Lower Bounds on Messages and Rounds for Network Authentication Protocols

Li Gong

SRI International
Computer Science Laboratory
333 Ravenswood Avenue
Menlo Park, California 94025 U.S.A.

Abstract

Research in authentication protocols has largely focused on developing and analyzing protocols that are secure against certain types of attacks. There is little and only scattered discussion on protocol efficiency. This paper presents results on the lower bounds on the numbers of messages and rounds required for network authentication. For each proven lower bound, an authentication protocol achieving the bound is also given, thus proving that the bound is a tight bound if the given optimal protocol is secure.

1 Introduction

Authentication is by definition a process to verify one's claim of identity. Since authentication is usually a prelude to further communication and computation, an authentication protocol often arranges that the protocol participants, once their identities are verified, agree upon an encryption key—a temporary key—for later use (e.g., within a user session). Thus an authentication protocol is sometimes also called a key distribution protocol.

Current research in authentication protocols has largely focused on the *security* of protocols, and there is only scattered published discussion on the issue of protocol *efficiency* (e.g., [Bird 93, Birrell 85, Gong 89, Gong 93, Neuman 93, Yahalom 93]). The treatment of efficiency or performance is generally given a low priority and is often rather ad hoc. One possible explanation is that since such protocols tend to involve only a few messages, optimization is not seen as a very urgent requirement.

However, as in the field of algorithm complexity, it is natural and beneficial to inquire whether a protocol that achieves authentication in a particular environment is also in some sense minimal or optimal. For example, eliminating one message from a five-message protocol represents a 20% reduction in the number of messages and possibly a similar amount of reduction in the overall running time of the protocol. Even for those who do not care for reduction of one or two messages, it is useful to know that the protocols they use are not too far from being optimal. Moreover, instead of dealing with each individual protocol, it is desirable that some general results on lower bounds are proven, which can serve as a reference for system and protocol designers.

To give just one example of how this type of result can be useful, assume (fictitiously) that, for a particular environment, no protocols can be both message-optimal and round-optimal. Faced with this impossibility result, a system can provide a flexible authentication service that can give clients the option to make dynamic trade-offs according to individual need. For example, a client who needs immediate authentication may elect to run the version of the authentication protocol that uses more messages but has fewer rounds and is thus faster to complete. On the other hand, when the network is unreliable and retransmission is frequent or when there is a very slow network link (such as a 2400 baud modem connection) on the critical path, a client who is experiencing communication problems may wish to run the version that uses fewer messages. No known authentication system has such flexibility.

In this paper, we attempt to investigate the issue of efficiency in a more systematic manner. In section 2, we first describe the system model and specify some important assumptions we make about the environment. We then select a few settings in which authentication normally takes place, and describe the most common objectives or goals of authentication. After that, we define two metrics of efficiency, the number of messages and the number of rounds. In section 3, we prove lower bounds of the metrics for each setting and each set of goals. For each proven lower bound, we also give an optimal protocol achieving the bound, thus showing that the bound is a tight bound if the protocol given is secure. We then discuss the use of partial or independent handshakes, of uncertified keys, and of public-key or mixed-key systems. In section 7, we generalize our results to the case with multiple clients setting up group keys. Finally, we conclude with a discussion of future work.

Whenever we cannot find a published protocol that is optimal for the particular setting, we provide an example protocol, to demonstrate the achievability of the lower bound rather than to suggest a practical protocol for use in a real distributed system.

2 Preliminaries

2.1 System Model and Assumptions

The environment we assume is a distributed system where parties (e.g., processes, users, machines) communicate with each other only by sending and receiving messages via com-

1st Conf.- Computer & Comm. Security '93-11/93 -VA,USA
© 1993 ACM 0-89791-629-8/93/0011...$1.50

munication links between them. We assume that the underlying cryptographic mechanisms are not vulnerable with regard to message secrecy and integrity so that we do not consider attacks such as cryptanalysis and message slicing. Any principal can place or inject a message on any link at any time; can see all exchanged messages; can delete, alter, or redirect any message being passed along any link; can initiate communications with another party; and can replay messages recorded from past communications.

The model of authentication taken here is the common one often found in the literature (e.g., [Needham 78]). There are three participants: two clients, denoted by A and B, and an authentication server S via which A and B agree upon a temporary key. We assume that client A is always the originator of the protocol execution, and we call B the responder.

Both conventional cryptosystems (e.g., DES [NBS 77]) and public-key systems (e.g., RSA [Rivest 78]) are useful in authentication protocols [Needham 78]. We concentrate first on the use of conventional cryptosystems and will analyze the use of public key or mixed-key systems later. Similarly, we start with mutual authentication and will discuss other variations later.

When participants use only conventional encryption systems, we assume that before an execution of the authentication protocol each client shares a secret key with the server, but the two clients do not share any secret. After the successful completion of the protocol, the clients share a secret, often known as a session or temporary key, to be used for future communication. Thus authentication here means identification plus key or certificate distribution via the trusted authentication server.

We assume that the temporary key is for conventional cryptosystems only. We further assume that a client will not accept or act upon a temporary key (or a candidate temporary key) unless the client can determine that the message in which the key is distributed is fresh. Such a key whose "goodness" remains unconfirmed until it is further used is called an "uncertified key" [Burrows 89, p.32]. We discuss its usage in section 5.

2.2 Settings

There are two important setting parameters to consider. The first parameter is the mechanism each participant uses to establish the freshness of messages. Broadly speaking, there are two well-known mechanisms. One is based on synchronized clocks [Denning 81][1], the other uses nonces [Needham 78]. We discuss the two cases separately.

