A Logic of Authentication

Michael Burrows Martín Abadi Roger Needham

Questions of belief are essential in analyzing protocols for the authentication of principals in distributed computing systems. In this paper we motivate, set out, and exemplify a logic specifically designed for this analysis; we show how various protocols differ subtly with respect to the required initial assumptions of the participants and their final beliefs. Our formalism has enabled us to isolate and express these differences with a precision that was not previously possible. It has drawn attention to features of protocols of which we and their authors were previously unaware, and allowed us to suggest improvements to the protocols. The reasoning about some protocols has been mechanically verified.

This paper starts with an informal account of the problem, goes on to explain the formalism to be used, and gives examples of its application to protocols from the literature, both with shared-key cryptography and with public-key cryptography. Some of the examples are chosen because of their practical importance, while others serve to illustrate subtle points of the logic and to explain how we use it. We discuss extensions of the logic motivated by actual practice—for example, in order to account for the use of hash functions in signatures. The final sections contain a formal semantics of the logic and some conclusions.

1. The Problem

In distributed computing systems and similar networks of computers it is necessary to have procedures by which various pairs of *principals* (people, computers, services) satisfy themselves mutually about each other's identity. A common way to approach this is by means of secrets, usually encryption keys. In barest outline, an *authentication protocol* guarantees that if the principals really are who they say they are then they will end up in possession of one or more shared secrets, or at least they will become able to recognize the use of other principals' secrets (see, for example, Needham & Schroeder 1978). For example, some authentication protocols establish shared encryption keys that principals can use in subsequent communication. Other authentication protocols, based on public-key cryptography (see, for example, Rivest *et al.* 1978), first distribute the public keys of principals, and then use them to establish shared secrets.

Protocols involving shared-key cryptography use an *authentication server* which shares a key with each principal and typically generates new session keys for communication between the principals. Public-key protocols use a *certification authority* which has a well-known public key and is trusted to pass on the public keys of the principals. Both authentication servers and certification authorities are trusted to make proper use of the data they hold when executing authentication protocols; authentication servers are often also trusted to generate new secrets in a proper manner—for example, not to issue the same one every time.

Authentication would be straightforward in a sufficiently benign environment. Such cannot usually be assumed, and it is particularly necessary to take precautions against confusion caused by the reissue of old messages. Specifically, one must ensure that a replay cannot force the use of an old and possibly compromised secret. A good part of the work in the literature is devoted to ensuring that the information upon which the protocols act is timely. The style of the precautions taken has caused it to be recognized for a long time that we are dealing with questions of belief, trust, and delegation; however, this has been recognized only imprecisely rather than in any formalism.

As a result of varied design decisions appropriate to different circumstances, a variety of authentication protocols exist. Therefore, there is a perceived need to explicate them formally to understand to what extent they really achieve the same results. They differ subtly in their final states, and sometimes a protocol may be shown to depend on assumptions that one might not care to make. We define a logic of authentication in order to explain the protocols step by step, with all initial assumptions made explicit and with the final states clearly set out.

These are examples of the sort of questions we would like to be able to answer with the help of formal methods:

What does this protocol achieve?

Does this protocol need more assumptions than another one?

Does this protocol do anything unnecessary that could be left out without weakening it?

Does this protocol encrypt something that could be sent in clear without weakening it?

In later sections, we show how the logic is able to help us answer these questions for a number of example protocols.

It is important to note that certain aspects of authentication protocols have been deliberately ignored in our treatment. Since we operate at an abstract level, we do not consider errors introduced by concrete implementations of a protocol, such as deadlocks, or even inappropriate use of cryptosystems (as in Voydock & Kent 1983). Furthermore, while we allow for the possibility of hostile intruders, there is no attempt to deal with the authentication of an untrustworthy principal, nor to detect weaknesses of encryption schemes or unauthorized release of secrets (as in Dolev & Yao 1983; Millen *et al.* 1987). Rather, our study concentrates on the beliefs of trustworthy parties involved in the protocols and on the evolution of these beliefs as a consequence of communication. Our experience with authentication protocols has indicated that this kind of study is one of the most needed by current protocol designers.

We focus on fairly standard uses of encryption and secrets. We also discuss some variants of practical importance, and how they can be explained logically. Our treatment includes both shared-key cryptography and public-key cryptography. (Shared-key cryptography can be treated in isolation, as in (Burrows *et al.* 1988); on the other hand, a logic for public keys should include one for shared keys: for economic reasons, many practical schemes based on public keys use some shared keys.)

Our goal, however, is not to provide a logic that would explain every authentication method, but rather a logic that would explain most of the central concepts in authentication. This logic may serve both as a basic tool and for illustrative purposes. It is hoped that protocol designers will adapt it to suit their specific needs.

Before introducing the notation and technical terminology, it may be worth stating some of the main principles in the vernacular. They cannot be regarded as precise, of course.

If you've sent Joe a number that you have never used for this purpose before and if you subsequently receive from Joe something that depends on knowing that number, then you ought to believe that Joe's message originated recently—in fact, after yours.

If you believe that only you and Joe know K, then you ought to believe that anything you receive encrypted with K as key comes originally from Joe.

If you believe that K is Joe's public key, then you should believe that any message that you can decrypt with K comes originally from Joe.

If you believe that only you and Joe know X, then you ought to believe that any encrypted message you receive containing X comes originally from Joe.

2. The Formalism

Authentication protocols are typically described by listing the messages sent between the principals, and by symbolically showing the source, the destination, and the contents of each message. This conventional notation is not convenient for manipulation in a logic, since we wish to attach exact meanings to each part of each message and these meanings are not always apparent from the data contained in the messages. In order to introduce a more useful notation whilst preserving correspondence with the original description of the protocols, we transform each message into a logical formula. This logical formula is an idealized version of the original message. Then we annotate each idealized protocol with assertions, much as in a proof in Hoare logic (Hoare 1969). These assertions are expressed in the same notation used to write messages. An assertion usually describes beliefs held by the principals at the point in the protocol where the assertion is inserted.

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

Basic notation

Our formalism is built on a many-sorted modal 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; 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; K ranges over encryption keys. All these may be used either as metasymbols (to write schemata) or as free variables (with an implicit universal quantification); this minor confusion is essentially harmless.

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:

$P \models X$: 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.

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

$P \mid\sim X$: 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 when he sent the message.

$P \Rightarrow X$: P has *jurisdiction* over X. The principal P is an authority on X and should be trusted on this matter. This construct is used when a principal has delegated authority over some statement. For example, encryption keys need to be generated with some care, and in some protocols certain servers are trusted to do this properly. This may be expressed by the assumption that the principals believe that the server has jurisdiction over statements about the quality of keys.

$\sharp(X)$: 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 generated for the purpose of being fresh. Nonces commonly include a timestamp or a number that is used only once, such as a sequence number.

$P \overset{K}{\leftrightarrow} Q$: 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* (the inverse of K, denoted K^{-1}) will never be discovered by any principal except P, or a principal trusted by P.

$P \overset{X}{\rightleftharpoons} Q$: 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. Often, X is fresh as well as secret. An example of a shared secret is a password.

$\{X\}_K$: This represents the formula X encrypted under the key K. Formally, $\{X\}_K$ is an 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. In the interests of brevity, we typically omit this in our examples.

$\langle X \rangle_Y$: 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 rôle, as proof of origin for X. The notation is intentionally reminiscent of that for encryption, which also guarantees the identity of the source of a message through knowledge of a certain kind of secret.

A more formal treatment of the semantics of these constructs can be found below. Here, we simply give and motivate the rules of inference that characterize them.

Logical postulates

Some informal preliminaries are useful to understand the rules of inference of our logic.

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; furthermore, we assume that when principal P says X then he actually believes X. 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.

An encrypted message is represented as a logical statement bound together and encrypted with an encryption key. It is assumed that the encryption is done in such a way that we know the whole message was sent at once. 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.

Now we are ready to discuss the logical postulates. We do not present the postulates in the most general form possible; our main concern is to have enough machinery to carry out some realistic examples and to explain the essence of our method.

- 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:

  $$\frac{P \models Q \overset{K}{\leftrightarrow} P, \quad P \triangleleft \{X\}_K}{P \models Q \mid\sim X}$$

  That is, if P believes that the key K is shared with Q and sees a message 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 X 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:

  $$\frac{P \models \overset{K}{\mapsto} Q, \quad P \triangleleft \{X\}_{K^{-1}}}{P \models Q \mid\sim X}$$

  For shared secrets, we postulate:

  $$\frac{P \models Q \overset{Y}{\rightleftharpoons} P, \quad P \triangleleft \langle X \rangle_Y}{P \models Q \mid\sim X}$$

  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 $\triangleleft$, given below, guarantee that $\langle X \rangle_Y$ was not just uttered by P himself.

  In real life the decryption of a message to yield a content says, in and of itself, only that the content was produced at some time in the past; we have no idea whether it is new or the result of a replay.

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

  $$\frac{P \models \sharp(X), \quad P \models Q \mid\sim X}{P \models Q \models X}$$

That is, if P believes that X could have been uttered only recently and that Q once said X, 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 $\{Y\}_K$. (When this restriction is not met, we can conclude only that Q has recently said X. We might introduce a "has recently said" operator to express this conclusion, should the need arise in an example.)

This is the only postulate that promotes from $\mid\!\sim$ to $\mid\!\equiv$. It reflects in an abstract and timeless way the practice of protocol designers of using challenges and responses. One participant issues a fresh statement as a challenge. Since the challenge has been generated recently, any message containing it is accepted as timely and taken seriously. In general, challenges need not be encrypted but responses must be.

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

$$\frac{P \mid\!\equiv Q \Rightarrow X, \quad P \mid\!\equiv Q \mid\!\equiv X}{P \mid\!\equiv X}$$

- A necessary property of the belief operator is that P believes a set of statements if and only if P believes each individual statement separately. This justifies the following rules:

$$\frac{P \mid\!\equiv X, \quad P \mid\!\equiv Y}{P \mid\!\equiv (X, Y)} \qquad \frac{P \mid\!\equiv (X, Y)}{P \mid\!\equiv X} \qquad \frac{P \mid\!\equiv Q \mid\!\equiv (X, Y)}{P \mid\!\equiv Q \mid\!\equiv X}$$

  Other similar rules could be introduced as required.

- A similar rule applies to the operator $\mid\!\sim$:

$$\frac{P \mid\!\equiv Q \mid\!\sim (X, Y)}{P \mid\!\equiv Q \mid\!\sim X}$$

  Note that if $P \mid\!\equiv Q \mid\!\sim X$ and $P \mid\!\equiv Q \mid\!\sim Y$ it does *not* follow that $P \mid\!\equiv Q \mid\!\sim (X, Y)$, since this would imply that the two parts X and Y were uttered at the same time.

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

$$\frac{P \triangleleft (X, Y)}{P \triangleleft X} \qquad \frac{P \triangleleft \langle X \rangle_Y}{P \triangleleft X} \qquad \frac{P \mid\!\equiv Q \overset{K}{\leftrightarrow} P, \quad P \triangleleft \{X\}_K}{P \triangleleft X}$$

$$\frac{P \mid\!\equiv \overset{K}{\mapsto} P, \quad P \triangleleft \{X\}_K}{P \triangleleft X} \qquad \frac{P \mid\!\equiv \overset{K}{\mapsto} Q, \quad P \triangleleft \{X\}_{K^{-1}}}{P \triangleleft X}$$

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 statement applies to $\{X\}_{K^{-1}}$.

