A Logic of Authentication

MICHAEL BURROWS and MARTÍN ABADI
Digital Equipment Corporation
and
ROGER NEEDHAM
University of Cambridge Computer Laboratory

Authentication protocols are the basis of security in many distributed systems, and it is therefore essential to ensure that these protocols function correctly. Unfortunately, their design has been extremely error prone. Most of the protocols found in the literature contain redundancies or security flaws. A simple logic has allowed us to describe the beliefs of trustworthy parties involved in authentication protocols and the evolution of these beliefs as a consequence of communication. We have been able to explain a variety of authentication protocols formally, to discover subtleties and errors in them, and to suggest improvements. In this paper we present the logic and then give the results of our analysis of four published protocols, chosen either because of their practical importance or because they serve to illustrate our method.

Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: General—*security and protection*; C.2.2 [**Computer-Communication Networks**]: Network Protocols—*protocol verification*; D.4.6 [**Operating Systems**]: Security and Protection—*authentication*; *cryptographic controls*; *verification*; E.3 [**Data**]: Data Encryption—*public key cryptosystems*; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs

General Terms: Security, Theory, Verification

Additional Key Words and Phrases: Authentication protocols, cryptographic protocol, key distribution protocols, logics of knowledge and belief, Needham–Schroeder, X.509

1. INTRODUCTION

Authentication protocols are the basis of security in many distributed systems, and it is therefore essential to ensure that these protocols function correctly [15]. Unfortunately, their design has been extremely error prone. Although authentication protocols typically have few messages, the composition of each message can be subtle, and the interactions between the messages can be complex.

This paper was nominated for publication in *TOCS* by the Program Committee for the ACM SIGOPS Symposium on Operating Systems Principles, December 1989.
The three authors completed part of this work at Digital Equipment Corporation and part at the University of Cambridge.
Authors' addresses: M. Burrows and M. Abadi, Systems Research Center, Digital Equipment Corporation, 130 Lytton Avenue, Palo Alto, CA 94301; and R. Needham, University of Cambridge Computer Laboratory, Corn Exchange Street, Cambridge CB2 3QG, U.K.

Moreover, protocol designers often misunderstand the available techniques, copying features from existing protocols inappropriately. As a result, many of the protocols found in the literature contain redundancies or security flaws. To add to the confusion, protocols use different cryptosystems (e.g., [14], [17]) and cater for a wide range of applications; it is seldom clear how these protocols compare in the guarantees they offer.

The goal of authentication can be stated rather simply, though informally and imprecisely. After authentication, two *principals* (people, computers, services) should be entitled to believe that they are communicating with each other and not with intruders. In this paper we define a logic of authentication to express such beliefs precisely and to capture the reasoning that leads to them. These are examples of questions that we would like to be able to answer with the help of formal methods:

—Does this protocol work? Can it be made to work?
—Exactly what does this protocol achieve?
—Does this protocol need more assumptions than another protocol?
—Does this protocol do anything unnecessary?

In later sections we show how the logic has enabled us to answer these questions for a number of published protocols. It is worth noting that we have not tried to answer some other questions. Since we operate at an abstract level, we do not consider errors introduced by concrete implementations of a protocol, errors such as deadlocks, or even inappropriate use of cryptosystems (as in [19]). Furthermore, although we allow for the possibility of hostile intruders, there is no attempt to deal with the authentication of an untrustworthy principal, or to detect weaknesses of encryption schemes or unauthorized release of secrets (as in [7] and [12]). We focus on the beliefs of trustworthy parties involved in the protocols and on the evolution of these beliefs as a consequence of communication. This kind of study seems to be one of the most needed by current protocol designers; we have often uncovered subtleties and suggested improvements in existing protocols. We hope that protocol designers will build on our techniques to suit their specific needs.

In an earlier paper [2], we developed a fragment of the logic and demonstrated its ability to describe two authentication protocols: the well-known Needham–Schroeder protocol [15] and the Otway–Rees protocol [16]. The main emphasis there was on testing the soundness and viability of the approach. We have since extended the logic to explain a much wider range of protocols and techniques. In this paper we present the logic (in Section 2) and then use it to analyze four protocols, chosen either because of their practical importance or because they serve to illustrate our method. A more extensive report includes all the material of both papers, and we refer the interested reader to it for further examples and details on the logic and its semantics [3].

2. THE FORMALISM

In this section we describe the syntax and semantics of our logic, its rules, and the transformations that we apply to protocols before their formal analysis.

2.1 Basic Notation

Our formalism is built on a many-sorted model logic. In the logic we distinguish several sorts of objects: principals, encryption keys, and formulas (also called statements). We identify messages with statements in the logic. Typically, the symbols A, B, and S denote specific principals; K_{ab}, K_{as}, and K_{bs} denote specific shared keys; K_a, K_b, and K_s denote specific public keys, and K_a^{-1}, K_b^{-1}, and K_s^{-1} denote the corresponding secret keys; and N_a, N_b, and N_c denote specific statements. The symbols P, Q, and R range over principals; X and Y range over statements; and K ranges over encryption keys.

The only propositional connective is conjunction, denoted by a comma. Throughout, we treat conjunctions as sets and take for granted properties such as associativity and commutativity. In addition to conjunction, we use the following constructs:

—P **believes** X: P *believes* X, or P would be entitled to believe X. In particular, the principal P may act as though X is true. This construct is central to the logic.

—P **sees** X: P *sees* X. Someone has sent a message containing X to P, who can read and repeat X (possibly after doing some decryption).

—P **said** X: P *once said* X. The principal P at some time sent a message including the statement X. It is not known whether the message was sent long ago or during the current run of the protocol, but it is known that P believed X then.

—P **controls** X: P has *jurisdiction* over X. The principal P is an authority on X and should be trusted on this matter. For example, a server is often trusted to generate encryption keys properly. This may be expressed by the assumption that the principals believe that the server has jurisdiction over statements about the quality of keys.

—**fresh**(X): The formula X is *fresh*; that is, X has not been sent in a message at any time before the current run of the protocol. This is usually true for *nonces*, that is, expressions invented for the purpose of being fresh. Nonces commonly include a timestamp or a number that is used only once.

—$P \overset{K}{\leftrightarrow} Q$: P and Q may use the *shared key* K to communicate. The key K is good, in that it will never be discovered by any principal except P or Q, or a principal trusted by either P or Q.

—$\overset{K}{\mapsto} P$: P has K as a *public key*. The matching *secret key* (denoted K^{-1}) will never be discovered by any principal except P or a principal trusted by P.