Except for a brief discussion toward the end of the paper, we will not consider settings when one participant relies on clocks and another relies on nonces, because the results of this setting can be derived from the two simpler settings. Other hybrid schemes not discussed include use of a timestamp as a nonce or for dual purposes, or the setting in which one party has the option to use a timestamp or a nonce, depending on environmental restrictions, e.g., whether it is possible to piggyback a "challenge" in a previous message, as in Kerberos [Neuman 93].

The second parameter is concerned with the question of who chooses the temporary key. We consider three possibilities. One is that the server chooses the key. The second is that any one client can choose the key. The third is that both clients participate in choosing the key.

If the client is competent, letting a client choose a key can shorten protocol. However, if one client does not necessarily trust the other, then both must participate in choosing a key.

Letting the server S and one or both clients be involved in choosing the key is thought to be unnecessary in using conventional cryptosystems, because S is in a position to know all clients' secrets, including both long-term and temporary keys, and thus must be trusted not to divulge such secrets. Also, the server is generally assumed to be better at generating quality keys. However, in the case of using public-key systems, letting clients be involved in choosing the keys can improve security, because S no longer has to know all the secrets.

2.3 Goals

The goals of authentication have been carefully studied, and a protocol usually falls in one of two levels [Burrows 89]. We will briefly discuss other variations in section 4.

A protocol at the first level can be viewed as authentication only. That is, after completing the protocol, each client will have received a key in a timely or fresh message, and the client believes that the key is suitable to be shared with the other client.

A protocol at the second level can be called authentication with handshake. That is, after completing the protocol, each client will also believe that the other client has received the key properly and believes in the suitability of the key. This extra requirement is generally met by a handshake exchange at the end of the protocol.

2.4 Metrics

In this paper we focus on two important efficiency metrics of a protocol: the total number of messages and the number of rounds. We now define these metrics more precisely.

A message is a data item sent by one client to a single destination at one time. A message may contain headers to indicate its source and destination. In some implementations, if the size of the data item is too large, the message is actually fragmented at a lower level into many packets. We will still count this as one message. A message being forwarded will count as a new message sent by the intermediate party. A broadcast to many destinations will be viewed as the same message being sent to those destinations separately and thus will count as many messages.

We count a broadcast as many messages because in a typical implementation, a broadcast will result in multiple messages that add a system and network load many times greater than load added by a simple message. Also, the multiple recipients of a broadcast all must respond, e.g., by examining the incoming message, changing state if necessary, and possibly replying to the sender or sending another message according to the protocol specification. We aim to include all messages that consume non-negligible system resources so that minimizing the number of messages makes more efficient uses of these resources.

Let us assume that the network is uniformly connected so a message will always travel to its destination in one unit of time, no matter where the source and the destination are. We neglect the computation time at each node. A round consists of all messages that can be sent and received in parallel within one time unit. Thus a participant can simultaneously send different messages to different destinations in

[1] Clock synchronization mechanisms often do not address their own security problems. For a more detailed discussion and more references on the risks of using clocks, see [Gong 92].

one round, and so can multiple participants send messages in one round. The number of rounds in a protocol is the total number of time units from the instant that the originator sends the first message till the instant that the last message is received, under the best execution scenario.

This metric is important because the number of rounds in a protocol is an indirect measure of the worst-case protocol execution time. In some applications, a client may wish to trade other metrics, such as the number of messages, for a faster protocol completion time.

2.5 Proof Methods

Based on the assumptions (especially the one excluding uncertified keys), we make the following observations which are vital to our proof methods:

1. A client cannot send out a handshake message before it has received the temporary key. Thus, the last handshake message cannot be sent before all clients have received the temporary key.

2. A client without a synchronized clock cannot accept a temporary key before it sends out a nonce.

3. The protocol responder (client) or the server cannot send out any message (e.g., a nonce) before the protocol originator sends out a notification message.

Based on the above observations, our main proof technique for the lower bound on the number of messages is to identify crucial messages that are necessary for completion of a protocol execution but which could not be further combined. For example, for one client, the following three messages cannot be combined: the message to send out a nonce, the one to receive a temporary key, and the one for handshake.

The main proof technique for the lower bound on the number of rounds is to identify a critical path – a causal chain of messages – that cannot be further shortened. For example, the sequence of sending out a nonce, later receiving a temporary key, and finally sending a handshake message cannot be shorten to less than three rounds.

3 Lower Bounds on Messages and Rounds

Since there are two setting parameters with a total of six settings, and two levels of authentication goals, there are 12 cases to consider. [Readers uninterested in detailed proofs may wish to skip to section 3.13 for a summary.] For ease in enumerating all 12 cases, we use some shorthand notation (see Table 1 below) to denote these cases.

In our proofs, we give only brief and informal arguments.[2] For each proven lower bound, we give an optimal protocol to show that the bound is actually achievable. When we cannot find a published protocol that is optimal for the particular setting, we will provide an example protocol solely to demonstrate the achievability of the lower bound rather than to suggest a practical protocol for use in a real distributed system.

Besides pointing out that all these protocols are carefully designed to resist known types of attacks, we choose not to devote much space to discuss the security of these protocols, since efficiency rather than security is the central theme of this paper. Because the lower bounds are proven independently of the security of the example protocols, these bounds are in fact tight bounds if the security of the given optimal protocols can be established. If an example protocol is later shown to be insecure, the lower bound still stands, but a new protocol would be needed to demonstrate the achievability of the lower bound.