The first rule in the second line 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 $P \triangleleft X$ and $P \triangleleft Y$ it does *not* follow that $P \triangleleft (X, Y)$, since this would imply that X and Y were uttered at the same time.

- If one part of a formula is known to be fresh, then the entire formula must also be fresh:

$$\frac{P \models \sharp(X)}{P \models \sharp(X, Y)}$$

Other similar rules can be written, for instance to show that if X is fresh then $\{X\}_K$ is fresh; we do not need these rules in our examples.

- The same key is used between a pair of principals in either direction. We write the following two rules to reflect this property:

$$\frac{P \models R \overset{K}{\leftrightarrow} R'}{P \models R' \overset{K}{\leftrightarrow} R} \qquad \frac{P \models Q \models R \overset{K}{\leftrightarrow} R'}{P \models Q \models R' \overset{K}{\leftrightarrow} R}$$

- Similarly, a secret can be used between a pair of principals in either direction. We write the following two rules to reflect this property:

$$\frac{P \models R \overset{X}{\rightleftharpoons} R'}{P \models R' \overset{X}{\rightleftharpoons} R} \qquad \frac{P \models Q \models R \overset{X}{\rightleftharpoons} R'}{P \models Q \models R' \overset{X}{\rightleftharpoons} R}$$

Given the postulates, we can construct proofs in the logic. A formula X is provable in the logic from a formula Y if there is a sequence of formulas $Z_0, \ldots, Z_n$ where $Z_0 = Y$, $Z_n = X$, and each Z_{i+1} can be obtained from previous ones by the application of a rule. As usual, this can be generalized to prove schemata.

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

$$A \models S \mapsto A \overset{K}{\leftrightarrow} B$$

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

$$A \models \forall K.(S \Rightarrow A \overset{K}{\leftrightarrow} B)$$

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

$$A \models \forall K.(S \Rightarrow B \Rightarrow A \overset{K}{\leftrightarrow} B)$$
$$A \models S \Rightarrow \forall K.(B \Rightarrow A \overset{K}{\leftrightarrow} B)$$

convey different meanings.

In some of 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.

Our formal manipulation of quantifiers is quite straightforward. All we use is the ability to instantiate variables in jurisdiction statements, as reflected by the rule

$$\frac{P \models \forall V_1 \ldots V_n.(Q \Rightarrow X)}{P \models Q' \Rightarrow X'}$$

where $Q' \Rightarrow X'$ is the result of simultaneously instantiating the variables $V_1, \ldots, V_n$ in $Q \Rightarrow X$.

Idealized protocols

In the literature, each protocol step is typically written in the form

$$P \rightarrow Q : \mathit{message}$$

This denotes that the principal P sends the message and that the principal Q receives it. The message is presented in an informal notation designed to suggest the bit-string that a concrete implementation would use. Unfortunately, this presentation is often ambiguous and obscure in its meaning, and is not an appropriate basis for formal analysis.

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

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

This 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

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

When the message is sent to B, we may deduce that the formula

$$B \triangleleft \{A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}$$

holds, indicating that the receiving principal becomes aware of the message and can act upon it.

In this idealized form, we omit parts of the message that do not contribute to the beliefs of the recipient. In particular, we remove hints that are added to an implementation to allow it to proceed in a timely fashion, but whose presence would not affect the result of the protocol if each host were to act spontaneously. For instance, we may omit a message used as a hint that communication is to be initiated.

The idealized protocols of the examples given below do not include cleartext message parts; idealized messages are of the form $\{X_1\}_{K_1}, \ldots, \{X_n\}_{K_n}$, where each encrypted part is treated separately. 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. In general, it is easy to generate a useful logical form for a protocol once it is intuitively understood. However, the idealized form of each message cannot be determined by looking merely at a single protocol step by itself. Only knowledge of the entire protocol can determine the essential logical contents of the message. There are guidelines to 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 important, for the sake of soundness, we always want to guarantee that each principal believes the formulas that he generates as messages.

Protocol analysis

From a practical viewpoint, the analysis of a protocol is performed as follows:

- The idealized protocol is derived from the original one.
- Assumptions about the initial state are written.
- Logical formulas are attached to the statements of the protocol, as assertions about the state of the system after each statement.
- The logical postulates are applied to the assumptions and the assertions, in order to discover the beliefs held by the parties in the protocol.

This procedure may be repeated as new assumptions are found to be necessary and as the idealized protocol is refined.

More precisely, we annotate idealized protocols with formulas and manipulate these formulas with the postulates. A protocol is a sequence of "send" statements $S_1, \ldots, S_n$, each of the form $P \rightarrow Q : X$ with $P \neq Q$. An *annotation* for a protocol consists of a sequence of assertions inserted before the first statement and after each statement; the assertions we use are conjunctions of formulas of the forms $P \models X$ and $P \triangleleft X$. The first assertion contains the assumptions, while the last assertion contains the conclusions. Roughly, annotations can be understood as simple formulas in Hoare logic. We write them in the familiar form

$$[\mathit{assumptions}]\ S_1\ [\mathit{assertion}\ 1]\ \ldots\ [\mathit{assertion}\ n-1]\ S_n\ [\mathit{conclusions}]$$

(In the examples below, however, we do not demonstrate the use of this notation, since we wish to concentrate on the assumptions and conclusions.)

We want annotations to be *valid* in the following sense: if the assumptions hold initially, then each assertion holds after the execution of the protocol prefix that it follows. Clearly, validity is a semantic concept. Its syntactic counterpart is derivability; we give rules to derive legal annotations to protocols:

- For single protocol steps: the annotation $[Y]\,(P \rightarrow Q : X)\,[Y, Q \triangleleft X]$ is legal. All formulas that hold before a message is sent still hold afterwards; the only new development is that the recipient sees the message.

- For sequences of protocol steps: if the annotations $[X]S_1 \ldots [Y]$ and $[Y]S'_1 \ldots [Z]$ are legal then so is $[X]S_1 \ldots [Y]S'_1 \ldots [Z]$. Thus, annotations can be concatenated.
- Logical postulates are used:
  - If X is an assertion (but not the assumptions) in a legal annotation $\mathcal{A}$, if X' is provable from X, and if $\mathcal{A}'$ is the result of substituting X' for X in $\mathcal{A}$, then $\mathcal{A}'$ is a legal annotation. Thus, new assertions can be derived from established ones.
  - If X is the assumptions of a legal annotation $\mathcal{A}$, if X' is provable from X, and if $\mathcal{A}'$ is the result of substituting (X, X') for X in $\mathcal{A}$, then $\mathcal{A}'$ is a legal annotation. Thus, the consequences of the original assumptions can be written down explicitly next to the original assumptions.

A legal annotation of a protocol is much like a sequence of comments about the states of belief of principals in the course of authentication. Step by step, we can follow the evolution from the original assumptions to the conclusions—from the initial beliefs to the final ones.

On time

Note that our logic has not, and does not need, any notion of time to be associated with individual statements. The requirement to deal with time is entirely satisfied by the division of time into past and present, and by the semantics of the constructs themselves. This is possible because we found it sufficient to reason with stable formulas, that is, formulas that stay true for the whole run of the protocol once they become true. In addition, we represent protocols as sequential algorithms and ignore concurrency issues. As in the published protocols, a partial ordering of algorithmic steps is imposed by functional dependence.

It was a conscious decision to avoid the explicit use of time in the logic presented, and we found it unnecessary for describing the protocols investigated so far. We feel that, though this approach might seem simple-minded, it has greatly increased the ease with which the logic can be manipulated. More ambitious proofs may require finer temporal distinctions, reflected by constructs to reason about additional epochs, or even general-purpose temporal operators (see, for example, Nguyen & Perry; Halpern & Vardi 1986). However, it is not clear that such proofs would offer greater insight into the workings of the protocols, nor be simple enough to construct without considerable expertise.

It should be noted that some authentication protocols make use of timestamps, but this does not require time to be made explicit in the logic. As for nonces in general, the only important property of timestamps is whether they are known to be fresh.

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. Occasionally, however, analysis of a protocol suggests that further assumptions are made. 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:

$$A \models A \overset{K}{\leftrightarrow} B$$
$$B \models A \overset{K}{\leftrightarrow} B$$

Some authentication protocols achieve more than this. In particular, many achieve, in addition:

$$A \models B \models A \overset{K}{\leftrightarrow} B$$
$$B \models A \models A \overset{K}{\leftrightarrow} B$$

However, common belief in the goodness of K is never required—that is, A and B need not believe that they both believe that they both believe that ... they both believe that K is good. Some protocols may attain only weaker goals, as for example $A \models B \models X$, for some X, which reflects only that A believes that B has recently sent messages and exists at present.

Some public-key protocols are not intended to result in the exchange of a shared key, but instead transfer some other piece of data. For example, the interaction of a principal with the certification authority might be intended to transfer a single public key:

$$A \models \overset{K}{\mapsto} B$$

In addition, principals may establish some shared secrets:

$$A \models A \overset{N_a}{\rightleftharpoons} B$$

In such 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 Otway-Rees Protocol

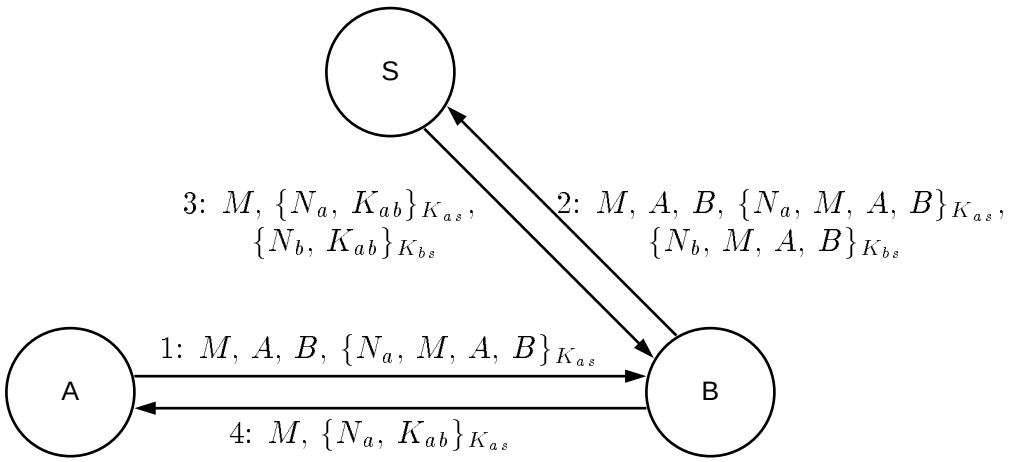
Otway and Rees proposed a shared-key authentication protocol which involves two principals and an authentication server (1987). The protocol is attractive in that it provides good timeliness guarantees in a small number of messages. Moreover, it makes no use of synchronized clocks, and is easily implemented as two nested remote procedure calls. Although not widely used, it is a well designed protocol that may have application in certain environments.

We give the protocol below, with A and B as the two principals, K_{as} and K_{bs} as their private keys, S as the authentication server. The principals A and B generate the nonces N_a, N_b, and M; the server S generates K_{ab}, which becomes the session key between A and B.