—$P \overset{X}{=} Q$: The formula X is a *secret* known only to P and Q, and possibly to principals trusted by them. Only P and Q may use X to prove their identities to one another. An example of a secret is a password.

—$\{X\}_K$: This represents the formula X encrypted under the key K. Formally, $\{X\}_K$ is a convenient abbreviation for an expression of the form $\{X\}_K$ *from* P. We make the realistic assumption that each principal is able to recognize and ignore his own messages; the originator of each message is mentioned for this purpose.

—$\langle X \rangle_Y$: This represents X combined with the formula Y; it is intended that Y be a secret and that its presence prove the identity of whoever utters $\langle X \rangle_Y$. In implementations, X is simply concatenated with the password Y. Our notation highlights that Y plays a special role, as proof of origin for X, in much the same way as an encryption key.

(A different notation is used for some of these constructs in other works on the logic. The current notation is more verbose, but perhaps more readable and thus more appropriate for a paper with few formulas.)

2.2 Logical Postulates

In the study of authentication, we are concerned with the distinction between two epochs: the *past* and the *present*. The present epoch begins at the start of the particular run of the protocol under consideration. All messages sent before this time are considered to be in the past, and the authentication protocol should be careful to prevent any such messages from being accepted as recent. All beliefs held in the present are stable for the entirety of the protocol run. However, beliefs held in the past are not necessarily carried forward into the present. The simple division of time into past and present suffices for our purposes and has greatly increased the ease with which the logic can be manipulated.

It is assumed that encryption guarantees that each encrypted section cannot be altered or pieced together from smaller encrypted sections. If two separate encrypted sections are included in one message, we treat them as though they arrived in separate messages. A message cannot be understood by a principal who does not know the key (or, in the case of public-key cryptography, by a principal who does not know the inverse of the key); the key cannot be deduced from the encrypted message. Each encrypted message contains sufficient redundancy to allow a principal who decrypts it to verify that he has used the right key. In addition, messages contain sufficient information for a principal to detect (and ignore) his own messages.

After these informal preliminaries, we are now ready for discussing the main logical postulates that we use in proofs.

(1) The *message-meaning* rules concern the interpretation of messages. Two of the three concern the interpretation of encrypted messages, and the third concerns the interpretation of messages with secrets. They all explain how to derive beliefs about the origin of messages.

For shared keys, we postulate

$$\frac{P \textbf{ believes } Q \overset{K}{\leftrightarrow} P, \quad P \textbf{ sees } \{X\}_K}{P \textbf{ believes } Q \textbf{ said } X}.$$

That is, if P believes that the key K is shared with Q and sees X encrypted under K, then P believes that Q once said X. For this rule to be sound, we must guarantee that P did not send the message himself; it suffices to recall that $\{X\}_K$ stands for a formula of the form $\{X\}_K$ *from* R, and to require that $R \neq P$. Similarly,

for public keys, we postulate

$$\frac{P \textbf{ believes } \overset{K}{\leftrightarrow} Q, \quad P \textbf{ sees } \{X\}_{K^{-1}}}{P \textbf{ believes } Q \textbf{ said } X}.$$

For shared secrets, we postulate

$$\frac{P \textbf{ believes } \quad Q \overset{Y}{\rightleftharpoons} P, \quad P \textbf{ sees } \langle X \rangle_Y}{P \textbf{ believes } Q \textbf{ said } X}.$$

That is, if P believes that the secret Y is shared with Q and sees $\langle X \rangle_Y$, then P believes that Q once said X. This postulate is sound because the rules for **sees** (given below) guarantee that $\langle X \rangle_Y$ was not just uttered by P himself.

(2) The *nonce-verification* rule expresses the check that a message is recent and, hence, that the sender still believes in it:

$$\frac{P \textbf{ believes fresh}(X), \quad P \textbf{ believes } Q \textbf{ said } X}{P \textbf{ believes } Q \textbf{ believes } X}.$$

That is, if P believes that X could have been uttered only recently (in the present) and that Q once said X (either in the past or in the present), then P believes that Q believes X. For the sake of simplicity, X must be "cleartext"; that is, it should not include any subformula of the form $\{Y\}_K$.

(3) The *jurisdiction* rule states that if P believes that Q has jurisdiction over X then P trusts Q on the truth of X:

$$\frac{P \textbf{ believes } Q \textbf{ controls } X, \quad P \textbf{ believes } Q \textbf{ believes } X}{P \textbf{ believes } X}.$$

(4) If a principal sees a formula, then he also sees its components, provided he knows the necessary keys:

$$\frac{P \textbf{ sees } (X, Y)}{P \textbf{ sees } X}, \qquad \frac{P \textbf{ sees } \langle X \rangle_Y}{P \textbf{ sees } X}, \qquad \frac{P \textbf{ believes } Q \overset{K}{\leftrightarrow} P, P \textbf{ sees } \{X\}_K}{P \textbf{ sees } X},$$

$$\frac{P \textbf{ believes } \overset{K}{\mapsto} P, P \textbf{ sees } \{X\}_K}{P \textbf{ sees } X}, \qquad \frac{P \textbf{ believes } \overset{K}{\mapsto} Q, P \textbf{ sees } \{X\}_{K^{-1}}}{P \textbf{ sees } X}.$$

Recall that $\{X\}_K$ stands for a formula of the form $\{X\}_K$ *from* R. As a side condition, it is required that $R \neq P$; that is, $\{X\}_K$ is not from P himself. A similar condition applies to $\{X\}_{K^{-1}}$.

The fourth rule is justified by the implicit assumption that if P believes that K is his public key then P knows the corresponding secret key, K^{-1}.

Note that if P **sees** X and P **sees** Y it does *not* follow that P **sees** (X, Y), since this means that X and Y were uttered at the same time.

(5) If one part of a formula is fresh, then the entire formula must also be fresh:

$$\frac{P \textbf{ believes fresh}(X)}{P \textbf{ believes fresh}(X, Y)}.$$

2.3 On Quantifiers in Delegations

Delegation statements typically mention one or more variables. For example, principal A may let the server S generate an arbitrary shared key for A and B. We can express this as follows:

$$A \textbf{ believes } S \textbf{ controls } A \overset{K}{\leftrightarrow} B$$

Here the key K is universally quantified, and we can make explicit this quantification by writing

$$A \textbf{ believes } \forall K.(S \textbf{ controls } A \overset{K}{\leftrightarrow} B)$$

For complex delegation statements, it is generally necessary to write quantifiers explicitly in order to avoid ambiguities. For example, the reader can verify that the two formulas

$$A \textbf{ believes } \forall K.(S \textbf{ controls } B \textbf{ controls } A \overset{K}{\leftrightarrow} B)$$
$$A \textbf{ believes } S \textbf{ controls } \forall K.(B \textbf{ controls } A \overset{K}{\leftrightarrow} B)$$

convey different meanings.

