\beta_2 \ldots s_n$, where s_i's are states and β_i's transitions. □

3.1 Correspondence

A global transition can be viewed as a ground term. For example, the communication transition $(\mathsf{P}.(1), \mathsf{Q}.(5), \mathsf{Comm}([\mathsf{N}]))$ can be viewed[6] as a term formed by the *tuple* operation on the terms P.(1), Q.(5) and Comm([N]). These terms can themselves be further broken down, e.g., Comm([N]) into the function symbol Comm and the term [N]. If we extend this term formation process to include variables as well as ground terms, we obtain the language of *transition terms*. Thus the general form of a transition term is

$$(x.\ell, cmd(t)) \quad \text{or} \quad (x.\ell, y.\ell', cmd(t))$$

where *cmd* is one of the command words allowed in a protocol statement. The left term corresponds to internal transitions while the right one corresponds to communication transitions. For compactness, the label variables (i.e., ℓ and ℓ' above) can be omitted when there is no ambiguity. We will omit the formal definition of this term language.

Starting with transition terms, we can form *compound transition terms* (or *compound term* in short). Let γ_i be transition terms, then the following is a compound term ($n \geq 1$):

$$\gamma_1 \mid \ldots \mid \gamma_n$$

The symbol "|" is read "or" and has similar meaning as logical disjunction in classical logic. We can further extend this to include an infinite number of disjunction. Thus the most general form of a compound term is the following:

$$|_{t \in T}\ \gamma_t$$

Here each transition term is assumed to be indexed by an index t in the set T of indices. We note that an index can itself be a term.

Let g be a global transition and γ be a transition term. We say $match(g, \gamma)$ holds if there exists a substitution θ such that $g = \gamma : \theta$. This can be easily extended to a compound term as follows: $match(g, |_{t \in T}\ \gamma_t)$ holds if there exists $t_c \in T$ such that $match(g, \gamma_{t_c})$ holds.

Terminology. Let $|_{t \in T}\ \gamma_t$ and $|_{t \in T'}\ \mu_t$ be two compound terms. We say $|_{t \in T}\ \gamma_t$ is *less general* than $|_{t \in T'}\ \mu_t$, denoted by $|_{t \in T}\ \gamma_t \leq |_{t \in T'}\ \mu_t$, if for all ground term g:

$$match(g, |_{t \in T}\ \gamma_t) \text{ implies } match(g, |_{t \in T'}\ \mu_t)$$

Note that $\leq$ induces a partial ordering on compound terms.

We say $|_{t \in T}\ \gamma_t$ and $|_{t \in T'}\ \mu_t$ are *disjoint* if there does not exist a ground term g such that

$$match(g, |_{t \in T}\ \gamma_t) \text{ and } match(g, |_{t \in T'}\ \mu_t)$$

Next, let Π be a set of executions. We say $|_{t \in T}\ \gamma_t$ is *replay-free relative to* Π if for all executions $\xi = s_1\beta_1 s_2\beta_2 \ldots s_n \in \Pi$:

[6] This represents a communication of term N from P to Q. (1) labels the sending step in P's protocol, while (5) labels the receiving step in Q's protocol.

$[i \neq j \wedge match(\beta_i, |_{t\in T}\ \gamma_t) \wedge match(\beta_j, |_{t\in T}\ \gamma_t)]$
implies $\beta_i \neq \beta_j$

□

There are two types of correspondence assertions: *general* and *restricted*. Their semantics are almost identical except that restricted correspondence relaxes the one-to-one correspondence requirement for multiple identical transitions. The main use of restricted correspondence is for transitions that may be subject to replay.

3.1.1 General Correspondence Assertions

A general correspondence assertion has the form

$$|_{t\in T}\ \gamma_t \hookrightarrow |_{t\in T'}\ \mu_t$$

The symbol $\hookrightarrow$ denotes the general correspondence relation. We refer to the left hand side of such an assertion as the *head* and the right hand side as the *body*.

We distinguish two cases in defining the semantics for a general correspondence assertion: Case 1. The variables occurring in the head and body are disjoint. Case 2. There are common variables occurring in head and body.