Message 1	$A \rightarrow B$:	$M,\ A,\ B,\ \{N_a,\ M,\ A,\ B\}_{K_{as}}$
Message 2	$B \rightarrow S$:	$M,\ A,\ B,\ \{N_a,\ M,\ A,\ B\}_{K_{as}},\ \{N_b,\ M,\ A,\ B\}_{K_{bs}}$
Message 3	$S \rightarrow B$:	$M,\ \{N_a,\ K_{ab}\}_{K_{as}},\ \{N_b,\ K_{ab}\}_{K_{bs}}$
Message 4	$B \rightarrow A$:	$M,\ \{N_a,\ K_{ab}\}_{K_{as}}$

This message sequence is represented in the diagram below.

A passes to B some encrypted material useful only to the server, together with enough information for B to make up a similar encrypted message. B forwards both to the server, who decrypts and checks whether the components M, A, and B match in the encrypted messages. If so, S generates K_{ab} and embeds it in two encrypted messages, one for each participant, accompanied by the appropriate nonces. Both are sent to B, who forwards the appropriate part to A. Then A and B decrypt, check their nonces, and if satisfied proceed to use K_{ab}.



The Otway-Rees Protocol

Now we transform the protocol. The nonce N_c corresponds to M, A, B in the protocol description above.

Message 1	$A \to B$:	$\{N_a,\ N_c\}_{K_{as}}$
Message 2	$B \to S$:	$\{N_a,\ N_c\}_{K_{as}},\ \{N_b,\ N_c\}_{K_{bs}}$
Message 3	$S \to B$:	$\{N_a,\ (A \overset{K_{ab}}{\leftrightarrow} B),\ (B \mid\sim N_c)\}_{K_{as}}$,
		$\{N_b,\ (A \overset{K_{ab}}{\leftrightarrow} B),\ (A \mid\sim N_c)\}_{K_{bs}}$
Message 4	$B \to A$:	$\{N_a,\ (A \overset{K_{ab}}{\leftrightarrow} B),\ (B \mid\sim N_c)\}_{K_{as}}$

The idealized messages correspond quite closely to the messages described in the published protocol. We omit cleartext communication throughout, since it provides no guarantees of any kind.

Further differences can be seen in the idealized forms of messages 3 and 4. 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. For example, when A receives N_a and K_{ab}, he can deduce that the key K_{ab} is intended for use in talking to B, because the nonce N_a appears both in this message and in A's request for such a key.

Even more interesting are the statements $A \mid\sim N_c$ and $B \mid\sim N_c$. These do not appear to correspond to anything in the concrete protocol; they represent the fact that the messages are sent at all, because if the common nonces had not matched nothing would ever have happened.

At this point, we can convince ourselves that the idealized protocol accurately represents the actual one and that the guidelines for constructing idealized protocols are not violated.

The protocol analyzed

To analyze this protocol, we first give the assumptions:

$$
\begin{array}{ll}
A \mid\equiv A \overset{K_{as}}{\leftrightarrow} S & B \mid\equiv B \overset{K_{bs}}{\leftrightarrow} S \\
S \mid\equiv A \overset{K_{as}}{\leftrightarrow} S & S \mid\equiv B \overset{K_{bs}}{\leftrightarrow} S \\
S \mid\equiv A \overset{K_{ab}}{\leftrightarrow} B & \\
A \mid\equiv (S \mid\Rightarrow A \overset{K}{\leftrightarrow} B) & B \mid\equiv (S \mid\Rightarrow A \overset{K}{\leftrightarrow} B) \\
A \mid\equiv (S \mid\Rightarrow (B \mid\sim X)) & B \mid\equiv (S \mid\Rightarrow (A \mid\sim X)) \\
A \mid\equiv \sharp(N_a) & B \mid\equiv \sharp(N_b) \\
A \mid\equiv \sharp(N_c) &
\end{array}
$$

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 which is to become a shared secret between A and B. The next group of four indicates the trust that A and B have in the server to generate a good encryption key and to forward a message from the other client honestly. The final three assumptions show that three nonces have been invented by various principals who consider them to be fresh.

Once we have the assumptions and the idealized version of the protocol, we can proceed to verify it. The rest of the procedure consists merely of applying the postulates of the logic and the annotation rules to the formulas available. It would be excessive to give the detailed deductions that we have checked mechanically—however, we would be happy to provide details of our formal proofs to readers interested in constructing their own. The proof may be briefly outlined as follows.

A sends his message to B. Now B sees the message, but does not understand it:

$$B \triangleleft \{N_a,\ N_c\}_{K_{as}}$$

B is able to generate a message of the same form and to pass it on to S along with A's message. On receiving the message, S can decrypt each encrypted part according to the relevant message-meaning postulate, and so deduce that both A and B have encrypted the nonce N_c in their messages:

$$S \models A \mid\sim (N_a, N_c)$$
$$S \models B \mid\sim (N_b, N_c)$$

Note that S cannot tell whether this message is a replay or not, since there is nothing in the message that he knows to be fresh. S emits a message containing two encrypted messages to B. One of the parts is intended for A, and B passes it on.

At this point, both A and B have received a message from the server containing a new encryption key and a nonce. We successively apply the postulates on message meaning, nonce verification, and jurisdiction, and emerge with the following final beliefs:

$$A \models A \overset{K_{ab}}{\leftrightarrow} B \qquad B \models A \overset{K_{ab}}{\leftrightarrow} B$$
$$A \models B \models N_c \qquad B \models A \mid\sim N_c$$

It is interesting to note that this protocol does not make use of K_{ab} as an encryption key, so neither principal can know whether the key is known to the other. A is in a slightly better position than B, in that A has been told that B emitted a message containing a nonce that A believes to be fresh. This allows A to infer that B has sent a message recently—B exists. B has been told by the server that A has used a nonce, but B has no idea whether this is a replay of an old message or not.

In addition, we may notice that there are various forms of redundancy in the protocol. Two nonces are generated by A; however the verification using N_a could just as well have been done using N_c. Therefore, N_a can be eliminated, so reducing the amount of encryption in the protocol. Moreover, it is clear from the analysis that N_b need not be encrypted in the second message. As these possibilities are explored, we rapidly move towards an improved protocol of different structure.

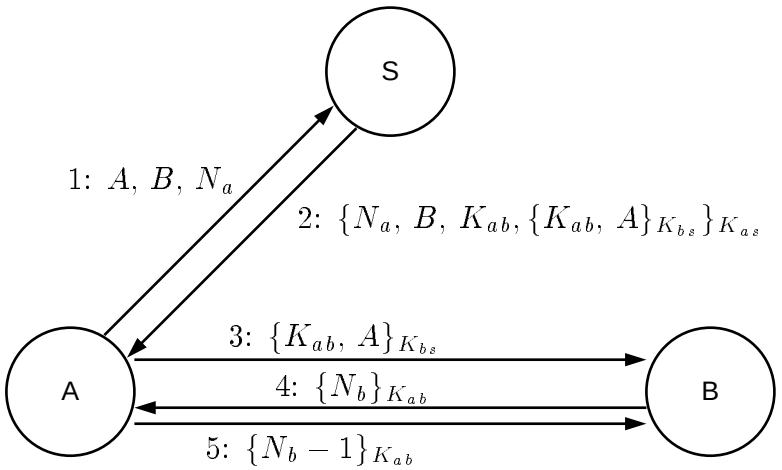
5. The Needham-Schroeder Protocol (with shared keys)

Many existing authentication protocols are derived from the Needham-Schroeder protocol (1978). The original protocol is interesting, both because so much work has been based on it, and also because it has a serious weakness.

This protocol has the same cast of players as the Otway-Rees protocol.

Message 1	$A \rightarrow S$:	$A,\ B,\ N_a$
Message 2	$S \rightarrow A$:	$\{N_a,\ B,\ K_{ab}, \{K_{ab},\ A\}_{K_{bs}}\}_{K_{as}}$
Message 3	$A \rightarrow B$:	$\{K_{ab},\ A\}_{K_{bs}}$
Message 4	$B \rightarrow A$:	$\{N_b\}_{K_{ab}}$
Message 5	$A \rightarrow B$:	$\{N_b - 1\}_{K_{ab}}$

Here only A makes contact with the server, who provides A with the session key, K_{ab}, and a certificate encrypted with B's key conveying the session key and A's identity to B. Then B decrypts this certificate and carries out a nonce handshake with A to be assured that A is present currently, since the certificate might have been a replay. The use of $N_b - 1$ in the last message is conventional. Almost any function of N_b would do, as long as B can distinguish his message from A's—thus, subtraction is used to indicate that the message is from A, rather than from B.



The Needham-Schroeder Protocol (with shared keys)

The idealized protocol is as follows:

Message 2	$S \rightarrow A$:	$\{N_a,\ (A \overset{K_{ab}}{\leftrightarrow} B),\ \sharp(A \overset{K_{ab}}{\leftrightarrow} B),\ \{A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}\}_{K_{as}}$
Message 3	$A \rightarrow B$:	$\{A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}$

Message 4 $B \rightarrow A$: $\{N_b, (A \overset{K_{ab}}{\leftrightarrow} B)\}_{K_{ab}}$ *from* B

Message 5 $A \rightarrow B$: $\{N_b, (A \overset{K_{ab}}{\leftrightarrow} B)\}_{K_{ab}}$ *from* A

The first message is omitted, since it does not contribute to the logical properties of the protocol. The result is as if S acted spontaneously. The last two messages of the idealized protocol are written in full, including the senders' names. This is merely to distinguish the two messages, which might otherwise be confused. In this case, the concrete realization of this distinction is, of course, the subtraction in the final message.

The additional statements about the key K_{ab} in messages 2, 4, and 5 are present to assure A that the key can be used as a nonce and to assure each principal that the other believes the key is good. These statements can be included because neither message would have been sent if the statements were not believed.

The protocol analyzed

To start, we give some assumptions:

$$
\begin{array}{ll}
A \mid\equiv A \overset{K_{as}}{\leftrightarrow} S & B \mid\equiv B \overset{K_{bs}}{\leftrightarrow} S \\
S \mid\equiv A \overset{K_{as}}{\leftrightarrow} S & S \mid\equiv B \overset{K_{bs}}{\leftrightarrow} S \\
S \mid\equiv A \overset{K_{ab}}{\leftrightarrow} B & \\
A \mid\equiv (S \mid\Rightarrow A \overset{K}{\leftrightarrow} B) & B \mid\equiv (S \mid\Rightarrow A \overset{K}{\leftrightarrow} B) \\
A \mid\equiv (S \mid\Rightarrow \sharp(A \overset{K}{\leftrightarrow} B)) & \\
A \mid\equiv \sharp(N_a) & B \mid\equiv \sharp(N_b) \\
S \mid\equiv \sharp(A \overset{K_{ab}}{\leftrightarrow} B) & B \mid\equiv \sharp(A \overset{K}{\leftrightarrow} B)
\end{array}
$$

Most of the assumptions are routine. The first group of five describes the keys initially known to the principals. The next three indicate exactly what the clients trust the server to do. As before, S is trusted to make new keys for A and B, but here A also trusts S to generate a key which has the properties of a nonce. In fact, one can argue that a good encryption key is very likely to make a good nonce in any case. However, the need for this assumption has highlighted the need for this feature in the protocol.

The last assumption, $B \mid\equiv \sharp(A \overset{K}{\leftrightarrow} B)$, is unusual. As discussed below, the protocol has been criticized for using this assumption, and the authors did not realize they were making it. The proof outlined here shows how this added assumption is needed to attain authentication.