In our earlier work on the logic, this need was not recognized, and in fact, it does not arise in any of the examples treated here (there are no nested jurisdiction statements). Therefore, we leave quantifiers implicit in this paper.

All we use is the ability to instantiate variables in jurisdiction statements, as reflected by the rule

$$\frac{P \textbf{ believes } \forall V_1 \cdots V_n.(Q \textbf{ controls } X)}{P \textbf{ believes } Q' \textbf{ controls } X'},$$

where Q' **controls** X' is the result of simultaneously instantiating the variables $V_1, \ldots, V_n$ in Q **controls** X. Our formal manipulation of quantifiers is thus quite straightforward.

2.4 Idealized Protocols

In the literature, authentication protocols are described by listing their messages; each message is typically written in the form

$$P \rightarrow Q\text{: } message.$$

This denotes that the principal P sends *message* to the principal Q. The message is presented in an informal notation designed to suggest the bit-string that a concrete implementation would use. This presentation is often ambiguous and not an appropriate basis for our formal analysis.

Therefore, we transform each protocol step into an idealized form. A message in the idealized protocol is a formula. For instance, the protocol step

$$A \rightarrow B\text{: } \{A, K_{ab}\}_{K_{bs}}$$

may tell B, who knows the key K_{bs}, that K_{ab} is a key to communicate with A. This step should then be idealized as

$$A \rightarrow B\text{: } \{A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}.$$

The idealized protocols of the examples given below do not include cleartext message parts; idealized messages are of the form $\{X_1\}_{K_1}, \ldots, \{X_n\}_{K_n}$. We have

omitted cleartext communication simply because it can be forged, and so its contribution to an authentication protocol is mostly one of providing hints as to what might be placed in encrypted messages.

We view the idealized protocols as clearer and more complete specifications than the traditional descriptions found in the literature, which we view merely as implementation-dependent encodings of the protocols. Therefore, we recommend the use of the idealized forms when generating and describing protocols. Though not entirely trivial, deriving a practical encoding from an idealized protocol is far less time consuming and error prone than understanding the meaning of a particular informal encoding.

Nevertheless, in order to study protocols from the existing literature, we must first generate idealized forms for each protocol. Simple guidelines control what transformations are possible, and these help in determining the idealized form for a particular protocol step. Roughly, a real message m can be interpreted as a formula X if whenever the recipient gets m he may deduce that the sender must have believed X when he sent m. Real nonces are transformed into arbitrary new formulas; throughout, we assume that the sender believes these formulas. The notation $\langle X \rangle_Y$, which denotes the use of Y as a secret, can be introduced only when the secret is intended as a proof of identity. Most importantly, for the sake of soundness, we always want to guarantee that each principal believes the formulas that he generates as messages. These simple guidelines suffice for our purposes, but further work on formal transformation rules might be useful.

2.5 Protocol Analysis

In order to analyze idealized protocols, we annotate them with logical formulas, much as in a proof in Hoare logic [10]. We write formulas before the first message and after each message. The main rules for deriving legal annotations are

—If X holds before the message $P \rightarrow Q$: Y then both X and Q **sees** Y hold afterwards; and

—if Y can be derived from X by the logical postulates then Y holds whenever X holds.

An annotation of a protocol is like a sequence of comments about the beliefs of principals and what they see in the course of authentication. In particular, the formula before the first message represents the beliefs of the principals at the start of the protocol. Step by step, we can follow the evolution from the initial beliefs to the final ones—from the original assumptions to the conclusions.

3. THE GOALS OF AUTHENTICATION, FORMALIZED

Initial assumptions must invariably be made to guarantee the success of each protocol. Typically, the assumptions state what keys are initially shared between the principals, which principals have generated fresh nonces, and which principals are trusted in certain ways. In most cases, the assumptions are standard and obvious for the type of protocol being considered. Once all the assumptions have been written, the verification of a protocol amounts to proving that some formulas hold as conclusions.

There is room for debate about what should be the goals of authentication protocols that these conclusions describe. Often authentication is a precursor to some communication protected by a shared session key, so we might desire conclusions that describe the situation at the start of such a communication. Thus, we might deem that authentication is complete between A and B if there is a K such that

$$A \textbf{ believes } A \overset{K}{\leftrightarrow} B, \qquad B \textbf{ believes } A \overset{K}{\leftrightarrow} B$$

Some protocols achieve more than this, as in the following example:

$$A \textbf{ believes } B \textbf{ believes } A \overset{K}{\leftrightarrow} B, \qquad B \textbf{ believes } A \textbf{ believes } A \overset{K}{\leftrightarrow} B$$

Other protocols attain only weaker final states, such as A **believes** B **believes** X, for some X, which reflects only that A believes that B has recently sent messages.

Some public-key protocols are not intended to result in the exchange of a shared key, but instead transfer some other piece of data. In these cases, the required goals are generally obvious from the context.

In the following sections, we examine a number of protocols and determine the nature of the guarantees they offer.

4. THE KERBEROS PROTOCOL

The Kerberos protocol establishes a shared key between two principals with help from an authentication server [13]. It is based on the shared-key Needham–Schroeder protocol [15], but makes use of timestamps as nonces, both to remove security problems [1, 6] and to reduce the total number of messages required. Kerberos was developed as part of Project Athena at MIT and is also used elsewhere.

We give the protocol below, with A and B as the two principals, K_{as} and K_{bs} as their private keys, and S as the authentication server. S and A generate the time stamps T_s and T_a, respectively, and S generates the lifetime L. The fourth message is used only if mutual authentication is required.

Message 1. $A \rightarrow S$: A, B.
Message 2. $S \rightarrow A$: $\{T_s, L, K_{ab}, B, \{T_s, L, K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$.
Message 3. $A \rightarrow B$: $\{T_s, L, K_{ab}, A\}_{K_{bs}}, \{A, T_a\}_{K_{ab}}$.
Message 4. $B \rightarrow A$: $\{T_a + 1\}_{K_{ab}}$.