Formally, let

$$\rho \quad = \quad |_{t\in T}\ \gamma_t \hookrightarrow |_{t\in T'}\ \mu_t$$

be a general correspondence assertion.

Case 1. $var(|_{t\in T}\ \gamma_t) \cap var(|_{t\in T'}\ \mu_t) = \emptyset$.

Let $\xi = s_1\beta_1 s_2\beta_2 \ldots s_n$ be an execution. Define

$$I = \{i \mid match(\beta_i, |_{t\in T}\ \gamma_t)\}$$

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

- $f(i) < i$, and
- $match(\beta_{f(i)}, |_{t\in T'}\ \mu_t)$, and
- f is one-to-one.

Case 2. $var(|_{t\in T}\ \gamma_t) \cap var(|_{t\in T'}\ \mu_t) \neq \emptyset$.

We would view ρ as an abbreviation for the following (possibly infinite) set, $\mathcal{S}(\rho)$, of correspondence assertions:

$$\mathcal{S}(\rho) \quad = \quad \{\rho : \theta \mid \theta \text{ is a ground substitution over } var(|_{t\in T}\ \gamma_t) \cap var(|_{t\in T'}\ \mu_t)\}$$

Then ξ *satisfies* ρ if and only if ξ satisfies all assertions in $\mathcal{S}(\rho)$.

This definition is well-defined because the assertions in $\mathcal{S}(\rho)$ do not share common variables in their heads and bodies, and hence their satisfaction relation is provided by Case 1.

3.1.2 Restricted Correspondence Assertions

The syntax for a restricted correspondence assertion is identical to that of a general correspondence assertion, except the relation symbol is $\overset{r}{\hookrightarrow}$ instead. Their semantics are almost identical too, except that a (potentially) different set I' is used is in place of I in the definition in Case 1 above. The definition of I' is as follows:

$$I' = \begin{array}{l}\text{a maximal subset } S \text{ of } I \text{ such that} \\ \text{for all } i, j \in S : i \neq j \text{ implies } \beta_i \neq \beta_j\end{array}$$

Essentially, this condition says that exactly one transition would be included in I' if there are multiple identical transitions matching the head.

3.2 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.

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 a predefined GetSecret action. We refer readers to its precise formulation in [17].

The specific part consists of *secrecy assertions*. A secrecy assertion is specified using formulas similar to those used for *initial conditions* (see example in Section 5). The formulas however are interpreted in the *final* state of an execution, as opposed to the initial state for initial conditions. Again, we omit the definitions here and refer readers to [17] for the precise syntax and semantics.

3.3 Equivalence

Many proofs can be simplified if we define an additional class of assertions, namely, *equivalence* assertion. The general form of an equivalence assertion is

$$|_{t\in T}\ \gamma_t \equiv |_{t\in T'}\ \mu_t$$

Its semantics is as follows. Let $\xi = s_1\beta_1 s_2\beta_2 \ldots s_n$ be an execution and let ρ denote the above equivalence assertion. Then ξ *satisfies* ρ if the following holds:

$$\forall\, i : [match(\beta_i, |_{t\in T}\, \gamma_t) \quad \text{iff} \quad match(\beta_i, |_{t\in T'}\, \mu_t)]$$

The basic idea of an equivalence assertion is that as far as ξ is concerned, the two sides of the equivalence would match the same set of transitions.

3.4 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 $f \in \mathcal{C}$, for all execution $\xi \in \Pi$, ξ satisfies f, and
- for all secrecy assertion $f \in \mathcal{S}$, for all execution $\xi \in \Pi$, ξ satisfies f, and
- for all execution $\xi \in \Pi$, ξ satisfies the general secrecy condition.

□

4 Procedures and Rules

4.1 Proof Procedures

The procedure to establish a correspondence assertion is not unlike the typical proof procedure in logic. First, a set of *primitive* correspondence assertions is identified. These primitive correspondence assertions play a role similar to that of axioms in a typical proof system. In our approach, they are justified either directly by inspecting the protocol specification or from first principles using the semantic model. Then starting from these primitive correspondence assertions, new correspondence assertions can be derived by applying proof rules, until the desired conclusion is reached. The semantic meaning of such a proof is given by an *entailment* relation. We define entailment and present a set of proof rules for correspondence in the next subsection.