Again the detail in the verification is suppressed. First, A sends a cleartext message containing a nonce. This can be seen by the server, who repeats the nonce in the reply. The reply from S also contains the new key to be used between A and B. Then A sees the entire message,

$$A \triangleleft \{N_a,\ (A \overset{K_{ab}}{\leftrightarrow} B),\ \sharp(A \overset{K_{ab}}{\leftrightarrow} B),\ \{A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}\}_{K_{as}}$$

which he decrypts. Since A knows N_a to be fresh, we can also apply the nonce-verification postulate, leading to:

$$A \models S \models A \overset{K_{ab}}{\leftrightarrow} B$$
$$A \models S \models \sharp(A \overset{K_{ab}}{\leftrightarrow} B)$$

The jurisdiction postulate enables us to infer:

$$A \models A \overset{K_{ab}}{\leftrightarrow} B$$
$$A \models \sharp(A \overset{K_{ab}}{\leftrightarrow} B)$$

Also, A has seen the part of the message encrypted under B's private key,

$$A \triangleleft \{A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}$$

This allows A to send this as a message to B. At this point, B decrypts the message, and from the appropriate message-meaning postulate, we obtain:

$$B \models S \mid\sim A \overset{K_{ab}}{\leftrightarrow} B$$

Unlike A, however, B is unable to proceed unless we resort to the dubious assumption set out above. B knows of nothing in the message which is fresh, so he cannot tell when this message was generated. B simply assumes that the message from the server is fresh.

If we make the necessary assumption, the rest of the protocol proceeds without any problem. We immediately obtain

$$B \models A \overset{K_{ab}}{\leftrightarrow} B$$

via the postulates of nonce verification and jurisdiction.

The last two messages cause A and B each to become convinced that the other exists (that is, he has sent messages recently) and is in possession of the key. B first encrypts his nonce and sends it to A, who can deduce that B believes in the key,

$$A \models B \models A \overset{K_{ab}}{\leftrightarrow} B$$

because he has been guaranteed the freshness of the key by S. Then A replies similarly, and B can deduce that A also believes in the key,

$$B \models A \models A \overset{K_{ab}}{\leftrightarrow} B$$

Note that the freshness of the nonce N_b is sufficient for B to deduce this. It is not necessary to reuse the dubious assumption.

This results in the following beliefs:

$$A \models A \overset{K_{ab}}{\leftrightarrow} B \qquad B \models A \overset{K_{ab}}{\leftrightarrow} B$$
$$A \models B \models A \overset{K_{ab}}{\leftrightarrow} B \qquad B \models A \models A \overset{K_{ab}}{\leftrightarrow} B$$

In fact, we could extend the idealized protocol, adding $A \models B \models A \overset{K_{ab}}{\leftrightarrow} B$ to the last message, and obtain even

$$B \models A \models B \models A \overset{K_{ab}}{\leftrightarrow} B$$

This conclusion, however, seems to be of little importance for the subsequent use of the session key.

This is a stronger outcome than in the Otway-Rees protocol, but it is reached at the cost of the extra assumption that B accepts the key as new. Denning and Sacco pointed out that compromise of a session key can have very bad results: an intruder has unlimited time to find an old session key and to reuse it as though it were fresh (1981). Bauer, Berson, and Feiertag pointed out that there are even more drastic consequences if A's private key is compromised: an intruder can use A's key to obtain session keys to talk to many other principals, and can continue to use these session keys even after A's key has been changed (1983). It is comforting that the logical analysis makes explicit the assumption.

Clearly, the problem is that B has no interaction with S that starts with B's initiative. It is possible to rectify this by starting with B rather than A, and this was done by Needham and Schroeder (1987). The note by Needham and Schroeder was published adjacent to the paper by Otway and Rees. Perhaps for the lack of a calculus to describe these protocols, none of the people involved realized that the proposals were essentially the same. The only significant difference is that the second Needham-Schroeder protocol goes on to use the session key (K_{ab}) explicitly, thereby assuring each principal that the other knows the key, while Otway and Rees allow these final stages to be combined with the first transmissions of data.

A slight peculiarity in the original Needham-Schroeder protocol is that the certificate $\{K_{ab}, A\}_{K_{bs}}$ is encrypted with A's key in the second message. Looking back through the formal analysis, one sees that this does not affect the properties of the protocol, since the certificate is sent to B immediately afterwards without further encryption. It may also be noticed that in message 4 a nonce is being sent encrypted when this is in general not necessary. However, in this instance, if the nonce were sent unencrypted it would be necessary to send something else encrypted in order for A to deduce that B knows the session key.

6. The Kerberos Protocol

The Kerberos protocol was developed as part of Project Athena at MIT (Miller *et al.* 1987), and is now being used by a number of other organizations. It is based on the Needham-Schroeder protocol, but makes use of timestamps as nonces to remove the problems shown in the last section and to reduce the total number of messages required.

A slightly simplified version of the protocol is shown below:

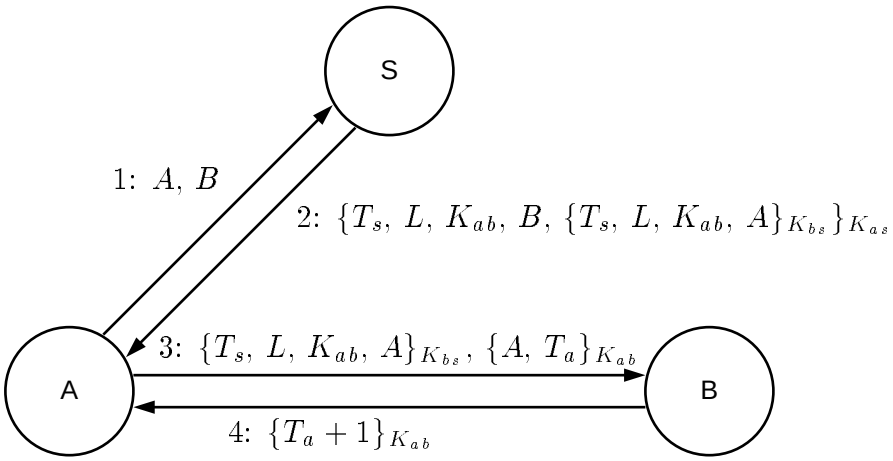
Message 1	$A \rightarrow S$:	$A,\ B$
Message 2	$S \rightarrow A$:	$\{T_s,\ L,\ K_{ab},\ B,\ \{T_s,\ L,\ K_{ab},\ A\}_{K_{bs}}\}_{K_{as}}$
Message 3	$A \rightarrow B$:	$\{T_s,\ L,\ K_{ab},\ A\}_{K_{bs}},\ \{A,\ T_a\}_{K_{ab}}$
Message 4	$B \rightarrow A$:	$\{T_a + 1\}_{K_{ab}}$

Here, T_s and T_a are timestamps, and L is a lifetime. The protocol description states that the fourth message is used only if mutual authentication is required.

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

As in the Needham-Schroeder protocol, the first message is omitted, since it does not contribute to the logical properties of the protocol. For simplicity, the lifetime L has been combined with the timestamp T_s, and is treated just like a nonce.



The Kerberos Protocol

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 suffice to distinguish the two. There is no obvious reason for the use both features, though it seems likely that the second was inherited from the Needham-Schroeder protocol, in which a nonce is incremented in order to distinguish two messages that are otherwise identical.

The protocol analyzed

First we write the assumptions, which are quite standard:

$$
\begin{array}{ll}
A \models A \overset{K_{as}}{\leftrightarrow} S & B \models B \overset{K_{bs}}{\leftrightarrow} S \\
S \models A \overset{K_{as}}{\leftrightarrow} S & S \models B \overset{K_{bs}}{\leftrightarrow} S \\
S \models A \overset{K_{ab}}{\leftrightarrow} B & \\
A \models (S \models\!\Rightarrow A \overset{K}{\leftrightarrow} B) & B \models (S \models\!\Rightarrow A \overset{K}{\leftrightarrow} B) \\
A \models \sharp(T_s) & B \models \sharp(T_s) \\
 & B \models \sharp(T_a)
\end{array}
$$

It is clear from the last three assumptions that the protocol relies heavily on the use of synchronized clocks, since each principal believes that timestamps generated elsewhere

are fresh.

The analysis is straightforward. A receives message 2, and from the rules of message meaning and nonce verification, we obtain:

$$A \models A \overset{K_{ab}}{\leftrightarrow} B$$
$$A \triangleleft \{T_s,\ A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}$$

A passes on the encrypted message from the authentication server, together with another message containing a timestamp. Initially, B can decrypt only one part:

$$B \models A \overset{K_{ab}}{\leftrightarrow} B$$

Knowledge of the new key allows B to decrypt the rest of the message; we thus obtain:

$$B \models A \models A \overset{K_{ab}}{\leftrightarrow} B$$

The fourth message simply assures A that B believes in the key, and received A's last message. The final result is:

$$A \models A \overset{K_{ab}}{\leftrightarrow} B \qquad B \models A \overset{K_{ab}}{\leftrightarrow} B$$
$$A \models B \models A \overset{K_{ab}}{\leftrightarrow} B \qquad B \models A \models A \overset{K_{ab}}{\leftrightarrow} B$$

If only the first three messages are used, $A \models B \models A \overset{K_{ab}}{\leftrightarrow} B$ is not generated. That is, the three-message protocol does not convince A of B's existence. As with the Needham-Schroeder protocol, we can obtain

$$A \models B \models A \models A \overset{K_{ab}}{\leftrightarrow} B$$

by adding $B \models A \models A \overset{K_{ab}}{\leftrightarrow} B$ to the fourth message of the idealized protocol. Again, this conclusion seems to be of little importance for the subsequent use of the session key.

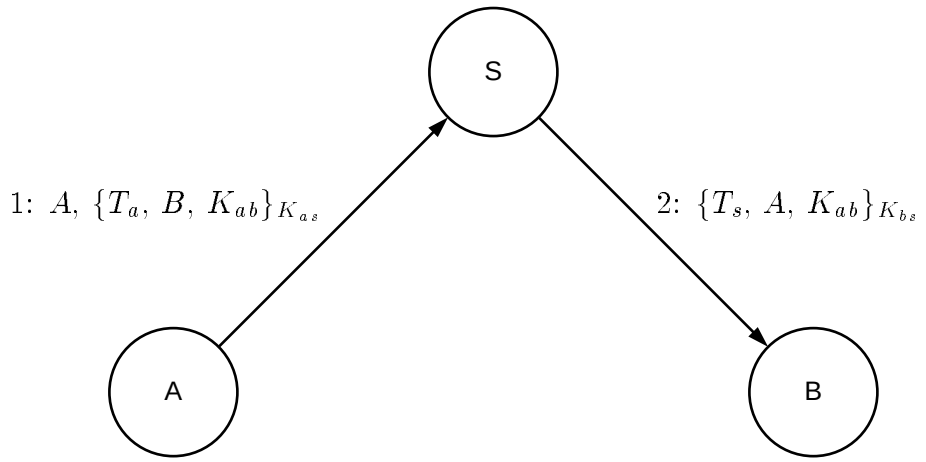
This result is similar to that for the Needham-Schroeder protocol. However, a major assumption in the Kerberos protocol is that the principals' clocks are synchronized with the server's clock. In practice, the effect of totally synchronized clocks is obtained by synchronizing clocks to within a few minutes with a secure time server, and detecting replays within the synchronization interval. However, actual implementations do not always include this check and so provide only weaker guarantees. Also, as in the Needham-Schroeder protocol, we can obtain the same final beliefs without the double encryption in message 2.