This message sequence is represented in Figure 1. First, A sends a cleartext message to S stating his desire to communicate with B. The server responds with an encrypted message containing a timestamp, a lifetime, a session key for A and B, and a *ticket* that only B can read. This ticket also contains the timestamp, the lifetime, and the key. A forwards the ticket to B together with an *authenticator* (a timestamp encrypted with the session key). B first decrypts the ticket and checks the timestamp and lifetime. If the ticket has been created recently enough, he uses the enclosed key to decrypt the authenticator. Then, if the authenticator's

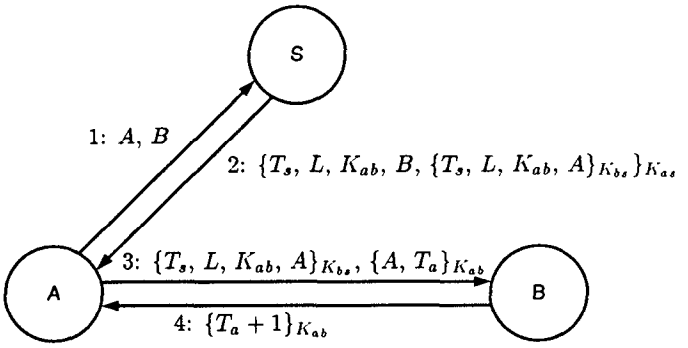

Fig. 1. The Kerberos Protocol.

timestamp is recent, he uses the session key to return the timestamp, which A checks. Once the principals are satisfied, they can proceed to use the session key.

We idealize the protocol as follows:

Message 2. $S \rightarrow A$: $\{T_s, A \overset{K_{ab}}{\leftrightarrow} B, \{T_s, A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}\}_{K_{as}}$.

Message 3. $A \rightarrow B$: $\{T_s, A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}, \{T_a, A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{ab}}$ *from* A.

Message 4. $B \rightarrow A$: $\{T_a, A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{ab}}$ *from* B.

The idealized messages correspond quite closely to the messages described in the published protocol. For simplicity, the lifetime L has been combined with the time stamp T_s, which is treated just like a nonce. The first message is omitted, since it does not contribute to the logical properties of the protocol.

A further difference can be seen in the idealized form of Message 2. The concrete protocol mentions the key K_{ab}, which in this sequence has been replaced by the statement that A and B can use K_{ab} to communicate. This interpretation of the messages is possible only because we know how the information in the messages should be understood. Moreover, the idealized forms of the authenticator and of Message 4 contain the explicit statement that K_{ab} is a good session key, whereas this statement is only implicit in the use of K_{ab} in the concrete protocol. In fact, we could soundly add B **believes** A **believes** $A \overset{K_{ab}}{\leftrightarrow} B$ to Message 4; we do not do so simply because the consequences of this addition seem of little importance for the subsequent use of the session key.

There is some potential for confusion between the second half of the third message and the last message. In the idealized protocol, we avoid this confusion by mentioning the originators explicitly. In the concrete protocol, either the mention of A in the third message or the addition in the fourth suffices to distinguish the two—Kerberos is slightly redundant in this respect.

At this point, we can check that the idealized protocol corresponds to the concrete one and that the guidelines for constructing idealized protocols are respected.

4.1 The Protocol Analyzed

To analyze this protocol, we first give the following assumptions:

A **believes** $A \overset{K_{as}}{\leftrightarrow} S$,	B **believes** $B \overset{K_{bs}}{\leftrightarrow} S$,
S **believes** $A \overset{K_{as}}{\leftrightarrow} S$,	S **believes** $B \overset{K_{bs}}{\leftrightarrow} S$,
S **believes** $A \overset{K_{ab}}{\leftrightarrow} B$,	B **believes** (S **controls** $A \overset{K}{\leftrightarrow} B$),
A **believes** (S **controls** $A \overset{K}{\leftrightarrow} B$),	B **believes fresh**(T_s),
A **believes fresh**(T_s),	B **believes fresh**(T_a).

The first group of four is about shared keys between the clients and the server. The fifth indicates that the server initially knows a key for communication between A and B. The next group of two indicates the trust that A and B have in the server to generate a good encryption key. The final three assumptions show that A and B believe that timestamps generated elsewhere are fresh; this indicates that the protocol relies heavily on the use of synchronized clocks.

We analyze the idealized version of Kerberos by applying our rules to the assumptions; the analysis is straightforward. In the interests of brevity, we give many of the formal details necessary for our machine-assisted proof only for Message 2, and we omit similar details later on. The main steps of the proof are as follows:

A receives Message 2. The annotation rules yield that

$$A \textbf{ sees } \{T_s, (A \overset{K_{ab}}{\leftrightarrow} B), \{T_s, A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}\}_{K_{as}}$$

holds afterward. Since we have the hypothesis

$$A \textbf{ believes } A \overset{K_{as}}{\leftrightarrow} S$$

the message-meaning rule for shared keys applies and yields the following:

$$A \textbf{ believes } S \textbf{ said } (T_s, (A \overset{K_{ab}}{\leftrightarrow} B), \{T_s, A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}})$$

One of our rules to break conjunctions (omitted here) then produces

$$A \textbf{ believes } S \textbf{ said } (T_s, (A \overset{K_{ab}}{\leftrightarrow} B))$$

Moreover, we have the following hypothesis:

$$A \textbf{ believes fresh}(T_s)$$

The nonce-verification rule applies and yields

$$A \textbf{ believes } S \textbf{ believes } (T_s, A \overset{K_{ab}}{\leftrightarrow} B)$$

Again, we break a conjunction, to obtain the following:

$$A \textbf{ believes } S \textbf{ believes } A \overset{K_{ab}}{\leftrightarrow} B$$

Then, we instantiate K to K_{ab} in the hypothesis

$$A \textbf{ believes } S \textbf{ controls } A \overset{K}{\leftrightarrow} B$$

deriving the more concrete

$$A \textbf{ believes } S \textbf{ controls } A \overset{K_{ab}}{\leftrightarrow} B$$

Finally, the jurisdiction rule applies, and yields the following:

$$A \textbf{ believes } A \overset{K_{ab}}{\leftrightarrow} B$$

This concludes the analysis of Message 2.

A passes the ticket on to B, together with another message containing a timestamp. Initially, B can decrypt only the ticket:

$$B \textbf{ believes } A \overset{K_{ab}}{\leftrightarrow} B$$

Logically, this result is obtained in the same way as that for Message 2, via the message-meaning, nonce-verification, and jurisdiction postulates.