A secrecy assertion can be viewed as a specialized invariant. Thus the usual invariance verification procedure can be used. That is, to prove a secrecy assertion ρ, we must establish: (1) ρ is satisfied in all executions that contain only an initial state. Or equivalently, it is logically entailed by the initial conditions. (2) If ρ holds for an execution ξ, then for all global transition β enabled at the last state, s, of ξ, and $s \xrightarrow{\beta} s'$, then ρ holds for $\xi\beta s'$ as well. We note that β can be a transition of intruder Z. Thus establishing (2) may involve characterizing the set of transitions Z is capable of performing.

Verification of the general secrecy condition is more protocol dependent. Typically, it involves inspecting the protocol specification and observing that a secret is always *bound*[7] to a nonce, hence an old compromised secret would not be accepted in a current authentication exchange. We omit the details here due to space limitation.

Verification of an equivalence assertion can be syntactic (i.e., by inspecting the assertion itself), by inspecting the protocol specification, or by appealing to the semantic model. This is illustrated in the proof of the example in Section 5.

4.2 Proof Rules

Definition. (Entailment) Let Π be a set of executions, Σ a set of assertions, and ρ a correspondence assertion. Then Σ *entails* ρ *relative to* Π if for all executions $\xi \in \Pi$

$$[\forall\, \varphi \in \Sigma : \xi \text{ satisfies } \varphi] \quad \text{implies} \quad \xi \text{ satisfies } \rho$$

□

A *proof rule* is an ordered pair (Σ, ρ), where Σ is a set of assertions and ρ is an assertion.

Definition. (Soundness) Let $R = (\Sigma, \rho)$ be a proof rule. R is *sound* if for all sets of executions Π, Σ entails ρ relative to Π. □

We typically present a proof rule $R = (\Sigma, \rho)$ in the form

$$\frac{\rho_1, \rho_2, \cdots}{\rho}$$

where $\Sigma = \{\rho_1, \rho_2, \ldots\}$. We call $\rho_1, \rho_2, \ldots$ the *premises* of R, and ρ the *conclusion* of R.

A sound proof rule can be applied to all sets of executions. Often times, we are also interested in proof rules that apply only to sets of executions that satisfy certain extra conditions. We call them *conditional proof rules.* A conditional proof rule is specified by an ordered pair as in an ordinary proof rule together

[7] A typical example of such a "bonding" is if the key being distributed (i.e., the new *session* key) is always encrypted together with a nonce generated by the intended recipient.

with a statement of the conditions. Typically, it is presented as

$$\frac{\rho_1, \rho_2, \cdots}{\rho} \qquad (cond)$$

where *cond* denotes the conditions.

A conditional proof rule is said to be *sound* if it is sound in the sense of an ordinary proof rule when only sets of executions satisfying its conditions are considered.

Restriction Rule

$$\frac{|_{t\in T}\ \gamma_t \hookrightarrow |_{t\in T'}\ \mu_t}{|_{t\in T}\ \gamma_t \overset{r}{\hookrightarrow} |_{t\in T'}\ \mu_t}$$

Promotion Rule

$$\frac{|_{t\in T}\ \gamma_t \overset{r}{\hookrightarrow} |_{t\in T'}\ \mu_t}{|_{t\in T}\ \gamma_t \hookrightarrow |_{t\in T'}\ \mu_t} \qquad (|_{t\in T}\ \gamma_t \text{ is replay-free})$$

Substitution Rule

Suppose $var(|_{t\in T}\ \gamma_t) \cap var(|_{t\in T'}\ \mu_t) = var(|_{t\in T''}\ \nu_t) \cap var(|_{t\in T'}\ \mu_t)$. Then

$$\frac{|_{t\in T}\ \gamma_t \hookrightarrow |_{t\in T'}\ \mu_t, \quad |_{t\in T}\ \gamma_t \equiv |_{t\in T''}\ \nu_t}{|_{t\in T''}\ \nu_t \hookrightarrow |_{t\in T'}\ \mu_t}$$