7. The Wide-mouthed-frog Protocol

The following unpublished protocol was proposed by one of us (M.B.), and is perhaps the simplest protocol that uses shared-key cryptography and an authentication server. It transfers a key from A to B via S in only two messages by using synchronized clocks, and by allowing A to choose the session key:

Message 1 $A \rightarrow S$: $A,\ \{T_a,\ B,\ K_{ab}\}_{K_{as}}$

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



The Wide-mouthed-frog Protocol

A sends a session key to S, including a timestamp T_a. S checks that the first message is timely, and if it is, it forwards the message to B, together with its own timestamp T_s. B then checks that the timestamp from S is later than any other it has received from S. Since all timestamps are either checked or generated by S, each principal need only record the difference between his own clock and S's clock, updating it whenever clock drift causes an authentication message to be rejected.

The idealized protocol is shown below:

Message 1 $A \rightarrow S$: $\{T_a,\ (A \overset{K_{ab}}{\leftrightarrow} B)\}_{K_{as}}$

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

The protocol analyzed

The assumptions are as follows:

$$\begin{array}{ll} A \models A \overset{K_{as}}{\leftrightarrow} S & B \models B \overset{K_{bs}}{\leftrightarrow} S \\ S \models A \overset{K_{as}}{\leftrightarrow} S & S \models B \overset{K_{bs}}{\leftrightarrow} S \\ A \models A \overset{K_{ab}}{\leftrightarrow} B & B \models (A \Rightarrow A \overset{K}{\leftrightarrow} B) \\ & B \models (S \Rightarrow A \models A \overset{K}{\leftrightarrow} B) \\ S \models \sharp(T_a) & B \models \sharp(T_s) \end{array}$$

As in the Kerberos protocol, synchronized clocks are required. The more unusual assumptions are that A knows the session key in advance, and that B trusts A to invent good keys.

The analysis is almost trivial. S receives the first message, and we deduce:

$$S \models A \models A \overset{K_{ab}}{\leftrightarrow} B$$

S then sends the second message, which B receives and interprets. From the rules of message meaning, nonce verification, and jurisdiction, we obtain the final beliefs:

$$\begin{array}{ll} A \models A \overset{K_{ab}}{\leftrightarrow} B & B \models A \overset{K_{ab}}{\leftrightarrow} B \\ & B \models A \models A \overset{K_{ab}}{\leftrightarrow} B \end{array}$$

The most dubious assumption in this protocol is that B trusts A to invent good keys. None of the protocols presented in this paper attempt to deal with malice on the part of one of the principals—either principal could simply reveal his keys to a third party. However, in this case, A is being trusted to be *competent*, rather than to be non-malicious. If the generation of session keys requires more care than A is expected to take, one might be unwilling to use this protocol. Of course, this assumption can be avoided, but only at the cost of an extra message, which allows B to pass on to A a session key invented by S.

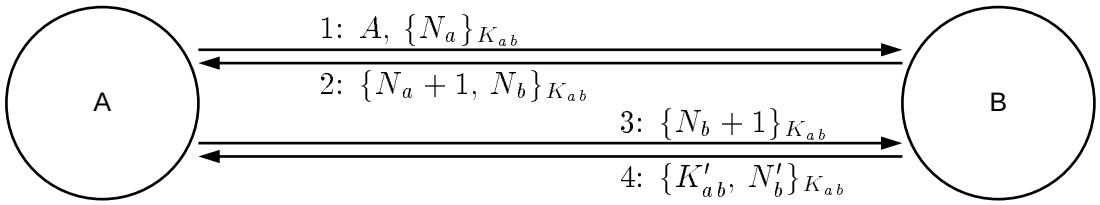
8. The Andrew Secure RPC Handshake

An early version of the Andrew secure RPC protocol uses an authentication handshake between two principals whenever a client binds to a new server (Satyanarayanan 1987). The handshake is intended to allow a client, which we call 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 Needham-Schroeder protocol; it is no longer used.

We discuss it here as an illustration of how easily such problems can be missed, and of how they manifest themselves in the logic.

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 Andrew Secure RPC Handshake

The idealized protocol closely resembles the real 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	$B \rightarrow A$:	$\{A \overset{K'_{ab}}{\leftrightarrow} B,\ N'_b\}_{K_{ab}}$

The protocol analyzed

First, we write the assumptions:

$$A \mid\equiv A \overset{K_{ab}}{\leftrightarrow} B \qquad B \mid\equiv A \overset{K_{ab}}{\leftrightarrow} B$$
$$A \mid\equiv (B \mid\Rightarrow A \overset{K}{\leftrightarrow} B) \qquad B \mid\equiv A \overset{K'_{ab}}{\leftrightarrow} B$$
$$A \mid\equiv \sharp(N_a) \qquad B \mid\equiv \sharp(N_b)$$
$$B \mid\equiv \sharp(N'_b)$$

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.

We can now proceed with the analysis. The first message gives, simply:

$$B \mid\equiv A \mid\sim N_a$$
$$B \triangleleft N_a$$

The second message contains a nonce that A believes is fresh, so we can deduce:

$$A \mid\equiv B \mid\equiv (N_a, N_b)$$

Hence, A knows that B exists. Similarly, the third message gives us:

$$B \mid\equiv A \mid\equiv N_b$$

B, now assured of A's existence, sends the new key, and we obtain the final beliefs:

$$B \mid\equiv A \overset{K'_{ab}}{\leftrightarrow} B$$
$$A \mid\equiv B \mid\sim (A \overset{K'_{ab}}{\leftrightarrow} B,\ N'_b)$$
$$B \mid\equiv A \mid\equiv N_b$$
$$A \mid\equiv B \mid\equiv (N_a,\ N_b)$$

Unfortunately, we cannot proceed further. We cannot obtain $A \mid\equiv B \mid\equiv A \overset{K'_{ab}}{\leftrightarrow} B$ 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 convince A to use an old, possibly compromised session key. This problem is similar to that found in the Needham-Schroeder protocol. Fortunately, the problem can be fixed simply by adding the nonce N_a to the last message:

Message 4 $\quad B \rightarrow A$: $\quad \{K'_{ab},\ N'_b,\ N_a\}_{K_{ab}}$

In fact, more substantial changes to the protocol can also reduce the total amount of encryption needed. The nonces are not used as secrets, and so need not be encrypted when first mentioned. Similarly, the starting sequence number N'_b could just as easily be sent in clear, or identified with one of the other nonces.

We arrive at a stronger protocol than the original one, but use less encryption. It suffices for B to send a key K'_{ab} (along with a nonce N_a),

$$B \to A: \quad \{N_a,\ A \overset{K'_{ab}}{\leftrightarrow} B\}_{K_{ab}}$$

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 acknowledgement that K'_{ab} has been accepted,

$$A \to B: \quad \{A \overset{K'_{ab}}{\leftrightarrow} B\}_{K'_{ab}}$$

B believes this message is timely because K'_{ab} is fresh, $B \models \sharp(A \overset{K'_{ab}}{\leftrightarrow} B)$. Optionally, B can go on to send an initial sequence number N'_b in clear.

If this new protocol is analyzed, we obtain the final beliefs:

$$\begin{array}{ll} A \models A \overset{K'_{ab}}{\leftrightarrow} B & B \models A \overset{K'_{ab}}{\leftrightarrow} B \\ A \models B \models A \overset{K'_{ab}}{\leftrightarrow} B & B \models A \models A \overset{K'_{ab}}{\leftrightarrow} B \end{array}$$

As a concrete realization of the protocol, we propose:

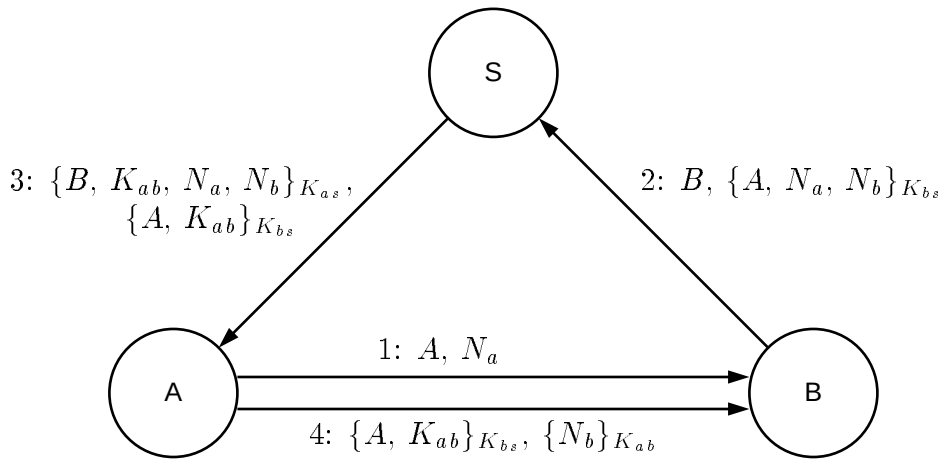
$$\begin{array}{lll} \text{Message 1} & A \to B: & A,\ N_a \\ \text{Message 2} & B \to A: & \{N_a,\ K'_{ab}\}_{K_{ab}} \\ \text{Message 3} & A \to B: & \{N_a\}_{K'_{ab}} \\ \text{Message 4} & B \to A: & N'_b \end{array}$$

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.

9. The Yahalom Protocol

The logic as we have described it serves in the analysis of a variety of protocols. On occasion, however, we have had to integrate new mechanisms for reasoning about some intriguing protocols. This section exemplifies how we adapt the logic, in this case to handle uncertified keys, and also illustrates the use of secrets. Once more, the logic guides us in understanding the operation of a protocol and in suggesting improvements to it. We consider the following ingenious protocol, due to Yahalom (personal communication 1988).

Message 1	$A \rightarrow B$:	$A,\ N_a$
Message 2	$B \rightarrow S$:	$B,\ \{A,\ N_a,\ N_b\}_{K_{bs}}$
Message 3	$S \rightarrow A$:	$\{B,\ K_{ab},\ N_a,\ N_b\}_{K_{as}},\ \{A,\ K_{ab}\}_{K_{bs}}$
Message 4	$A \rightarrow B$:	$\{A,\ K_{ab}\}_{K_{bs}},\ \{N_b\}_{K_{ab}}$



The Yahalom Protocol

The cast is the usual one. The novelty here is that the unusual sequence of messages results in strong guarantees for both A and B with few messages: A sends a nonce N_a to S, indirectly, then gets it back with the key K_{ab}, and B sends a nonce N_b to S, then gets it back with the key K_{ab}, indirectly.