Knowledge of the new key allows B to decrypt the rest of Message 3. Through the message-meaning and the nonce-verification postulates, we deduce the following:

$$B \textbf{ believes } A \textbf{ believes } A \overset{K_{ab}}{\leftrightarrow} B$$

The fourth message simply assures A that B believes in the key and has received A's last message. After new applications of the message-meaning and nonce-verification postulates to the fourth message, the final result is as follows:

$$A \textbf{ believes } A \overset{K_{ab}}{\leftrightarrow} B \qquad B \textbf{ believes } A \overset{K_{ab}}{\leftrightarrow} B$$

$$A \textbf{ believes } B \textbf{ believes } A \overset{K_{ab}}{\leftrightarrow} B \qquad B \textbf{ believes } A \textbf{ believes } A \overset{K_{ab}}{\leftrightarrow} B$$

If only the first three messages are used, we do not obtain

$$A \textbf{ believes } B \textbf{ believes } A \overset{K_{ab}}{\leftrightarrow} B$$

That is, the three-message protocol does not convince A of B's existence—A observes the same messages whether B is running or not.

Although the result resembles that for the Needham–Schroeder protocol [3], a major assumption in the Kerberos protocol is that the principal's clocks are synchronized with the server's clock. The effect of totally synchronized clocks can be obtained by synchronizing clocks to within a few minutes with a secure time server and then detecting replays within this interval. However, actual implementations do not always include this check and so provide only weaker guarantees.

A slight (but potentially expensive) peculiarity is that S double-encrypts the ticket in the second message. Looking back through the formal analysis, we see that this does not affect the properties of the protocol, since A forwards the ticket to B immediately afterward without further encryption. It has recently been proposed that future versions of Kerberos remove this unnecessary double encryption.

5. THE ANDREW SECURE RPC HANDSHAKE

An early version of the Andrew secure RPC protocol uses an authentication handshake between two principals whenever a client binds to a new server [18]. The handshake is intended to allow a client A to obtain a new session key K'_{ab} from a server B, given that they already share a key K_{ab}. This protocol is vulnerable to an attack similar to that observed in the case of the shared-key Needham–Schroeder protocol. We give it here as an illustration of how easily

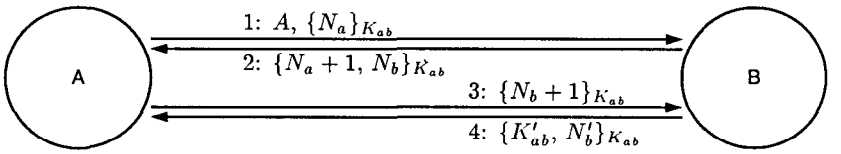

Fig. 2. The Andrew Square RPC Handshake.

such problems can be missed and of how they manifest themselves in the logic (see Figure 2).

Message 1. $A \rightarrow B$: $A, \{N_a\}_{K_{ab}}$.
Message 2. $B \rightarrow A$: $\{N_a + 1, N_b\}_{K_{ab}}$.
Message 3. $A \rightarrow B$: $\{N_b + 1\}_{K_{ab}}$.
Message 4. $B \rightarrow A$: $\{K'_{ab}, N'_b\}_{K_{ab}}$.

N_a and N_b are nonces; N'_b is an initial sequence number to be used in subsequent communication. The first message simply transfers a nonce, which B returns in the second message. If A is satisfied with the reply, he returns B's nonce. After B receives and checks the third message, he sends a new session key to A. As in the Kerberos protocol, nonces are returned incremented by one, even though there is no danger of generating identical messages during the protocol.

The idealized protocol closely resembles the concrete one:

Message 1. $A \rightarrow B$: $\{N_a\}_{K_{ab}}$.
Message 2. $B \rightarrow A$: $\{N_a, N_b\}_{K_{ab}}$.
Message 3. $A \rightarrow B$: $\{N_b\}_{K_{ab}}$.
Message 4. $B \rightarrow A$: $\{A \overset{K'_{ab}}{\leftrightarrow} B, N'^{b}\}_{K_{ab}}$.

5.1 The Protocol Analyzed

First, we write the following assumptions:

A **believes** $A \overset{K_{ab}}{\leftrightarrow} B$	B **believes** $A \overset{K_{ab}}{\leftrightarrow} B$
A **believes** (B **controls** $A \overset{K}{\leftrightarrow} B$)	B **believes** $A \overset{K'_{ab}}{\leftrightarrow} B$
A **believes fresh**(N_a)	B **believes fresh**(N_b)
	B **believes fresh**(N'_b)

The first group of two indicates that A and B initially share a key. The next two show that B has invented a new key and that A trusts B to invent good keys. Finally, each client is able to generate fresh nonces.

For brevity, we do not describe our deductions, and simply list the final results:

B **believes** $A \overset{K'_{ab}}{\leftrightarrow} B$
A **believes** B **said** ($A \overset{K'_{ab}}{\leftrightarrow} B, N'_b$)
B **believes** A **believes** N_b
A **believes** B **believes** (N_a, N_b)

Unfortunately, we can go no further. We cannot obtain A **believes** B **believes** $A \overset{K'_{ab}}{\leftrightarrow} B$ because there is nothing in the fourth message that A believes to be fresh. We must conclude that the protocol suffers from the weakness that an intruder can replay an old message as the last message in the protocol and can convince A to use an old, possibly compromised session key. In other words, an intruder may find an old session key and may replay the fourth message of the handshake in which that key was established; he can then impersonate B. The problem can be fixed simply by adding the nonce N_a to the last message, and indeed a descendant of the Andrew file system has adopted this solution.

In fact, more substantial changes to the protocol can also reduce the total amount of encryption needed. Only two messages need be encrypted, one from A and one from B. First, B sends a key K'_{ab} (along with a nonce N_a):

$$B \rightarrow A: \{_a, A \overset{K'_{ab}}{\leftrightarrow} B\}_{K_{ab}}.$$

In a concrete implementation, N_a may be a timestamp or a nonce that A sent to B in a recent unencrypted message. A must reply with an acknowledgment that K'_{ab} has been accepted:

$$A \rightarrow B: \{ \overset{K'_{ab}}{\leftrightarrow} B\}_{K'_{ab}}.$$

B believes this message is timely because K'_{ab} is fresh. Optionally, B can go on to send an initial sequence number N'_b in clear.