[2] It is always debatable as to what constitutes a proof. The essays by Lakatos [Lakatos 76] provide excellent philosophical, logical, and historical perspectives of theorem proving.

AO	authentication only
AH	authentication with handshake
SO	server choosing the temporary key
CO	one client choosing the temporary key
CC	both clients choosing the temporary key
TB	clock-based
NB	nonce-based
Kas	encryption key shared between A and S
Na	nonce generated by A
Ta	timestamp taken from A's clock
K	temporary key
$K1$, $K2$	candidate temporary keys chosen by A, B
$\{x\}_k$	x encrypted with key k
x, y	x concatenated with y
$A \rightarrow B : x$	A sending message x to B

Table 1: Notation

For adequate protection, all messages (except those cited from previously published protocols) have the following standard format:

$$\{Sender, Recipient, C1, K, C2, Freshness - id\}_{Ksr}$$

Each separately encrypted portion of a message contains the identity of its sender, followed by that of the intended recipient. This ensures that an attacker cannot redirect a message without being detected. A message distributing a temporary key contains a 3-tuple $\{client1, key, client2\}$ which indicates that the key being distributed is for the two clients herein named. This ensures that an attacker cannot cause misunderstandings of the meaning of the message by replaying and redirecting messages. The last part of the message is a freshness identifier, which can be either a timestamp taken from the sender's clock or a nonce previously generated by the intended recipient. This defeats replay of past messages. The whole message is encrypted with Ksr, a key shared between the sender and the recipient. Each message should also carry a unique identifier (not shown in the above format) specifying the protocol name and version number, and the sequence or position number of the message within the protocol. This identifier, together with the freshness identifier, prevents an attacker from mixing messages from different protocols or different protocol runs. Obviously, this overly conservative arrangement is unlikely to result in the shortest messages.

3.1 Case 1: TB+AO+SO

Messages. The originator has to notify S of starting the protocol, who then needs to send at least two more messages to the two clients to distribute the temporary key. Thus a lower bound is three messages. The Denning-Sacco protocol [Denning 81] achieves this lower bound.

Rounds. Key distribution cannot happen before S is notified by the originator, thus two rounds is a lower bound.

We can rearrange the messages in the Denning-Sacco protocol so that S sends the key directly to B instead of sending it via A, resulting in the following protocol which achieves the lower bound because messages 2 and 3 can be sent concurrently.

1. A → S: A, B

2. S → A: $\{B, K, Ts\}_{Kas}$
3. S → B: $\{A, K, Ts\}_{Kbs}$

The above protocol is both message and round optimal. For illustration purposes, we group together the messages belonging to the same round and separate the different rounds with blank lines.

3.2 Case 2: TB+AH+SO

Case 2 differs from Case 1 only in that a handshake is now needed.

Messages. The orignator has to notify the server, who must distribute a key to both clients in at least two more messages. The last handshake message cannot be sent out before both clients have received the key. Thus a lower bound is four messages. The following protocol achieves this lower bound.

1. A → S: A, B
2. S → A: $\{S, A, A, K, B, Ts\}_{Kas}$, $\{S, B, A, K, B, Ts\}_{Kbs}$
3. A → B: $\{S, B, A, K, B, Ts\}_{Kbs}$, $\{A, B, Ta\}_K$
4. B → A: $\{B, A, Tb\}_K$

Rounds. The three stages of protocol initiation, key distribution, and handshake cannot happen concurrently, thus three rounds is a lower bound. The following protocol, which is derived from previous protocol by rearranging the messages, achieves this lower bound because messages 2 and 3, and messages 4 and 5, can be sent concurrently.

1. A → S: A, B

2. S → A: $\{S, A, A, K, B, Ts\}_{Kas}$
3. S → B: $\{S, B, A, K, B, Ts\}_{Kbs}$

4. A → B: $\{A, B, Ta\}_K$
5. B → A: $\{B, A, Tb\}_K$

3.3 Case 3: TB+AO+CO

This case differs from Case 1 only in that now any client can choose the temporary key.

Messages. One client needs to choose the key and send it to the other client via the server. Thus two messages is a lower bound. The wide-mouthed-frog protocol [Burrows 89] achieves this lower bound.

Rounds. The two messages for key distribution can only be sent sequentially with the server as the intermediate party; thus two rounds is a lower bound, which is also achieved in the wide-mouthed-frog protocol.

3.4 Case 4: TB+AH+CO

This case differs from Case 3 only in that now a handshake is needed.

Messages. Distributing the key needs at least two messages, and the final handshake message requires an extra one; thus three messages is a lower bound. Adding one more message (message 3) to the wide-mouthed-frog protocol achieves this bound.

1. A → S: $A, B, \{B, K, Ta\}_{Kas}, \{A, Ta\}_K$
2. S → B: $A, B, \{A, K, Ts\}_{Kbs}, \{A, Ta\}_K$
3. B → A: $\{B, Tb\}_K$

Rounds. The two messages for key distribution must be sent sequentially because the server acts as a broker, and the final handshake message cannot be sent before the key is distributed; thus three rounds is a lower bound, which is also achieved in the above protocol.

3.5 Case 5: TB+AO+CC

When both clients participate in choosing the temporary key, client A chooses the candidate temporary key $K1$, and B chooses $K2$. Later, they can derive a temporary key K from the two candidate keys, possibly using a one-way hash function such as $K = h(K1, K2)$, for use in future communications.