As the idealized version of this protocol, we propose:

$$\begin{array}{lll}
\text{Message 2} & B \to S: & \{N_a,\ N_b\}_{K_{bs}} \\
\text{Message 3} & S \to A: & \{A \overset{K_{ab}}{\leftrightarrow} B,\ \sharp(A \overset{K_{ab}}{\leftrightarrow} B),\ N_a,\ N_b,\ B \mid\sim N_a\}_{K_{as}}, \\
 & & \{A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}} \\
\text{Message 4} & A \to B: & \{A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}},\ \{\langle N_b,\ A \overset{K_{ab}}{\leftrightarrow} B,\ S \equiv \sharp(A \overset{K_{ab}}{\leftrightarrow} B)\rangle_{N_b}\}_{K_{ab}}
\end{array}$$

The first message is omitted, since it does not affect the logical analysis. The result is as if B acted spontaneously. In the third and fourth messages it is necessary to make explicit that the server asserts that the key K_{ab} is fresh; this does not alter the essence of the protocol in any significant way—a good key should probably share the properties of a nonce. An unusual feature of the protocol is the use of N_b as a shared secret in the fourth message.

The protocol analyzed

Most of the initial assumptions here are the same as for the Otway-Rees and Needham-Schroeder protocols:

$$\begin{array}{ll}
A \equiv A \overset{K_{as}}{\leftrightarrow} S & B \equiv B \overset{K_{bs}}{\leftrightarrow} S \\
S \equiv A \overset{K_{as}}{\leftrightarrow} S & S \equiv B \overset{K_{bs}}{\leftrightarrow} S \\
S \equiv A \overset{K_{ab}}{\leftrightarrow} B & \\
A \equiv (S \Rightarrow A \overset{K}{\leftrightarrow} B) & B \equiv (S \Rightarrow A \overset{K}{\leftrightarrow} B) \\
A \equiv \sharp(N_a) & B \equiv \sharp(N_b)
\end{array}$$

However, several additional assumptions are also made in this protocol:

$$\begin{array}{l}
S \equiv \sharp(A \overset{K_{ab}}{\leftrightarrow} B) \\
B \equiv (S \Rightarrow \sharp(A \overset{K}{\leftrightarrow} B)) \\
B \equiv (A \Rightarrow S \equiv \sharp(A \overset{K}{\leftrightarrow} B)) \\
A \equiv (S \Rightarrow B \mid\sim N) \\
B \equiv A \overset{N_b}{\rightleftharpoons} B
\end{array}$$

The first three additional assumptions are needed because B must know that the session key is fresh in order to deduce the timeliness of the fourth message. The next represents A's trust in S to pass on a nonce from B. The last assumption is interesting in that it was surprising to the author of the protocol. The protocol actually uses N_b as a shared secret, although it was thought of simply as a nonce when the protocol was designed.

The second message produces:

$$S \models B \mid\sim (N_a,\ N_b)$$

From the third message, using the message-meaning, nonce-verification, and jurisdiction rules, we derive:

$$A \triangleleft N_b$$
$$A \models A \overset{K_{ab}}{\leftrightarrow} B$$
$$A \models S \models \sharp(A \overset{K_{ab}}{\leftrightarrow} B)$$
$$A \models B \models N_a$$

The fourth message is more complicated. We obtain:

$$B \models S \mid\sim A \overset{K_{ab}}{\leftrightarrow} B$$
$$B \triangleleft A \overset{K_{ab}}{\leftrightarrow} B$$

However, we cannot proceed further. The protocol leads B to use the key K_{ab} before ascertaining that it is a good key. That it is a good key is confirmed only after its first use, with the second part of the fourth message. We discuss below how we extend the logic to handle uncertified keys. For now, we proceed as if we could derive:

$$B \triangleleft \langle N_b,\ A \overset{K_{ab}}{\leftrightarrow} B,\ S \models \sharp(A \overset{K_{ab}}{\leftrightarrow} B)\rangle_{N_b}$$

Notice that use of the uncertified key allows B to see the contents of the message, but not to deduce the sender. Since B believes N_b to be both a secret and fresh, we have:

$$B \models A \models (A \overset{K_{ab}}{\leftrightarrow} B,\ S \models \sharp(A \overset{K_{ab}}{\leftrightarrow} B))$$

B trusts both S and A in statements about the freshness of the key, so:

$$B \models \sharp(A \overset{K_{ab}}{\leftrightarrow} B)$$

This allows us to perform nonce verification on the other part of the message, leading eventually to:

$$B \models A \overset{K_{ab}}{\leftrightarrow} B$$

So the final beliefs are:

$$A \models A \overset{K_{ab}}{\leftrightarrow} B \qquad B \models A \overset{K_{ab}}{\leftrightarrow} B$$

$$A \models B \models N_a \qquad B \models A \models A \overset{K_{ab}}{\leftrightarrow} B$$

The session key has been distributed, and each principal is aware that the other exists.

The analysis shows two interesting points: N_b is used as a shared secret, and B trusts A to pass on a statement about the freshness of the key. If A chose to replay an old key to B in message 4, B could not detect the fraud. This does not represent a major flaw in the protocol, since the principals are assumed not to be malicious, but merely highlights an assumption that was not clear at first.

A simple change to the protocol removes these features, strengthening the protocol and simplifying the analysis at the same time. The concrete protocol becomes:

Message 1	$A \to B$:	$A,\ N_a$
Message 2	$B \to S$:	$B,\ N_b,\ \{A,\ N_a\}_{K_{bs}}$
Message 3	$S \to A$:	$N_b,\ \{B,\ K_{ab},\ N_a\}_{K_{as}},\ \{A,\ K_{ab},\ N_b\}_{K_{bs}}$
Message 4	$A \to B$:	$\{A,\ K_{ab},\ N_b\}_{K_{bs}},\ \{N_b\}_{K_{ab}}$

In the analysis of this variant, there is no need to use an uncertified key, because the timeliness of the last message is guaranteed by the nonce N_b. Moreover, N_b no longer need be kept secret, so it need not be encrypted in message 2 and the first half of message 3. The resulting protocol has the same outcome, but with less encryption and considerably fewer assumptions.

On using uncertified keys

As was discovered above, the logic presented so far does not provide any mechanism for decryption with keys that are not known to be good. Formally, the message-meaning rules and the rules for $\triangleleft$ apply only to keys believed good for some specified principal.

Fortunately, it is straightforward to modify the logic and remedy this situation. It suffices to make the rules for $\triangleleft$ more liberal; concretely, we may supplement the rule

$$\frac{P \models Q \overset{K}{\leftrightarrow} P, \quad P \triangleleft \{X\}_K}{P \triangleleft X}$$

with the rule

$$\frac{P \models R \mathrel{|\!\sim} Q \overset{K}{\leftrightarrow} P, \quad P \triangleleft \{X\}_K}{P \triangleleft X}$$

Intuitively, this new rule reflects the fact that P may try any key mentioned to him. The proposed solution handles this particular protocol, but it fails to handle other more intricate protocols. In fact, there seems to be no natural place to stop in making the logic stronger and stronger in this sense, by changing the rules for $\triangleleft$.

A more thorough solution requires a fairly serious change in the logic, which we describe briefly. Protocol descriptions can be changed to include not only a sequence of message exchanges but explicit statements for decryption. It becomes the responsibility of the protocol writer to say which keys should be tried for decryptions. The rules for annotations would then be extended, to say that the decrypter sees the inside of the messages he decrypts. The additional complexity of this solution seems unnecessary, since none of the published protocols we have encountered requires it.

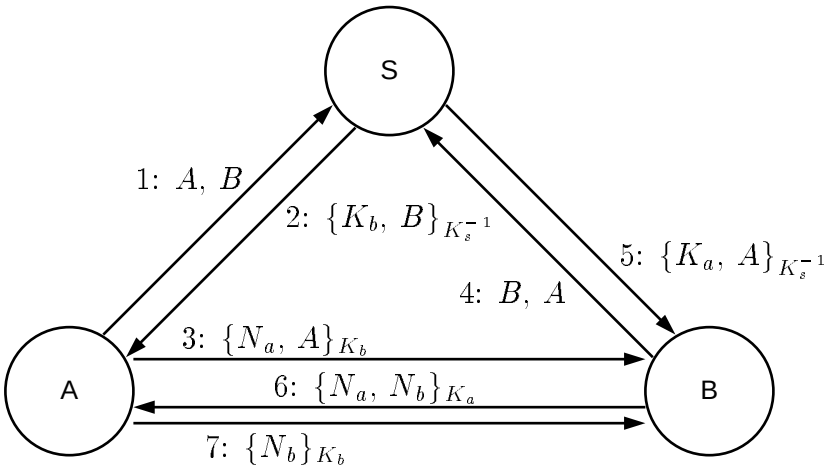
10. The Needham-Schroeder Public-Key Protocol

In their original paper, Needham and Schroeder also proposed a protocol based on public-key cryptography (1978). This protocol allows two principals to exchange two independent, secret numbers. This is unlike most other public-key protocols, which allow the principals to exchange a shared session key. A weakness in the protocol allows a replay attack in the interactions with the certification authority if a key is compromised, much as in the Needham-Schroeder protocol discussed in section 5.

The cast remains the usual one, but S operates only as a certification authority; K_a, K_b, and K_s are the public keys of A, B, and S, respectively; K_s^{-1} is the secret key that matches K_s; finally, N_a and N_b are nonce identifiers.

The message exchange goes as follows:

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 Needham-Schroeder Public-Key Protocol

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	$S \to A$:	$\{\overset{K_b}{\mapsto} B\}_{K_s^{-1}}$
Message 3	$A \to B$:	$\{N_a\}_{K_b}$
Message 5	$S \to B$:	$\{\overset{K_a}{\mapsto} A\}_{K_s^{-1}}$
Message 6	$B \to A$:	$\{\langle A \overset{N_b}{\rightleftharpoons} B\rangle_{N_a}\}_{K_a}$
Message 7	$A \to B$:	$\{\langle A \overset{N_a}{\rightleftharpoons} B,\ B \models (A \overset{N_b}{\rightleftharpoons} B)\rangle_{N_b}\}_{K_b}$

Messages 1 and 4 are deliberately omitted, since they do not contribute to the logical properties of the protocol. The idealized protocol proceeds as if S contacted A and B spontaneously.

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 so 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.

The protocol analyzed

First we state the assumed initial beliefs of the players:

$$
\begin{array}{ll}
A \mid\equiv \stackrel{K_a}{\mapsto} A & B \mid\equiv \stackrel{K_b}{\mapsto} B \\
A \mid\equiv \stackrel{K_s}{\mapsto} S & B \mid\equiv \stackrel{K_s}{\mapsto} S \\
S \mid\equiv \stackrel{K_a}{\mapsto} A & S \mid\equiv \stackrel{K_b}{\mapsto} B \\
S \mid\equiv \stackrel{K_s}{\mapsto} S & \\
A \mid\equiv (S \mid\Rightarrow \stackrel{K}{\mapsto} B) & B \mid\equiv (S \mid\Rightarrow \stackrel{K}{\mapsto} A) \\
A \mid\equiv \sharp(N_a) & B \mid\equiv \sharp(N_b) \\
A \mid\equiv A \stackrel{N_a}{\rightleftharpoons} B & B \mid\equiv A \stackrel{N_b}{\rightleftharpoons} B \\
A \mid\equiv \sharp(\stackrel{K_b}{\mapsto} B) & B \mid\equiv \sharp(\stackrel{K_a}{\mapsto} A)
\end{array}
$$

Each principal knows the public key of the certification agent S, as well as his own keys. In addition, S knows his own keys, and 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 identifier that he generates is fresh, and secret.

The last two assumptions are surprising, and they 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.