A similar rule holds for substitution in the body. The Substitution Rule is often used together with the following rule for equivalence assertions:

$$\frac{|_{t\in T}\ \gamma_t \equiv |_{t\in T'}\ \mu_t}{|_{t\in T}\ \gamma_t \ \Big|\ |_{t\in T''}\ \nu_t \equiv |_{t\in T'}\ \mu_t\ \Big|\ |_{t\in T''}\ \nu_t}$$

Abstraction Rule

Suppose $var(|_{t\in T}\ \gamma_t) \cap var(|_{t\in T'}\ \mu_t) = var(|_{t\in T}\ \gamma_t) \cap var(|_{t\in T''}\ \nu_t)$ and $|_{t\in T'}\ \mu_t \leq |_{t\in T''}\ \nu_t$. Then

$$\frac{|_{t\in T}\ \gamma_t \hookrightarrow |_{t\in T'}\ \mu_t}{|_{t\in T}\ \gamma_t \hookrightarrow |_{t\in T''}\ \nu_t}$$

Weakening Rule

Let θ be a substitution such that $domain(\theta) \subseteq var(|_{t\in T}\ \gamma_t) \cap var(|_{t\in T'}\ \mu_t)$ and $range(\theta) \cap var(|_{t\in T}\ \gamma_t) \cap var(|_{t\in T'}\ \mu_t) = \emptyset$. Then

$$\frac{|_{t\in T}\ \gamma_t \hookrightarrow |_{t\in T'}\ \mu_t}{(|_{t\in T}\ \gamma_t) : \theta \hookrightarrow (|_{t\in T'}\ \mu_t) : \theta}$$

Disjunction Rule

Let V be a set of variables, T a set of indices, and $\{\gamma_t(V) \mid t \in T\}$ a set of terms.

Suppose the following holds for terms in T: Let g, g' be ground terms and θ a ground substitution over V.

$[match(g, \gamma_t(V) : \theta) \wedge match(g', \gamma_{t'}(V) : \theta)]$ implies $t = t'$

Then

$$\frac{\forall\, t \in T : \gamma_t(V) \hookrightarrow \mu(V)}{|_{t\in T}\ \gamma_t(V) \hookrightarrow \mu(V)}$$

Transitivity Rule

Suppose $|_{t\in T'}\ \mu_t$ and $|_{t\in T''}\ \nu_t$ are disjoint and $T'' \subseteq T'$. Then

$$\frac{|_{t\in T}\ \gamma_t \hookrightarrow |_{t\in T'}\ \mu_t, \quad |_{t\in T''}\ \mu_t \hookrightarrow |_{t\in T''}\ \nu_t}{|_{t\in T}\ \gamma_t \hookrightarrow |_{t\in T'-T''}\ \mu_t \ \Big|\ |_{t\in T''}\ \nu_t}$$

A similar rule can be formulated for the head. Also, by using the Restriction Rule, general and restricted correspondence assertions can be mixed together in a transitivity inference. The result is a restricted correspondence assertion, if there is at least one restricted correspondence assertion in the premises.

Recursion Rule

Let t_0 be a term, and $s(x)$ a term that contains a variable x. We denote the application of a substitution $\{t/x\}$ by $s(t)$. Let T be the set of term $\{t_0, s(t_0), s(s(t_0)), \ldots\}$. Then, for $t_c \in T$

$$\frac{\forall\, t \in T : \gamma_t \hookrightarrow \mu_{s(t)} \mid \gamma_{s(t)}}{\gamma_{t_c} \hookrightarrow |_{n>0}\ \mu_{s^n(t_c)}}$$

where $s^n(t) = \underbrace{s(s(\ldots s}_{n}(t)\ldots))$.

5 An Example

In this section, we apply our methodology to protocol **P** presented in Section 2. Specifically, we first present a specification of the protocol, then a specification of correctness and finally an outline of the verification. Our intent is to provide a general flavor of our approach and to illustrate the use of our protocol specification and assertion languages, our semantic model and proof rules.