Messages. Each client has to send its contribution in selecting the temporary key to the server who then forwards it onto the other client. Thus four messages is a lower bound. The protocol below achieves this bound.

1. A → S: $A, B, \{A, S, A, K1, B, Ta\}_{Kas}$
2. S → B: $\{S, B, A, K1, B, Ts\}_{Kbs}$
3. B → S: $\{B, S, B, K2, A, Tb\}_{Kbs}$
4. S → A: $\{S, A, B, K2, A, Ts\}_{Kas}$

Rounds. The responder has to be notified before it can proceed to key selection. Then its contribution must be sent to the server first before being forwarded to the originator. Thus three rounds is a lower bound. Messages in the above protocol can be rearranged to achieve this bound. Here messages 1 and 2, and 3 and 4 can be sent concurrently.

1. A → S: $A, B, \{A, S, A, K1, B, Ta\}_{Kas}$
2. A → B: A, B

3. S → B: $\{S, B, A, K1, B, Ts\}_{Kbs}$
4. B → S: $\{B, S, B, K2, A, Tb\}_{Kbs}$

5. S → A: $\{S, A, B, K2, A, Ts\}_{Kas}$

3.6 Case 6: TB+AH+CC

This case differs from Case 5 only in that now a handshake is needed.

Messages. As in Case 5, four messages are needed to exchange key selections of A and B. After the last of these messages has been received, at least one more message is necessary to complete a two-way handshake. Thus five messages is a lower bound, which is achieved in the following protocol.

1. A → S: $A, B, \{A, S, A, K1, B, Ta\}_{Kas}$
2. S → B: $\{S, B, A, K1, B, Ts\}_{Kbs}$
3. B → S: $\{B, S, B, K2, A, Tb\}_{Kbs}, \{B, A, Tb\}_K$
4. S → A: $\{S, A, B, K2, A, Ts\}_{Kas}, \{B, A, Tb\}_K$
5. A → B: $\{A, B, Ta\}_K$

Again, K is the temporary key computed from $K1$ and $K2$. Note that Ta in messages 2 and 5 both represent the reading from A's clock, but the actual values in the two messages may differ.

Rounds. The responder has to be notified before it can proceed to key selection. Then its contribution must be sent to the server first before being forwarded on to the originator. Only after that can the originator send a handshake message. Thus four rounds is a lower bound. Messages in the above protocol can be rearranged to achieve this bound. Messages 1 and 2, 3 and 4, and 5 and 6 can be sent concurrently.

1. A → S: $A, B, \{A, S, A, K1, B, Ta\}_{Kas}$
2. A → B: A, B

3. S → B: $\{S, B, A, K1, B, Ts\}_{Kbs}$
4. B → S: $\{B, S, B, K2, A, Tb\}_{Kbs}$

5. S → A: $\{S, A, B, K2, A, Ts\}_{Kas}$
6. B → A: $\{B, A, Tb\}_K$

7. A → B: $\{A, B, Ta\}_K$

3.7 Case 7: NB+AO+SO

All cases from this point on are nonce-based. We recall the principle that each party concerned with freshness needs to choose a nonce of its own [Needham 87].

Messages. Each client has to choose a nonce and send it out, and each expects to receive a message from the server containing its nonce as well as the temporary key; therefore four messages is a lower bound. A protocol in the style of the Otway-Rees protocol [Otway 87] (i.e., 2 nested RPCs) achieves this lower bound:

1. A → B: A, B, Na
2. B → S: A, B, Na, Nb
3. S → B: $\{S, B, A, K, B, Nb\}_{Kbs}$, $\{S, A, A, K, B, Na\}_{Kas}$
4. B → A: $\{S, A, A, K, B, Na\}_{Kas}$

Rounds. The responder has to be notified before it can send out its nonce and later receive a fresh message; thus three rounds is a lower bound. The messages in the above protocol can be rearranged to achieve this bound.

1. A → B: A, B, Na

2. B → S: A, B, Na, Nb

3. S → A: $\{S, A, A, K, B, Na\}_{Kas}$
4. S → B: $\{S, B, A, K, B, Nb\}_{Kbs}$

The above protocol is both message and round optimal.

3.8 Case 8: NB+AH+SO

Messages. Compared with Case 7, at least one more message is needed to complete the two-way handshake (after both clients have received the temporary key). Thus five messages is a lower bound, which is achieved in the following protocol:

1. A → B: A, B, Na
2. B → S: A, B, Na, Nb
3. S → B: $\{S, B, A, K, B, Nb\}_{Kbs}$, $\{S, A, A, K, B, Na\}_{Kas}$
4. B → A: $\{S, A, A, K, B, Na\}_{Kas}, \{B, A, Na\}_K, Nb$
5. A → B: $\{A, B, Nb\}_K$

Rounds. As we found for the number of messages, at least one more round is needed than in Case 7 to complete the handshake; thus four rounds is a lower bound, which is achieved by rearranging the messages in the above protocol. Note that messages 3 and 4, and 5 and 6, can be sent concurrently.