If this new protocol is analyzed, we obtain the following stronger outcome:

A **believes** $A \overset{K'_{ab}}{\leftrightarrow} B$	B **believes** $A \overset{K'_{ab}}{\leftrightarrow} B$
A **believes** B **believes** $A \overset{K'_{ab}}{\leftrightarrow} B$	B **believes** A **believes** $A \overset{K'_{ab}}{\leftrightarrow} B$

As a concrete realization of the protocol, we propose the following:

Message 1. $A \rightarrow B: A, N_a$.
Message 2. $B \rightarrow A: \{N_a, K'_{ab}\}_{K_{ab}}$.
Message 3. $A \rightarrow B: \{N_a\}_{K'_{ab}}$.
Message 4. $B \rightarrow A: N'_b$.

In Message 3, the use of N_a is arbitrary; any predictable message will assure B that A has encrypted something with the new key.

6. THE NEEDHAM–SCHROEDER PUBLIC-KEY PROTOCOL

In their original paper, Needham and Schroeder proposed a protocol based on public-key cryptography; it allowed two principals to exchange two secret numbers [15]. A weakness in the protocol permits a replay attack in the interactions with the certification authority if a key is compromised, as in the shared-key Needham–Schroeder protocol.

Here, S, whose public key is K_s, operates only as a certification authority between A and B, whose public keys are K_a and K_b, respectively; N_a and N_b are

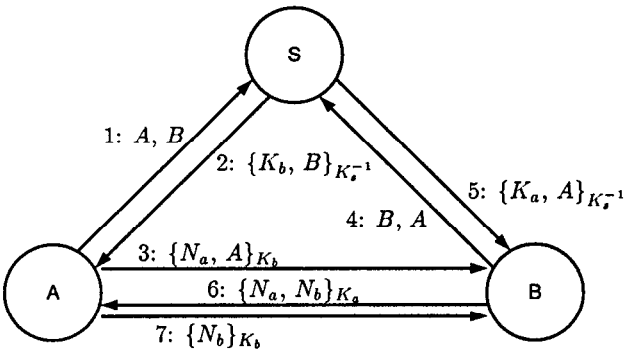

Fig. 3. The Needham–Schroeder Public-Key Protocol.

nonces. The message exchange goes as follows (see Figure 3):

Message 1. $A \rightarrow S$: A, B.
Message 2. $S \rightarrow A$: $\{K_b, B\}_{K_s^{-1}}$.
Message 3. $A \rightarrow B$: $\{N_a, A\}_{K_b}$.
Message 4. $B \rightarrow S$: B, A.
Message 5. $S \rightarrow B$: $\{K_a, A\}_{K_s^{-1}}$.
Message 6. $B \rightarrow A$: $\{N_a, N_b\}_{K_a}$.
Message 7. $A \rightarrow B$: $\{N_b\}_{K_b}$.

The protocol has two rather independent but interleaved components. It is expected that, initially, both A and B hold S's public key K_s. Therefore, the principals A and B can obtain each other's public keys from S. Messages 1, 2, 4, and 5 accomplish this purpose. In a second component, in Messages 3, 6, and 7, A and B use the public keys obtained. They communicate the secret nonce identifiers N_a and N_b. These secrets can be used later, for signing further messages. For example, if B receives a message $\{X, N_a\}_{K_b}$, then B may deduce that A sent X.

The idealized protocol is as follows:

Message 2. $S \rightarrow A$: $\{\overset{K_b}{\mapsto} B\}_{K_s^{-1}}$.
Message 3. $A \rightarrow B$: $\{N_a\}_{K_b}$.
Message 5. $S \rightarrow B$: $\{\overset{K_a}{\mapsto} A\}_{K_s^{-1}}$.
Message 6. $B \rightarrow A$: $\{\langle A \overset{N_b}{\rightleftharpoons} B \rangle_{N_a}\}_{K_a}$.
Message 7. $A \rightarrow B$: $\{\langle A \overset{N_a}{\rightleftharpoons} B \rangle_{N_b}\}_{K_b}$.

Messages 1 and 4 are deliberately omitted, since they do not contribute to the logical properties of the protocol. Messages 2 and 5 are straightforward, but the others require some explanation. It is interesting to note the difference between Message 3 and Messages 6 and 7. In Message 3, N_a is not known to B and thus is not being used to prove the identity of A; Message 3 is used simply to convey N_a to B. In Messages 6 and 7, N_a and N_b are used as secrets, so the $\langle X \rangle_Y$ notation

is used. These messages also convey beliefs that have no representation in the concrete protocol, because the messages would not be sent if the beliefs were not held. In fact, just as in the case of Kerberos, we could soundly add more statements of belief to Messages 6 and 7.

6.1 The Protocol Analyzed

First we state the assumed initial beliefs of the players:

A **believes** $\overset{K_a}{\mapsto} A$	B **believes** $\overset{K_b}{\mapsto} B$
A **believes** $\overset{K_s}{\mapsto} S$	B **believes** $\overset{K_s}{\mapsto} S$
S **believes** $\overset{K_a}{\mapsto} A$	S **believes** $\overset{K_b}{\mapsto} B$
S **believes** $\overset{K_s}{\mapsto} S$	
A **believes** (S **controls** $\overset{K}{\mapsto} B$)	B **believes** (S **controls** $\overset{K}{\mapsto} A$)
A **believes fresh**(N_a)	B **believes fresh**(N_b)
A **believes** $A \overset{N_a}{\rightleftharpoons} B$	B **believes** $A \overset{N_b}{\rightleftharpoons} B$
A **believes fresh**($\overset{K_b}{\mapsto} B$)	B **believes fresh**($\overset{K_a}{\mapsto} A$)

Each principal knows the public key of the certification agent S, as well as his own keys. In addition, S knows the public keys of A and B. Each principal trusts the certification agent to correctly sign certificates giving the public key of the other. Also, each principal believes that the secret that he generates is fresh. The last two assumptions are surprising and represent a weakness in the protocol. Each principal must assume that the message containing the public key of the other principal is fresh. The difficulty could be resolved by adding timestamps to Messages 2 and 5. This is analogous to the way that Kerberos' timestamps overcome the problem with the shared-key Needham–Schroeder protocol.

We obtain the final beliefs:

A **believes** $\overset{K_b}{\mapsto} B$	B **believes** $\overset{K_a}{\mapsto} A$
A **believes** B **believes** $A \overset{N_b}{\rightleftharpoons} B$	B **believes** A **believes** $A \overset{N_a}{\rightleftharpoons} B$

Each principal knows the public key of the other and has knowledge of a shared secret that he believes the other will accept as being shared only by the two principals. From this point, A and B can continue to exchange messages using N_a, N_b, and public-key encryption. In this way they can transfer data or other keys securely.