Once the assumptions are made, we apply the rules of message meaning and jurisdiction to message 2, to deduce:

$$A \mid\equiv \stackrel{K_b}{\mapsto} B$$

B can deduce little from message 3, since he cannot know who sent the message. We obtain simply:

$$B \triangleleft N_a$$

Message 5 is similar to message 2, giving:

$$B \models \overset{K_a}{\mapsto} A$$

In message 6, B uses the shared secret N_a to convince A that the message is from B, and that the message is timely, giving:

$$A \models B \models A \overset{N_b}{\rightleftharpoons} B$$

A is now able to use N_b in message 7. By the arrival of this message, B can deduce A's current beliefs about the state of the protocol, even though they have no representation in the concrete protocol. Thus, the final beliefs are:

$$\begin{array}{ll} A \models \overset{K_b}{\mapsto} B & B \models \overset{K_a}{\mapsto} A \\ A \models B \models A \overset{N_b}{\rightleftharpoons} B & B \models A \models A \overset{N_a}{\rightleftharpoons} B \\ & B \models A \models B \models A \overset{N_b}{\rightleftharpoons} B \end{array}$$

Each principal knows the public key of the other, and has knowledge of a shared secret which he believes the other will accept as being shared only by the two principals. B has gained slightly more knowledge through being the last recipient of a message in the protocol. 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.

Again, we could extend the idealized protocol, adding $\overset{K_a}{\mapsto} A$ and $\overset{K_b}{\mapsto} B$, $B \models \overset{K_a}{\mapsto} A$ to messages 6 and 7, respectively. We would then obtain

$$\begin{array}{ll} A \models B \models \overset{K_a}{\mapsto} A & B \models A \models \overset{K_b}{\mapsto} B \\ & B \models A \models B \models \overset{K_a}{\mapsto} A \end{array}$$

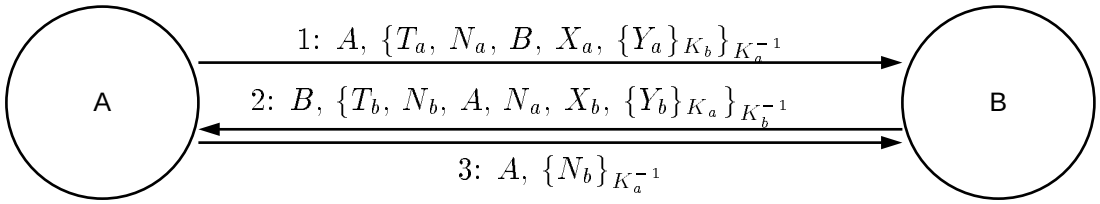
These beliefs have no importance for subsequent communication.

11. 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 (Comité Consultatif International Télégraphique et Téléphonique (CCITT) 1987). 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.

Unfortunately, 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 weaknesses while idealizing the protocol and the other during the subsequent analysis. The protocol shown here has been simplified slightly, in a way which is discussed later.

$$\begin{array}{lll} \text{Message 1} & A \rightarrow B: & A,\ \{T_a,\ N_a,\ B,\ X_a,\ \{Y_a\}_{K_b}\}_{K_a^{-1}} \\ \text{Message 2} & B \rightarrow A: & B,\ \{T_b,\ N_b,\ A,\ N_a,\ X_b,\ \{Y_b\}_{K_a}\}_{K_b^{-1}} \\ \text{Message 3} & A \rightarrow B: & A,\ \{N_b\}_{K_a^{-1}} \end{array}$$



The CCITT X.509 Protocol

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.

The idealized protocol is as follows:

$$\begin{array}{lll}
\text{Message 1} & A \to B: & \{T_a,\, N_a,\, X_a,\, \{Y_a\}_{K_b}\}_{K_a^{-1}} \\
\text{Message 2} & B \to A: & \{T_b,\, N_b,\, N_a,\, X_b,\, \{Y_b\}_{K_a}\}_{K_b^{-1}} \\
\text{Message 3} & A \to B: & \{N_b\}_{K_a^{-1}}
\end{array}$$

As usual, the timestamps T_a and T_b are viewed simply as nonces in the idealized protocol.

The protocol analyzed

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

$$\begin{array}{ll}
A \models \overset{K_a}{\mapsto} A & B \models \overset{K_b}{\mapsto} B \\
A \models \overset{K_b}{\mapsto} B & B \models \overset{K_a}{\mapsto} A \\
A \models \sharp(N_a) & B \models \sharp(N_b) \\
A \models \sharp(T_b) & B \models \sharp(T_a)
\end{array}$$

From message 1, via the message-meaning rules, we obtain:

$$\begin{array}{l}
B \models A \mathrel{|\!\sim} X_a \\
B \triangleleft Y_a
\end{array}$$

Using T_a for nonce verification, we also derive:

$$B \models A \models X_a$$

Note that $B \models A \models Y_a$ is not derivable. Message 2 produces:

$$\begin{array}{l}
A \models B \mathrel{|\!\sim} X_b \\
A \triangleleft Y_b
\end{array}$$

Using T_b or N_a for nonce verification, we obtain:

$$A \models B \models X_b$$

Once again, we cannot obtain $A \models B \models Y_b$. Message 3 gives:

$$B \models A \models N_b$$

That is, it assures B that A has recently sent N_b.

This represents an outcome weaker than the authors desired. Surprisingly, the protocol does not lead to $B \models A \models Y_a$ or $A \models B \models Y_b$. 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. This problem can be fixed by several means, the simplest of which 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 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 $B \models A \mid\sim X_a$ instead of $B \models A \models X_a$.

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 be sufficient 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 allows an intruder C to replay one of A's old messages.

The following concrete exchange illustrates the flaw. The nonce N_b is not secret, and there is nothing to prevent C from using the same value in an instance of the protocol between A and C. If C is able to cause A to attempt authentication with C at the required time, the following sequence of messages can result. The intruder first contacts B:

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

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.

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

B replies as though the message came from A, and provides a new nonce, N_b. At this point C causes A to initiate authentication with C, by whatever means.

$$A \rightarrow C: \quad A,\ \{T'_a,\ N'_a,\ C,\ X'_a,\ \{Y'_a\}_{K_c}\}_{K_a^{-1}}$$

A has now initiated authentication with C. The exact content of the message is immaterial. C replies to A, providing the nonce N_b, which was originally provided by B.

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

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.

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

One possible 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 uniquely to this instance of the protocol in a timely fashion. 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, though this has not been shown in the description above. The analysis of the protocol is changed only slightly by the introduction of hashing. The next section deals with hash functions and how their use can be modelled in the logic.

12. On Hashing

In practice, signatures are costly. Often, principals do not sign whole messages, mainly for reasons of efficiency. Rather, in order to sign a (long, mostly cleartext) message m, a hash $H(m)$ of m is computed and signed. Thus, a principal A may send $m,\ \{H(m)\}_{K_a^{-1}}$ instead of $\{m\}_{K_a^{-1}}$. It remains to understand how this scheme compares with straightforward signature methods qualitatively: what guarantees do hash functions offer?

Obviously, trust in protocols that use hash functions is not always warranted. If H is an arbitrary function, nothing convinces one that when A has uttered $H(m)$ he must have also uttered m. In fact, A may never have seen m. This may happen, for instance, if the author of m gave $H(m)$ to A, who signed it and sent it. This is similar to the way in which a manager signs a document presented by a subordinate without reading the

details of the document. However, the manager expects anyone receiving this signed document to behave as though the manager had full knowledge of the contents. Thus, provided the manager is not careless and the hash function is suitable, signing a hash value should be considered the same as signing the entire message.

Some assumptions underlie the use of hash functions in signatures. As a basic assumption, it is intended that, given a hash value, it should be computationally infeasible to put together a message with the same hash value. Moreover, it should be computationally infeasible to generate two messages giving the same hash value, to avoid uncertainty in the identity of the data being signed.

We wish to capture the reasoning implicit in the use of hash functions. We introduce a symbol H to represent each function of interest. A postulate to attribute the message m to the author of the message $H(m)$ is wanted.

For a hash function H, we postulate:

$$\frac{P \models Q \mid\sim H(X), \quad P \triangleleft X}{P \models Q \mid\sim X}$$

In fact a stronger postulate appears reasonable, in the case where a message has been broken up into lots of pieces. This case in fact arises in such proposed standards as CCITT's X.509 (1987):

$$\frac{P \models Q \mid\sim H(X_1, \ldots, X_k), \quad P \triangleleft X_1, \ldots, P \triangleleft X_k}{P \models Q \mid\sim (X_1, \ldots, X_k)}$$

The significance of this postulate is that a principal may see pieces of a composite message by different means; in particular, some pieces may be the results of decryption. The principal may indeed come by the various pieces at different times—it makes no difference. When a signed hash of the composite message is received, it is still possible to deduce the origin of the composite message.

13. Semantics

Hopefully, the formulas liberally sprinkled throughout this paper are intuitively clear. In this section we take a more formal approach than previously and discuss a formal semantics of these formulas.

Beliefs

We describe an "operational" semantics, according to which principals develop beliefs by computation. In order to obtain new beliefs, principals are supposed to examine their current beliefs and apply a few computationally tractable inference rules. These rules represent the idealized workings of principals in actual authentication protocols. Thus, the statement that an authentication protocol gives rise to certain beliefs is a strong one: it means that the principals develop these beliefs even with realistic computational resources. However, the restrictive, operational notion of belief that we have adopted would certainly be harmful in the study of security properties, where we would want to guarantee that intruders learn no secrets even with powerful methods or algorithms.

The *local state* of a principal P is two sets of formulas $\mathcal{M}_P$ and $\mathcal{B}_P$. Intuitively, $\mathcal{M}_P$ is the set of messages that the principal sees and $\mathcal{B}_P$ is the set of beliefs of the principal. The sets $\mathcal{M}_P$ and $\mathcal{B}_P$ enjoy some closure properties; for simplicity, we take closure properties that correspond directly to the inference rules of the logic. For instance,

$$\text{if } (P \overset{K}{\leftrightarrow} Q) \in \mathcal{B}_P \text{ and } \{X\}_K \in \mathcal{M}_P \text{ then } X \in \mathcal{M}_P.$$

We imagine the closure properties are enforced by algorithms to derive and add new messages and beliefs.

A *global state* is a tuple containing the local states of all principals; in all the cases we consider, it is a triple with the local states of A, B, and S. If s is a global state then s_P is the local state of P in s and $\mathcal{B}_P(s)$ and $\mathcal{M}_P(s)$ are the corresponding sets of beliefs and messages. The *satisfaction* relation between global states and formulas has a trivial definition: $P \models X$ holds in state s if $X \in \mathcal{B}_P(s)$, and $P \triangleleft X$ holds if $X \in \mathcal{M}_P(s)$. A set (or conjunction) of formulas holds in a given state if each of its members holds.

A *run* is a finite sequence of states $s_0, \ldots, s_n$ where $\mathcal{B}_P(s_i) \subseteq \mathcal{B}_P(s_{i+1})$ and $\mathcal{M}_P(s_i) \subseteq \mathcal{M}_P(s_{i+1})$ for all $i \leq (n-1)$ and for each principal P. In other words, the sets of messages seen and the sets of beliefs can only increase. A run is a run of a particular protocol if all of the messages the protocol prescribes are communicated—other messages may be initially present or may come from the environment. More precisely, a protocol is a finite sequence of n "send" statements of the form

$$(P_1 \rightarrow Q_1 : X_1), \ldots, (P_n \rightarrow Q_n : X_n)$$

A run of the protocol is a run of length $n+1$ where $X_i \in \mathcal{M}_{Q_i}(s_i)$ for all $i \leq n$.