Notation. Let SYS denote the set of all principals in the system. We use lower case variables (e.g.,

Initial Conditions:

$\forall x, y : x$ has y
$\forall x, y : x$ has $k_{yA} \Leftrightarrow [x = y \vee x = A]$
Z has $\mathbf{X} \Leftrightarrow [\mathbf{X} \in SYS \vee \mathbf{X} = k_{ZA}]$

Initiator (i) Protocol:

(1) BeginInit (r)
(2) Send (r, ["I am i."])
(3) Receive (r, [$\mathbf{X}$])
(4) Send (r, [$\{i, r, \mathbf{X}\}_{k_{iA}}$])
(5) EndInit (r)

Responder (r) Protocol:

(1) BeginRespond (i)
(2) Receive (i, ["I am i."])
(3) NewNonce (n)
(4) Send (i, n)
(5) Receive (i, [$\mathbf{X}$])
(6) Send (A, [$\{i, r, n, \mathbf{X}\}_{k_{rA}}$])
(7) Receive (A, [$\{i, r, n\}_{k_{rA}}$])
(8) EndRespond (i)

Server A Protocol:

(1) Receive (r, [$\{i, r, \mathbf{X}, \{i, r, \mathbf{X}\}_{k_{iA}}\}_{k_{rA}}$])
(2) Send (r, [$\{i, r, \mathbf{X}\}_{k_{rA}}$])

Figure 1: Protocol **P**

i, r, p, n) to range over primitive terms. In particular, i, r and p are assumed to range over names of legitimate principals in the system, i.e., SYS$-\{$A, Z$\}$, while x, y range over any principal in SYS. n ranges over the set of nonce constants. Bold face variables (e.g., $\mathbf{X}$) range over arbitrary non-null terms. □

5.1 Protocol Specification

We give a formal specification of **P** in our notation in Figure 1.[8] As is shown there, the original "global" protocol description given in Section 2 is formally represented by three separate local protocols, one for the initiator, one for the responder and one for the trusted principal.

Note that both the initiator and responder protocols can be instantiated by arbitrary principals, and are called *instantiable* protocols; while the server protocol can only be performed by the constant principal A, who represents a trusted server, and is called a *fixed* protocol.

The protocol specification also makes explicit the initial conditions. The first initial condition specifies that each principal in the system knows (and hence can use) the names of all other principals in the system. The second condition specifies that the secret key k_{xA} of principal x is shared only between itself and A. The third condition specifies that Z knows only the names of other principals and its secret key initially. This is not strictly necessary nor is it particularly restrictive, but its adoption would considerably simplify the statements of certain assertions later.

5.2 Correctness Specification

The minimal correspondence assertion one would expect any one-way authentication protocol to satisfy is:

$$(r, \text{EndRespond}(i)) \hookrightarrow (i, \text{BeginInit}(r)) \qquad \text{(C1)}$$

This says that every time a responder r finishes the responder protocol with i as the initiator, it must be the case that i have actually initiated the authentication earlier.[9]

In the case of **P**, we can in fact specify a stronger assertion:

$$(r, \text{EndRespond}(i)) \hookrightarrow (i, y, \text{Comm}([\{i, r, n\}_{k_{iA}}])) \qquad \text{(C2)}$$

(C2) can in turn be strengthened to (C3) as follows. This is the main correspondence property for **P**.

$$(x, r.(7), \text{Comm}([\{i, r, n\}_{k_{rA}}])) \hookrightarrow (i, y, \text{Comm}([\{i, r, n\}_{k_{iA}}])) \qquad \text{(C3)}$$

The constant label (7) is needed to distinguish the two Receive statements in step (5) and step (7) in the responder protocol. The correspondence is only required for the receive in step (7). (C3) is a formal statement of the correspondence property informally presented in Section 2.

For secrecy, there is only one secrecy assertion:

$$x \text{ has } k_{pA} \Leftrightarrow [x = p \vee x = A] \qquad \text{(S)}$$

This is almost identical to the initial condition, except that we are not concerned about whether Z's secret key is divulged or not.