7. THE CCITT X.509 PROTOCOL

A draft recommendation for the CCITT X.509 standard contains a set of three protocols using between one and three messages [4]. (It is our understanding that this has now become an official recommendation of the CCITT.) The protocols are intended for signed, secure communication between two principals, assuming that each knows the public key of the other. The three-message version is given below. The two-message and one-message protocols are formed by removing the last one or two messages, respectively.

The published protocol contains two weaknesses, either of which can be exploited by an intruder, as we shall demonstrate below. We found one of these

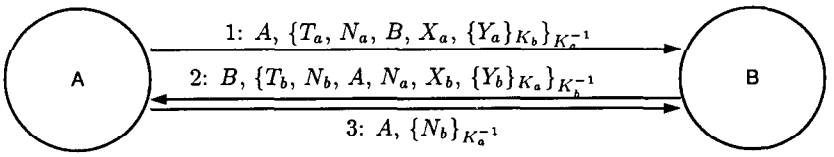

Fig. 4. The CCITT X. 509 Protocol.

weaknesses while idealizing the protocol and the other during the subsequent analysis (see Figure 4).

Message 1. $A \rightarrow B$: $A, \{T_a, N_a, B, X_a, \{Y_a\}_{K_b}\}_{K_a^{-1}}$.
Message 2. $B \rightarrow A$: $B, \{T_b, N_b, A, N_a, X_b, \{Y_b\}_{K_a}\}_{K_b^{-1}}$.
Message 3. $A \rightarrow B$: $A, \{N_b\}_{K_a^{-1}}$.

Here, T_a and T_b are timestamps, N_a and N_b are nonces, and X_a, Y_a, X_b, and Y_b are user data. The protocol ensures the integrity of X_a and X_b, assuring the recipient of their origin, and guarantees the privacy of Y_a and Y_b.

For the idealized protocol, we simply take the following:

Message 1. $A \rightarrow B$: $\{T_a, N_a, X_a, \{Y_a\}_{K_b}\}_{K_a^{-1}}$.
Message 2. $B \rightarrow A$: $\{T_b, N_b, N_a, X_b, \{Y_b\}_{K_a}\}_{K_b^{-1}}$.
Message 3. $A \rightarrow B$: $\{N_b\}_{K_a^{-1}}$.

As usual, the timestamps T_a and T_b are viewed as nonces.

7.1 The Protocol Analyzed

We assume that each principal knows his own secret key and the other's public key, and believes his own nonce and the other's timestamp to be fresh.

A **believes** $\overset{K_a}{\mapsto} A$	B **believes** $\overset{K_b}{\mapsto} B$
A **believes** $\overset{K_b}{\mapsto} B$	B **believes** $\overset{K_a}{\mapsto} A$
A **believes fresh**(N_a)	B **believes fresh**(N_b)
A **believes fresh**(T_b)	B **believes fresh**(T_a)

Now we can derive the following:

A **believes** B **believes** X_b $\qquad$ B **believes** A **believes** X_a

This represents an outcome weaker than the authors desired; in particular, we do not obtain B **believes** A **believes** Y_a or A **believes** B **believes** Y_b. Although Y_a and Y_b have each been transferred in a signed message, there is no evidence to suggest that the sender is actually aware of the data that he sent in the private part of the message. This corresponds to a scenario where some third party intercepts a message and removes the existing signature while adding his own, blindly copying the encrypted section within the signed message. The simplest fix is to sign the secret data Y_a and Y_b before it is encrypted for privacy.

Some redundancy is noticeable in the second message; either T_b or N_a is sufficient to ensure the timeliness of the message. The protocol description states

that the checking of T_b is optional in the three-message version of the protocol. In fact, it is perfectly reasonable to omit T_b altogether, since it is redundant in both the two- and three-message protocols.

Unfortunately, the CCITT X.509 document also suggests that T_a need not be checked in the three-message protocol. This is a serious problem because the checking of T_a is the only mechanism that establishes the timeliness of the first message. Logically, if T_a is not checked, we cannot perform nonce verification on the first message, and we obtain only the weaker outcome B **believes** A **said** X_a instead of B **believes** A **believes** X_a.

This difficulty explains the intention of the third message, which is to assure B that A generated his first message recently. The authors seem to have hoped that the use of N_b would suffice to link the third message to the first, since N_b links the last two messages and N_a links the first two messages. The error here is that N_b alone does not link the last two messages, and this may allow an intruder C to replay one of A's old messages and thereafter to impersonate A.

The following concrete exchange illustrates the flaw. The intruder first contacts B:

$$C \rightarrow B\text{: } A,\ \{T_a,\ N_a,\ B,\ X_a,\ \{Y_a\}_{K_b}\}_{K_a^{-1}}.$$

This is an old message originally sent by A. Remember that B is not presumed to check the timestamp T_a in the three-message protocol and so will not discover the replay of A's original message. B replies as though the message came from A, and provides a new nonce, N_b.

$$B \rightarrow C\text{: } B,\ \{T_b,\ N_b,\ A,\ N_a,\ X_b,\ \{Y_b\}_{K_a}\}_{K_b^{-1}}$$

At this point C causes A to initiate authentication with C, by whatever means.

$$A \rightarrow C\text{: } A,\ \{T_a',\ N_a',\ C,\ X_a',\ \{Y_a'\}_{K_c}\}_{K_a^{-1}}$$

C replies to A, providing the nonce N_b. (The nonce N_b is not secret, and nothing prevents C from using the same value in an instance of the protocol between A and C.)

$$C \rightarrow A\text{: } C,\ \{T_c,\ N_b,\ A,\ N_a',\ X_c,\ \{Y_c\}_{K_a}\}_{K_c^{-1}}$$

A replies to C, signing the exact message needed for C to convince B that the first message was sent recently by A and is not a replay of an old message—hence, this may allow C to impersonate A.

$$A \rightarrow C\text{: } A,\ \{N_b\}_{K_a^{-1}}$$

One solution is to include B's name in the last message. Since B guarantees the uniqueness of his own nonces, he can be sure that this message is linked to this instance of the protocol. The idealized version of Message 3 could then include any beliefs transmitted in Message 1, assuring B of their timeliness.

The X.509 protocol actually uses hashing to reduce the amount of encryption: In order to sign a Message m, a hash $H(m)$ of m is computed and signed. This has not been shown in the description above. The logic and the analysis of this particular protocol are changed only slightly by the introduction of hashing [3].