1. A → B: A, B, Na

2. B → S: A, B, Na, Nb

3. S → A: $\{S, A, A, K, B, Na\}_{Kas}, Nb$
4. S → B: $\{S, B, A, K, B, Nb\}_{Kbs}$

5. A → B: $\{A, B, Nb\}_K$
6. B → A: $\{B, A, Na\}_K$

3.9 Case 9: NB+AO+CO

Messages. Since one client chooses the temporary key, only the other client who later receives the key needs to choose and send a nonce. These exchanges account for two messages. But since the clients initially do not share any secret, the key has to be sent via the authentication server, which requires one more message and brings the lower bound to three messages. The following protocol achieves this bound:

1. A → B: A, B, Na
2. B → S: $A, B, \{B, S, A, K, B, Na\}_{Kbs}$
3. S → A: $\{S, A, A, K, B, Na\}_{Kas}$

Note that although S cannot tell if message 2 is a replay, A will detect any replay after receiving message 3. In this and some subsequent protocols (including all CC cases), A acts as a message gateway, decrypting and reencrypting messages as needed.

Rounds. If the originator chooses the key, then the responder must be notified before choosing a nonce and later receiving the key. If the responder chooses the key, then after being notified, it must send the key via the authentication server. Thus in either scenario, three rounds is a lower bound, which is achieved in the above protocol.

3.10 Case 10: NB+AH+CO

Messages. Compared with Case 9, an extra handshake stage costs at least one more message, and the lower bound of four messages is achieved in the following protocol:

1. A → B: A, B, Na
2. B → S: A, B, Na, Nb, $\{B, S, A, K, B, Na\}_{Kbs}, \{B, A, Na\}_K$
3. S → A: $\{S, A, A, K, B, Na\}_{Kas}, \{B, A, Na\}_K, Nb$
4. A → B: $\{A, B, Nb\}_K$

Rounds. As we found for the number of messages, at least one more round than in Case 9 is needed to complete the handshake. This lower bound of four rounds is achieved in the above protocol.

3.11 Case 11: NB+AO+CC

Messages. Each client sends out its nonce and later receives a candidate key (chosen by the other client) together with its nonce. This requires at least three messages. However, the authentication server must mediate key exchange; thus two more messages, or a total of five messages, are needed. The following protocol achieves this lower bound.

1. A → B: A, B, Na
2. B → S: $A, B, Nb, \{B, S, B, K2, A, Na\}_{Kbs}$
3. S → A: $\{S, A, B, K2, A, Na\}_{Kas}, Nb$
4. A → S: $\{A, S, A, K1, B, Nb\}_{Kas}$
5. S → B: $\{S, B, A, K1, B, Nb\}_{Kbs}$

Rounds. The responder must be notified before it can send its nonce to the originator and later receive the key chosen by the originator. This process requires three rounds. Since the key must be exchanged via the authentication server, one more round is needed. Thus four rounds is a lower bound, which can be achieved by rearranging the messages in the above protocol. Note that messages 2 and 3, and 4 and 5, can be sent concurrently.

1. A → B: A, B, Na

2. B → A: Nb
3. B → S: $A, B, \{B, S, B, K2, A, Na\}_{Kbs}$

4. S → A: $\{S, A, B, K2, A, Na\}_{Kas}$
5. A → S: $\{A, S, A, K1, B, Nb\}_{Kas}$

6. S → B: $\{S, B, A, K1, B, Nb\}_{Kbs}$

3.12 Case 12: NB+AH+CC

Messages. The handshake requires at least one more message than in Case 11; thus six messages is a lower bound, which can be achieved in the following protocol:

1. A → B: A, B, Na
2. B → S: $A, B, Nb, \{B, S, B, K2, A, Na\}_{Kbs}$
3. S → A: $\{S, A, B, K2, A, Na\}_{Kas}, Nb$
4. A → S: $\{A, S, A, K1, B, Nb\}_{Kas}, \{A, B, Nb\}_K$
5. S → B: $\{S, B, A, K1, B, Nb\}_{Kbs}, \{A, B, Nb\}_K$
6. B → A: $\{B, A, Na\}_K$

Rounds. Similarly, the handshake requires at least one more round than in Case 11; thus five rounds is a lower bound, which can be achieved by rearranging the messages in the above protocol. Note that messages 2 and 3, and 4 and 5, can be sent concurrently.

1. A → B: A, B, Na

2. B → A: Nb
3. B → S: $A, B, \{B, S, B, K2, A, Na\}_{Kbs}$

4. S → A: $\{S, A, B, K2, A, Na\}_{Kas}, Nb$
5. A → S: $\{A, S, A, K1, B, Nb\}_{Kas}, \{A, B, Nb\}_K$

6. S → B: $\{S, B, A, K1, B, Nb\}_{Kbs}, \{A, B, Nb\}_K$

7. B → A: $\{B, A, Na\}_K$

lower bound msg/round	authentication only (AO) key chooser: server (SO)	one client (CO)	both clients (CC)
clock-based (TB)	3/2	2/2	4/3
nonce-based (NB)	4/3	3/3	5/4
lower bound msg/round	**authentication + handshake (AH) key chooser: server (SO)**	**one client (CO)**	**both clients (CC)**
clock-based (TB)	4/3	3/3	5/4
nonce-based	5/4	4/4	6/5

Table 2: Lower Bounds on Numbers of Messages and Rounds

3.13 Summary and Observations

Table 2 summarizes the proven results about lower bounds on the numbers of messages and rounds.

It has long been suspected that nonce-based protocols require at least one more message than clock-based ones. The analysis in this paper confirms that clock-based protocols save exactly one message in all of the 12 scenarios.

Conducting a handshake consumes only one more message, since the first half of the handshake can always be piggy-backed on an earlier message.

Letting the authentication server choose the temporary key reduces one message from the case when both clients participate in choosing the temporary key. Letting any one client choose the temporary key further reduces the number of messages by one.