Thus the correctness specification is $(\{(C3)\}, \{(S)\})$.

[8] For brevity, we abbreviate $key(p, A)$ to k_{pA}.

[9] For mutual authentication, an additional correspondence

$$(i, \text{EndInit}(r)) \hookrightarrow (r, \text{BeginRespond}(i))$$

should be added.

5.3 Proof of Correctness

We note that the general secrecy condition is vacuously true for **P**, as it does not generate any new secret. Proof of the secrecy assertion (S) is based on invariance and is omitted for brevity (see [16]).

To prove correspondence, we first define two sets of terms. These sets would be used in the statement of intermediate assertions that are needed in the proofs.

The first set, referred to as $\mathcal{L}$, is defined as follows. Let

$$\begin{aligned} \mathbf{L}_0 &= \mathbf{X} \\ \mathbf{L}_{n+1} &= \mathbf{L}_n, \{i, r, \mathbf{L}_n\}_{k_{i\mathsf{A}}} \end{aligned}$$

and

$$\mathcal{L} = \{\mathbf{L}_n \mid n \geq 0\}$$

Thus, for examples, $\mathbf{X}$ and $\mathbf{X}, \{i, r, \mathbf{X}\}_{k_{i\mathsf{A}}}$ are elements of $\mathcal{L}$.

The second set, referred to as $\mathcal{L}'$, is defined as follows. First let

$$\begin{aligned} \ell_0 &= \mathbf{X} \\ \ell_{n+1} &= \{i, r, \ell_0, \ldots, \ell_n\}_{k_{i\mathsf{A}}} \end{aligned}$$

Then define

$$\mathbf{L}'_n = \ell_0, \ldots, \ell_n$$

and

$$\mathcal{L}' = \{\mathbf{L}'_n \mid n \geq 0\}$$

If we take a closer look at the two sets above, they are in fact parameterized by the variable $\mathbf{X}$. Thus, to be more accurate, we will refer to them as $\mathcal{L}_{\mathbf{X}}$ and $\mathcal{L}'_{\mathbf{X}}$ respectively.

For brevity, we present only the key lemmas below. Their proofs are contained in [16].

Lemma A. $\mathcal{L}_{\mathbf{X}} = \mathcal{L}'_{\mathbf{X}}$. □

Let $\mathcal{L}_n$ denote the instantiation of $\mathcal{L}_{\mathbf{X}}$ with n.

Lemma B. The terms in $\mathcal{L}_n$ are pairwise disjoint. □

Note that this is not true for $\mathcal{L}_{\mathbf{X}}$.

Lemma C.

$$(\mathsf{Z}, x, \mathsf{Comm}([\{\mathbf{X}\}_{k_{p\mathsf{A}}}])) \overset{r}{\hookrightarrow} (y, \mathsf{Z}, \mathsf{Comm}([\{\mathbf{X}\}_{k_{p\mathsf{A}}}]))$$

□

Lemma C says that Z is not able to produce terms of the form $\{\mathbf{X}\}_{k_{p\mathsf{A}}}$. Thus if it sends a term of this form out, it must be the case that it has gotten it earlier from some other principal. This is a restricted correspondence rather than a general correspondence because Z may replay a message multiple times.

Lemma D.

$$(x, r.(7), \mathsf{Comm}([\{i, r, n\}_{k_{r\mathsf{A}}}])) \hookrightarrow |_{\mathbf{L} \in \mathcal{L}_n} (r, y, \mathsf{Comm}([\{i, r, \mathbf{L}, \{i, r, \mathbf{L}\}_{k_{i\mathsf{A}}}\}_{k_{r\mathsf{A}}}]))$$

□

Lemma D says that whenever a responder r receives a message of the form $\{i, r, n\}_{k_{r\mathsf{A}}}$ in step (7) of its protocol, it must be the case that r has sent out earlier a message matching one of the forms $\{i, r, \mathbf{L}, \{i, r, \mathbf{L}\}_{k_{i\mathsf{A}}}\}_{k_{r\mathsf{A}}}$, where $\mathbf{L} \in \mathcal{L}_n$. Since there are only two Send in the responder protocol, and the first Send in step (4) always sends out a primitive terms, only the Send in step (6) could have matched the body of the assertion.