An annotation for the protocol *holds* in a run of the protocol if all of the formulas in the annotation hold in the corresponding states. More precisely, an annotation consists of $n+1$ sets of formulas of the forms $P \models X$ and $P \triangleleft X$ inserted before and after statements; it holds in a run of the protocol if the i-th set holds in the i-th state of the run for all $i \leq n$. An annotation is *valid* if it holds in all runs of the protocol where the first set of the annotation—the assumptions—holds.

Immediately, the annotation rules described earlier are sound: all legal annotations are valid. Just as trivially, the rules are also complete: all valid annotations are legal. To see this, given a protocol $(P_1 \rightarrow Q_1 : X_1), \ldots, (P_n \rightarrow Q_n : X_n)$ consider a run $s_0, \ldots, s_n$ where only the messages $X_1, \ldots, X_n$ are communicated. All valid annotations must hold in this run. Furthermore, we can show that any annotation that holds in this run can be derived.

True beliefs

While the semantics gives a meaning to the operators $\models$ and $\triangleleft$, the remaining operators are still largely a mystery. For instance, the semantics does not determine whether $A \mid\sim N_a$ is true or false in a given state. This is a deficiency if we are interested in judging the truth of beliefs.

Conjunction and quantification receive their usual interpretation (for quantification, we assume that the variables range over given domains of principals, keys, and formulas). In order to give a meaning to the remaining operators, however, the notion of state needs to be richer than the one we have used so far, as follows:

- Each state associates with each principal P a set $\mathcal{O}_P$ of formulas that he once said. This set has three closure properties: if $(\{X\}_K \textit{ from } P) \in \mathcal{O}_P$ then $X \in \mathcal{O}_P$; if $\langle X \rangle_Y \in \mathcal{O}_P$ then $X \in \mathcal{O}_P$; if $(X, Y) \in \mathcal{O}_P$ then $X \in \mathcal{O}_P$. We require that if $P_i \rightarrow Q_i : X_i$ is the i-th action of a protocol then the set of formulas once said increases only for P_i in the i-th state of all runs of this protocol. More precisely, if $R \neq P_i$ then $\mathcal{O}_R(s_i) = \mathcal{O}_R(s_{i-1})$ and $\mathcal{O}_{P_i}(s_i)$ is the closure (by the rules above) of $\mathcal{O}_{P_i}(s_{i-1}) \cup \{X_i\}$. In addition, each principal must believe all the formulas he has said recently, in the sense that if $X \in \mathcal{O}_P(s)$ because of a message in the protocol then $X \in \mathcal{B}_P(s)$.

  The formula $P \mid\sim X$ holds in state s if $X \in \mathcal{O}_P(s)$.

- In each run each principal P has jurisdiction over a set of formulas $\mathcal{J}_P$. We require that if $X \in \mathcal{J}_P$ and $P \models X$ holds then X holds as well.

The states in the run satisfy $P \Rightarrow X$ if $X \in \mathcal{J}_P$.

- Each run assigns a set of good shared keys $\mathcal{K}_{\{P,Q\}}$ to each pair of principals P and Q. We require that these keys are used only by the appropriate principals, that is, if $R \triangleleft (\{X\}_K \textit{ from } R')$ holds and $K \in \mathcal{K}_{\{P,Q\}}$ then either $R' = P$ and $(\{X\}_K \textit{ from } P) \in \mathcal{O}_P$ or $R' = Q$ and $(\{X\}_K \textit{ from } Q) \in \mathcal{O}_Q$.

  The states in the run satisfy $P \overset{K}{\leftrightarrow} Q$ if $K \in \mathcal{K}_{\{P,Q\}}$.

- Each run assigns a set of good public keys $\mathcal{K}_P$ to each principal P. We require that only the appropriate principals use the matching secret keys, that is, if $R \triangleleft (\{X\}_{K^{-1}} \textit{from } R')$ holds and $K \in \mathcal{K}_P$ then $R' = P$ and $(\{X\}_{K^{-1}} \textit{from } P) \in \mathcal{O}_P$.

  The states in the run satisfy $\overset{K}{\mapsto} P$ if $K \in \mathcal{K}_P$.

- Each run assigns a set of shared secrets $\mathcal{S}_{\{P,Q\}}$ to each pair of principals P and Q. We require that shared secrets are used only by the appropriate principals, that is, if $R \triangleleft \langle X \rangle_Y$ holds and $Y \in \mathcal{S}_{\{P,Q\}}$ then either $\langle X \rangle_Y \in \mathcal{O}_P$ or $\langle X \rangle_Y \in \mathcal{O}_Q$.

  The states in the run satisfy $P \overset{X}{\rightleftharpoons} Q$ if $X \in \mathcal{S}_{\{P,Q\}}$.

- Since we do not concern ourselves with expressions of the forms $P \models \{X\}_K$ and $P \models \langle X \rangle_Y$, we do not even attempt to assign a truth value to expressions of the forms $\{X\}_K$ or $\langle X \rangle_Y$.

- Each run determines a set of fresh formulas $\mathcal{F}$. This set has a closure property: if $X \in \mathcal{F}$ and X is a subformula of Y then $Y \in \mathcal{F}$. If $X \in \mathcal{F}$ and X was once said (that is, $X \in \mathcal{O}_P(s_i)$ for some P and i) then X should have been said recently (that is, $X \notin \mathcal{O}_P(s_0)$ for all P).

  The states in the run satisfy $\sharp(X)$ if $X \in \mathcal{F}$.

Clearly, some beliefs are false. This seems essential to a satisfactory semantics. Questions of trust and delegation, central to our study, would become meaningless if all beliefs had to be true. Moreover, we can consider many interesting runs—for instance, those where an intruder has broken the cryptosystem—because we leave open the possibility of incorrect beliefs.

Let us define *knowledge* as "truth in all possible worlds" (see, for example, Halpern & Moses 1984). More precisely, P knows X in state s if and only if X holds in all states s' where the local state of P is the same as in s, that is, $s'_P = s_P$. In general, the notions of knowledge and belief are incomparable. For instance, some erroneous initial beliefs are certainly not knowledge, while each principal knows all tautologies, but does not necessarily believe them.

Most beliefs happen to be true in practice, but the semantics does not account for this coincidence. To guarantee that all beliefs are true we would need to guarantee that all initial beliefs are true. In this case, belief is a rudimentary resource-bounded approximation to knowledge.

14. Conclusions

Recent literature has emphasized the importance of reasoning about knowledge for understanding distributed computation (see, for example, Halpern & Moses 1984). Furthermore, there have been some formal descriptions of cryptographic protocols (DeMillo *et al.* 1982; Merritt & Wolper; Halpern *et al.* 1988). Although these works 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 we have treated several examples. The following table lists protocols studied with the logic and summarizes their attributes.

	Needham-Schroeder conv. 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*	distribute key	distribute extra key	establish secrets	transfer data
Cryptosystem	conv.	conv.	conv.	conv.	conv.	conv.	public key	public key
Uses secrets					×		×	
Nonces/clocks	nonces	nonces	clocks	clocks	nonces	nonces	nonces	both
Proves presence of	$A\&B$	B	$A\&B$†	A	$A\&B$	$A\&B$	$A\&B$	$A\&B$†
Redundancy	×	×	×		×	×		×
Bugs	×					×	×	×‡

* In this case, A, rather than a trusted server, generates the key.

† B's presence is guaranteed to A only if optional protocol steps are used.

‡ Security breaches do not even require key compromise.

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.

The principals involved in the protocols are A and B; the initiator is A.

The examples in this study show how an extremely simple logic can capture subtle differences between protocols. The logic lacks all features that would make it difficult to use, yet it does what is needed. 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 suggesting corrections.

Acknowledgements

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 anonymous 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

Burrows, M., Abadi, M. & Needham, R.M. 1988 Authentication: A Practical Study in Belief and Action. *Proceedings of the Second Conference on Theoretical Aspects of Reasoning about Knowledge*, M. Vardi, ed., pp. 325–342.

Bauer, R.K., Berson, T.A. & Feiertag, R.J. 1983 A Key Distribution Protocol using Event Markers. *ACM Transactions on Computer Systems* Vol. 1, No. 3, pp. 249–255.

CCITT 1987 Draft Recommendation X.509. The Directory-Authentication Framework, Version 7. Gloucester.

DeMillo, R.A., Lynch, N.A. & Merritt, M.J. 1982 Cryptographic Protocols. *Proceedings of the Fourteenth ACM Symposium on the Theory of Computing*, pp. 383–400.

Denning, D.E. & Sacco, G.M. 1981 Timestamps in Key Distribution Protocols. *CACM* Vol. 24, No. 8, pp. 533–536.

Dolev, D. & Yao, A.C. 1983 On the Security of Public Key Protocols. *IEEE Transactions on Information Theory* Vol. IT-29, No. 2, pp. 198–208.

Hoare, C.A.R. 1969 An Axiomatic Basis for Computer Programming. *CACM* Vol. 12, No. 10, pp. 576–580.

Halpern, J.Y. & Moses, Y.O. 1984 Knowledge and Common Knowledge in a Distributed Environment. *Proceedings of the Third ACM Conference on the Principles of Distributed Computing*, pp. 480–490.

Halpern, J.Y., Moses, Y.O. & Tuttle, M.R. 1988 A Knowledge-Based Analysis of Zero Knowledge (Preliminary Report). *Proceedings of the Twentieth ACM Symposium on Theory of Computing*, pp. 132–147.

Halpern, J.Y. & Vardi, M.Y. 1986 The Complexity of Reasoning about Knowledge and Time. *Proceedings of the Eighteenth ACM Symposium on the Theory of Computing*, pp. 304–415.

Millen, J.K., Clark, S.C. & Freedman, S.B. 1987 The Interrogator: Protocol Security Analysis. *IEEE Transactions on Software Engineering* Vol. SE-13, No. 2, pp. 274–288.

Miller, S.P., Neuman, C., Schiller, J.I. & Saltzer, J.H. 1987 Kerberos Authentication and Authorization System. *Project Athena Technical Plan* Section E.2.1, MIT.

Merritt, M.J. & Wolper, P.L. States of Knowledge in Cryptographic Protocols. Draft.

Nguyen, V. & Perry, K.J. Do We Really Know What Knowledge Is? Draft.

Needham, R.M. & Schroeder, M.D. 1978 Using Encryption for Authentication in Large Networks of Computers. *CACM* Vol. 21, No. 12, pp. 993–999.

Needham, R.M. & Schroeder, M.D. 1987 Authentication Revisited. *Operating Systems Review* Vol. 21, No. 1, p. 7.

Otway, D. & Rees, O. 1987 Efficient and Timely Mutual Authentication. *Operating Systems Review* Vol. 21, No. 1, pp. 8–10.

Rivest, R.L., Shamir, A. & Adleman, L. 1978 A Method for Obtaining Digital Signatures and Public-key Cryptosystems. *Communications of the ACM* Vol. 21, No. 2, pp. 120-126.

Satyanarayanan, M. 1987 Integrating Security in a Large Distributed System. CMU technical report CMU-CS-87-179.

Voydock, V.L. & Kent, S.T. 1983 Security Mechanisms in High-Level Network Protocols. *Computing Surveys* Vol. 15, No. 2, pp. 135–171.