The actual time to complete a round of message exchanges depends on the network latency in message delivery; thus a lower bound on the rounds indirectly reflects the worst-case protocol execution time – the lapse between the time a protocol is initiated and the time the last message is received at its intended destination.

Nonce-based protocols requires exactly one round more than clock-based protocols. Also, in nonce-based protocols, the lower bounds are the same for letting the server or a client choose the temporary key, but one more round is needed if both clients participate in choosing the key.

We have obtained achievable lower bounds in the case of letting *either* client choose the temporary key. If, as required in a particular environment, one or the other client is predesignated to be the key chooser, these lower bounds are not necessarily achievable, and better lower bounds for such situations can be easily worked out with our proof methods.

For example, in clock-based cases (TB+AO+CO and TB+AH+CO), the proven lower bounds are achieved by letting the protocol originator choose the temporary key. It is easy to see that if the protocol responder must choose the key, then one more (notification) message from the originator to the responder, and thus one more round, is needed, since essentially the two clients now switch roles after the initial notification message. Interestingly, in nonce-based cases (NB+AO+CO and NB+AH+CO), the proven lower bounds are achieved by letting the protocol responder choose the temporary key. Similarly, one more message and one more round are needed if the originator must choose the key.

Finally, when there is a single trusted server and conventional cryptosystem is used, letting both clients participate in choosing the temporary key is generally not a good idea

because it costs more (than letting the server choose the key) but gains nothing in security – the server knows all the secrets anyway and is generally better at choosing good encryption keys.

4 Partial Handshakes and Independent Handshakes

The handshake we have discussed is mutual handshake, in that clients inform each other that they have received the temporary key. Sometimes, only a partial handshake is needed, in that only one party is concerned about the receipt of the temporary key by the other party. One such case is when subsequent communication immediately follows authentication and thus automatically completes the full handshake.

It is not difficult to analyze the achievable lower bounds in such cases, since if the direction of the partial handshake is the same as that of the last message, then no more message or round is needed than in the authentication-only (AO) cases: the handshake can always be piggy-backed on the last message. However, if the two directions are opposite each other, then the lower bounds are the same as the authentication-with-handshake (AH) cases.

In the discussion so far, handshake is treated as an extension of basic authentication in that clients use handshake messages to inform each other that the temporary key has been satisfactorily received. Handshakes can also be used to show the parties' presence to each other. Again, sometimes only a partial handshake is needed in that only one party's presence must be shown to the other party. For example, after initial authentication completes and the clients share a temporary key, the clients may want to perform handshake at a later stage, or perform handshake repeatedly over a period of time [Needham 78]. Therefore, it is worthwhile to examine the cost of such independent and possibly partial handshakes with regard to the use of timestamps and nonces, without the possibility of piggybacking handshake on earlier messages. The results can be easily worked out, and we skip the details and summarize the results in the following table.

number of msg/round	clock-based	nonce-based
half handshake	1/1	3/2
full handshake	2/2	3/3

Table 3: Lower Bounds for Independent Handshakes

Here the benefit of using timestamps is greater than before – in the case of partial handshake, two out of three messages can be reduced. This is reminiscent of one-way authentication [Needham 78], which cannot be done only with nonces.

In light of the above observation, perhaps the most economical scheme is to include clock synchronization (if clocks are not already synchronized) as part of the initial authentication and use timestamps in all following communications. For example, in Kerberos if the client's initial message contains a wrong timestamp, the server rejects the request but returns a current timestamp to the client for synchronizing the clock and preparing a subsequent request [Neuman 93].

5 Using Uncertified Keys

We now examine the case we excluded earlier: the use of uncertified keys. It does not make sense to exchange uncertified keys when all parties have synchronized clocks, since all that is necessary is to include a timestamp in the key distribution message. Thus, we only consider nonce-based scenarios. Moreover, it is insecure to let a client use an uncertified key solely generated by another client, because attacks replaying past messages will succeed. However, when both clients participate in choosing the final temporary key (CC), it is safe to try a temporary key derived from a received uncertified key and a self-generated key as long as the latter is secret and fresh. Therefore, we need to consider only two scenarios, NB+AO+CC and NB+AH+CC.

Similar to the analysis performed so far, we can clear see that to merely exchange uncertified keys (via the server), four messages and three rounds are optimal in the case of NB+CC. These numbers are identical to those in the clock-based authentication-only (TB+AO+CC) case. This is not surprising since exchanging uncertified keys removes the demand for freshness identifiers and thus can be as efficient as when all parties have synchronized clocks. However, exchanging an uncertified key is not equivalent to key distribution (i.e., authentication-only) because the temporary key is yet uncertified.

For the NB+AH+CC case, at least one more message is needed than the above simple case, and it is not difficult to construct a protocol to achieve the five-message lower bound.

1. A → S: $A, B, \{A, S, A, K1, B\}_{Kas}, Na$
2. S → B: $A, B, \{S, B, A, K1, B\}_{Kbs}, Na$
3. B → S: $\{B, S, B, K2, A\}_{Kbs}, \{B, A, Na\}_K, Nb$
4. S → A: $\{S, A, B, K2, A\}_{Kas}, \{B, A, Na\}_K$
5. A → B: $\{A, B, Nb\}_K$

As for the number of rounds, the responder has to be notified first, then choose and send a candidate temporary to the originator via the server (which requires two rounds), and finally receive a handshake message. Thus four rounds is a lower bound. We can rearrange the messages in the above protocol to achieve this lower bound.