Lemma E. Let $\mathbf{L} \in \mathcal{L}_n$. Then

$$(r, x, \mathsf{Comm}([\{i, r, \mathbf{L}, \{i, r, \mathbf{L}\}_{k_{i\mathsf{A}}}\}_{k_{r\mathsf{A}}}])) \hookrightarrow (i, y, \mathsf{Comm}([\{i, r, n\}_{k_{i\mathsf{A}}}]))$$

□

Lemma E says that whenever a responder r sends out a message of the form $\{i, r, \mathbf{L}, \{i, r, \mathbf{L}\}_{k_{i\mathsf{A}}}\}_{k_{r\mathsf{A}}}$, for some $\mathbf{L} \in \mathcal{L}_n$, it must be the case that i has sent out earlier a message of the form $\{i, r, n\}_{k_{i\mathsf{A}}}$. Since i sends out a string in its first Send, only its second Send in step (4) can match the body of the assertion.

Proof of (C3)

From Lemma E and Disjunction Rule, we have

$$|_{\mathbf{L} \in \mathcal{L}_n} (r, x, \mathsf{Comm}([\{i, r, \mathbf{L}, \{i, r, \mathbf{L}\}_{k_{i\mathsf{A}}}\}_{k_{r\mathsf{A}}}])) \hookrightarrow (i, y, \mathsf{Comm}([\{i, r, n\}_{k_{i\mathsf{A}}}]))$$

Then (C3) follows from Lemma D and above using the Transitivity Rule. □

Proof of (C2)

From the sequentiality of the responder protocol, we have

$$(r, \mathsf{EndRespond}(i)) \hookrightarrow (x, r, \mathsf{Comm}([\{i, r, n\}_{k_{i\mathsf{A}}}]))$$

Then from this and (C3), we obtain (C2) by the Transitivity Rule. □

6 Conclusion

We have presented a new approach to the analysis of authentication protocols. A key principle of our approach is the separation of correctness and verification. We have defined a specification language

for formally specifying authentication protocols and an assertion language for stating correctness requirements. The semantics of each language is rigorously defined using the semantic model developed in [17]. We have also discussed procedures for verifying the two classes of correctness properties.

We have presented an example to illustrate most of the elements in our approach. We start from a protocol specification, and then present a correctness specification and its verification. Although some of the proofs may appear mathematical rather than formal, most of them could be refined to the lowest level of details.

Our approach focuses on characterizing the positive aspects of a protocol, i.e., what properties a correct execution would have, while most other approaches tend to focus on the negative aspects, i.e., what an intruder cannot obtain or do. In this sense, our approach is similar to the BAN logic approach [6] and to conventional program verification [7].

We have also successfully used our approach to uncover errors in some protocols that we thought were correct. Typically, this arises in an attempted verification of an assertion. The failed proof usually suggests a counterexample.

Our approach is not amenable to full automation. This is evident from the proof in Section 5, where the equivalence of two sets (which is a potentially intractable problem) is established. However, we believe that it does fit well in a user-guided proof checking system, as many of the primitive assertions can be verified quite easily by inspecting the protocol specification or by using the semantic model.

We are currently exploring a more general syntax for protocol specification. The variables in our specification act more as a "template" than a value-holding variable in conventional programs. It appears that an accurate modeling of timestamp authentication protocols may require value-holding variables. We are also considering the addition of *conditionals* (i.e., if-then-else statements) to the language.

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] 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 [6].

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

[7] K.M. Chandy and J. Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, Reading, Massachusetts, 1988.

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

[9] 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.

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

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

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

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

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

[16] 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.

[17] T.Y.C. Woo and S.S. Lam. A semantic model for authentication protocols. In *Proceedings of the 14th IEEE Symposium on Research in Security and Privacy*, pages 178–194, Oakland, California, May 24–26 1993. IEEE Computer Society Press.