Table I. Summary of Results

	Needham–Schroeder shared key	Otway–Rees	Kerberos	Wide-mouthed frog	Yahalom	Andrew RPC	Needham–Schroeder public key	CCITT X.509
Goal	Distribute key	Distribute key	Distribute key	Distribute key[a]	Distribute key	Distribute extra key	Establish secrets	Transfer data
Keys	Shared	Shared	Shared	Shared	Shared	Shared	Public	Public
Uses secrets					×		×	
Nonces/clocks	Nonces	Nonces	Clocks	Clocks	Nonces	Nonces	Nonces	Both
Proves presence of	*A* and *B*	*B*	*A* and *B*[b]	*A*	*A* and *B*	*A* and *B*	*A* and *B*	*A* and *B*[b]
Redundancy	×	×	×		×	×		×
Bugs	×					×	×	×[c]

[a] In this case, *A*, rather than a trusted server, generates the key.
[b] *B*'s presence is guaranteed to *A* only if optimal protocol steps are used.
[c] Security breaches do not even require key compromise.

8. CONCLUSIONS

Recent literature has emphasized the importance of reasoning about knowledge for understanding distributed computation (e.g., [8]). Furthermore, there have been some formal descriptions of cryptographic protocols [5, 9, 11]. Although these works are not closely related to ours and, in particular, they have not suggested useful proof systems, they could serve as a foundation for our more specific analysis of authentication protocols.

In this paper we have described a logic to reason about authentication protocols and have treated several examples. Table I lists protocols studied with the logic, including some discussed in [3], and summarizes their attributes. The wide-mouthed-frog and the Yahalom protocols are described only in [3]; references for the other protocols can be found below.

The table shows some well-known properties:

—the goal of each protocol;
—the type of cryptosystem used, shared key or public key;
—whether secrets (other than keys) are used; and
—whether message timeliness is guaranteed with nonces or synchronized clocks.

In addition, we include aspects that our formalism helped bring to light:

—whether the protocol proves the presence of each party to the other,
—redundancy, and
—security problems.

In Table I the principals involved in the protocols are *A* and *B*; the initiator is *A*.

The examples show how a simple logic can capture subtle differences between protocols. For a variety of protocols, it enables us to exhibit step by step how beliefs are built up to the point of mutual authentication. For other protocols, it guides us in identifying mistakes and in suggesting corrections.

ACKNOWLEDGMENTS

The work was undertaken as the result of a suggestion by Butler Lampson. Andrew Birrell, Luca Cardelli, Dorothy Denning, Butler Lampson, Tim Mann, Michael Schroeder, Jennifer Steiner, and many referees encouraged the work and suggested improvements to the paper. Chris Mitchell provided information on the CCITT protocol. Kathleen Sedehi typeset an early version of this paper and produced the figures, and Cynthia Hibbard provided editorial assistance.

REFERENCES

1. BAUER, R. K., BERSON, T. A., AND FEIERTAG, R. J. A key distribution protocol using event markers. *ACM Trans. Comput. Syst. 1*, 3 (Aug. 1983), 249–255.
2. BURROWS, M., ABADI, M., AND NEEDHAM, R. M. Authentication: A practical study in belief and action. In *Proceedings of the 2nd Conference on Theoretical Aspects of Reasoning about Knowledge* (Asilomar, Ca., Feb. 1988) M. Vardi, Ed. Morgan Kaufmann, Los Altos, Calif., 1988, pp. 325–342.
3. BURROWS, M., ABADI, M., AND NEEDHAM, R. M. A logic of authentication. Rep. 39, Digital Equipment Corporation Systems Research Center, Palo Alto, Calif., Feb. 1989.
4. CCITT. CCITT draft recommendation X.509. The directory-authentication framework, version 7. CCITT, Gloucester, Nov. 1987.
5. DEMILLO, R. A., LYNCH, N. A., AND MERRITT, M. J. Cryptographic protocols. In *Proceedings of the 14th ACM Symposium on the Theory of Computing* (San Francisco, May 1982), ACM, New York, 1982, pp. 383–400.
6. DENNING, D. E., AND SACCO, G. M. Timestamps in key distribution protocols. *Commun. ACM 24*, 8 (Aug. 1981), 533–536.
7. DOLEV, D., AND YAO, A. C. On the security of public key protocols. *IEEE Trans. Inf. Theory IT-29*, 2 (Mar. 1983), 198–208.
8. HALPERN, J. Y., AND MOSES, Y. O. Knowledge and common knowledge in a distributed environment. In *Proceedings of the 3rd ACM Conference on the Principles of Distributed Computing* (Vancouver, British Columbia, Aug. 1984), ACM, New York, 1984, pp. 480–490.
9. HALPERN, J. Y., MOSES, Y. O., AND TUTTLE, M. R. A knowledge-based analysis of zero knowledge (preliminary report). In *Proceedings of the 20th ACM Symposium on Theory of Computing* (Chicago, Ill., May 1988), ACM, New York, 1988, pp. 132–147.
10. HOARE, C. A. R. An axiomatic basis for computer programming. *Commun. ACM 12*, 10 (Oct. 1969), 576–580.
11. MERRITT, M. J., AND WOLPER, P. L. States of knowledge in cryptographic protocols. Draft.
12. MILLEN, J. K., CLARK, S. C., AND FREEDMAN, S. B. The interrogator: Protocol security analysis. *IEEE Trans. Softw. Eng. SE-13*, 2 (Feb. 1987), 274–288.
13. MILLER, S. P., NEUMAN, C., SCHILLER, J. I., AND SALTZER, J. H. Kerberos authentication and authorization system. In *Project Athena Technical Plan*, Sect. E.2.1. MIT, Cambridge, Mass., July 1987.
14. National Bureau of Standards. Data encryption standard. Fed. Inf. Process. Stand. Publ. 46. National Bureau of Standards, Washington, D.C., Jan. 1977.
15. NEEDHAM, R. M., AND SCHROEDER, M. D. Using encryption for authentication in large networks of computers. *Commun. ACM 21*, 12 (Dec. 1978), 993–999.
16. OTWAY, D., AND REES, O. Efficient and timely mutual authentication. *Oper. Syst. Rev. 21*, 1 (Jan. 1987), 8–10.
17. RIVEST, R. L., SHAMIR, A., AND ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM 21*, 2 (Feb. 1978), 120–126.
18. SATYANARAYANAN, M. Integrating security in a large distributed system. *ACM Trans. Comput. Syst. 7*, 3 (Aug. 1989), 247–280.
19. VOYDOCK, V. L., AND KENT, S. T. Security mechanisms in high-level network protocols. *ACM Comput. Surv. 15*, 2 (June 1983), 135–171.

Received May 1989; revised September 1989; accepted October 1989