1. A → B: A, B, Na
2. A → S: $A, B, \{A, S, A, K1, B\}_{Kas}$
3. S → B: $A, B, \{S, B, A, K1, B\}_{Kbs}, Na$
4. B → S: $\{B, S, B, K2, A\}_{Kbs}, \{B, A, Na\}_K, Nb$
5. S → A: $\{S, A, B, K2, A\}_{Kas}, \{B, A, Na\}_K$
6. A → B: $\{A, B, Nb\}_K$

To summarize (see Table 4 below), compared with the authentication-only case NB+AO+CC, exchanging uncertified keys uses one less message and one less round but functionally achieves less because temporary keys remain uncertified to at least one client.[3]

[3]It can be arranged so that one client can certify the key with four messages and three rounds. For example, in the above message-optimal protocol, A can detect replay attacks after receiving message 4 by checking if that handshake message contains Na. To detect replay without the handshake message, we can include Na in the two key distribution messages 3 and 4, as in $\{B, S, B, K2, A, Na\}_{Kbs}$ and $\{S, A, B, K2, A, Na\}_{Kbs}$, since B would know Na after receiving message 2. A similar modification can be made to the above round-optimal protocol.

msg/round	Key Exchange Only	AH
NB+CC	4/3	5/4

Table 4: Lower Bounds When Using Uncertified Keys

In the case of NB+AH+CC, however, using uncertified keys can save one message and one round while achieving the same functionality as when uncertified keys are not allowed (compare Table 4 with Table 2). However, these advantages are only theoretical, because, as we have already pointed out at the end of section 3.13, when there is a single trusted server and conventional cryptosystem is used, letting both clients participate in choosing the temporary key costs more but gains nothing in security. In fact, letting the server choose the key is preferable to using uncertified keys, because the former is more efficient and probably more secure than the latter.

6 Using Public-Key or Mixed-Key Systems

When users and systems are sufficiently equipped (in hardware or software), they may wish to take advantage of the public-key systems, since the authentication server no longer has to know clients' private keys and thus does not necessarily know the temporary keys. The server may still impersonate clients by giving false information about a client's public key, but cannot compromise the security of a connection that has been legitimately established.[4]

The use of public-key systems does not obscure the need for traditional cryptosystems. For example, a privileged user who is equipped with a smart card will want to use public-key systems for improved security, but on the day he leaves the card at home, he should still be able to authenticate "normally" with his password. The flexibility for clients to choose dynamically between public-key and conventional systems is not generally available in today's authentication systems.

We now investigate the possible impacts on the numbers of messages and rounds if public-key or mixed-key systems are used. We assume that before an execution of the authentication protocol, a client and the server do not share a secret key, but each knows the other's public key, and one client does not know the other client's public key. There are several possible scenarios to consider.

6.1 Exchanging Preregistered or Dynamic Public Keys

In the simplest scenario, the purpose of authentication is to let the two clients know each other's public key (e.g., by way of a certificate issued by the server) after the protocol completes. The clients may or may not want a handshake at this stage, because presumably they will use the distributed public keys to establish a temporary key for future communication.

Note that a handshake message using public key systems has a different format from that when using conventional cryptosystems. A handshake message from client A to B includes a freshness identifier signed with A's private key and then encrypted with B's private key.

It is not difficult to check that, with or without handshake, letting the server distribute two previously registered client public keys is identical in terms of messages and rounds to the case when the server chooses the temporary key (SO).

If both clients wish to choose a temporary public key to be distributed to the other client, it is easy to check that this case is identical to when both clients participate in choosing the temporary key (CC).

6.2 Diffie-Hellman Type Key Exchange

After receiving each other's public key, the clients may wish to establish a temporary key in the style of Diffie-Hellman. In section 5, we discussed the use of uncertified keys in conventional cryptosystems. Diffie-Hellman key-exchange protocols also use uncertified keys, but in public-key systems. For the two clients to exchange their independently chosen random numbers [Diffie 76], clearly two messages and two rounds are optimal. Again, this is not equivalent to the case of authentication only (AO) because the temporary key is yet uncertified. To certify the temporary key, at least one more message and one more round are needed. We can easily check that three messages and three rounds are indeed achievable lower bounds. Note that in the Diffie-Hellman key exchange, the random number each client chooses is supposed to be fresh, so no other separate freshness identifiers are necessary, and thus there is no difference between TB and NB cases. The results are shown in Table 5 below.

	Key Exchange Only	AH
msg/round	2/2	3/3

Table 5: Diffie-Hellman Type Key Exchange

6.3 Using Mixed-Key Systems

It is quite possible that the two clients will wish to establish a secret temporary key (for a conventional cryptosystem) directly, without first exchanging their public keys. In the analysis given for the 12 cases (for using conventional cryptosystems), the lower bounds on messages and rounds are constrained mainly by freshness requirement (i.e., the delivery of nonces) and are independent of the cryptosystems used; thus the lower bounds remain the same in all 12 cases for the mixed-key system just described, all proofs are still valid, and protocols achieving the bounds can be easily constructed. We can further deduce that if the two clients communicate with the server with two different types of cryptosystems, no improvement in efficiency will be gained or lost, except that encryption and decryption are typically less efficient for public-key systems than for conventional cryptosystems.

In mixed-key systems, it makes sense to let the server *and* the clients participate in choosing the temporary key so that the server does not know the final temporary key. Let us use SC to denote the case when the server and one client choose the key, and use SCC to denote that all three parties choose the key. We have worked through all clock-based cases, and, without bothering the reader with details, we conclude that the (achievable) lower bounds for the case SC are identical to those for the case SO, and the (achievable) lower bounds for case SCC are identical to those for the case CC. We have yet to examine all the nonce-based cases.

[4] For a discussion of how to use *distributed authentication* to deal with potentially dishonest servers, see [Gong 93].

6.4 Summary

In Table 6 below, we summarize the equivalence (in terms of lower bound) between protocols using public key systems and those using conventional systems.

public-key or mixed-key system	conventional system
preregistered public keys	SO
dynamic public keys	CC
mixed-key TB+SC	TB+SO
mixed-key TB+SCC	TB+CC

Table 6: Systems Equivalent in Lower Bounds

7 Multiple Clients Establishing Group Keys

We now generalize our previous analysis to multiple clients. We discuss only the use of conventional cryptosystems and, as before, results on public-key or mixed-key systems can be similarly worked out. We keep the argument brief and, instead of specifying example protocols in full, we only outline the non-trivial ones. All lower bounds proven are achievable.

Suppose that a total of n clients want to establish a group key and $n \geq 3$. Now CC stands for the scenario in which all clients contribute to the choice of the temporary key, and AH means that every client receives a handshake message from every other client. To distinguish these cases about setting up group keys from previous cases, we add prefix G+ to all cases' names here, so we have case G+TB+AO+SO instead of TB+AO+SO. We designate client A as the protocol originator. For the ease of description, we line up the clients with A at the head of the line and client B at the end of the line. We exclude the use of uncertified keys.

We can now analyze the lower bounds on messages and rounds using similar techniques as before. For example, after the last key distribution message, at least $(n-1)$ more messages and one more round are needed to complete the handshake process. We recall that, in nonce-based cases, when a client contributes to the choice of the temporary key, it must include all other clients' nonces and its key choice in a single encrypted message to the server (to be forwarded to other clients). Otherwise, the server would have difficulty in binding a key to a nonce, and this has security implications. As a result, the client can only complete sending out its key choice after receiving nonces from all other clients.

7.1 Case G+TB+AO+SO

The server has to be notified and, after that, n messages are needed to distribute the group key; thus $n+1$ messages is a lower bound. The protocol is simple: A notifies the server who then sends n messages to distribute a key to all clients. Trivially, two rounds is a lower bound.

7.2 Case G+TB+AH+SO

Compared with the above case (G+TB+AO+SO), at least $n-1$ more messages (for the last handshake) are needed; thus $2n$ messages is a lower bound, which is achieved in the following protocol.

A notifies S who then prepares a package of n separately encrypted key distribution messages, each addressed to a different client, and returns the whole package to A in a single message. A takes out the message addressed to it, obtains the temporary key to compute a handshake message, puts this handshake message back into the package, and passes the package down the line of clients. Every client repeats this process until B is reached. B now has all the handshake messages (including its own) and passes them back up the line until A is reached. The total number of messages is $1+n+(n-1)=2n$.

Compared with case G+TB+AO+SO, one more round is needed for handshake; thus three rounds is a lower bound. The following protocol achieves this bound. A notifies S who in the second round distributes keys to all the clients. In round three, every client broadcasts a handshake message to all other clients.

7.3 Case G+TB+AO+CO

The client who is the key chooser has to distribute the group key via the server, thus n messages is a lower bound. The protocol is simply that A chooses a key and sends it to S who forwards it to the other $n-1$ clients. Trivially, two rounds is a lower bound.

7.4 Case G+TB+AH+CO

Compared with case G+TB+AO+CO, $n-1$ more messages are needed for the last handshake; thus $2n-1$ messages is a lower bound, which is achieved in the following protocol.

A sends its selection of the group key together with a handshake message to S, who then prepares a package of $n-1$ key distribution messages and sends it together with A's handshake message to the client next to A (in the line of clients). Starting from there, every client picks up the key distribution message for it, puts back in a handshake message, and passes the package down the line. B then passes all the handshake messages up the line again. The total number of messages in this protocol is $1+1+(n-2)+(n-1)=2n-1$.

One more round is needed than case G+TB+AO+CO thus three rounds is a lower bound which is achieved by letting S send A's choice of the group key to all other clients in parallel and letting the clients exchange handshake messages in the third round.

7.5 Case G+TB+AO+CC

Each client has to send its choice of key to other clients via the server and receive the choices of others via the server, thus $2n$ messages is a lower bound which is achieved in the following protocol.

Starting from A, each client chooses its key. encrypts it for S to read, and passes it down the line. B passes the whole package to S. S then puts all key distribution messages in one package and sends it to B. Starting from B, each client picks up the message for it and passes the rest up the line. The total cost is $2n$ messages.

One round of messages is needed to notify all the clients and two more rounds are needed to exchange keys via the server; thus three rounds is a lower bound which is trivially achievable.

7.6 Case G+TB+AH+CC

The extra handshake process needs $n-1$ more messages than case G+TB+AO+CC, thus $3n-1$ messages is a lower bound, which is achieved in the following protocol, which is adapted from the one in case G+TB+AO+CC.

When the key distribution package is passed back up the line, each client picks up its key distribution message and puts back in its handshake message before passing on the package. Once the package (containing all handshake messages) reaches A, the package is passed back down the line to B.

One more round is needed than case G+TB+AO+CC, so four rounds is a lower bound.

7.7 Case G+NB+AO+SO

Every client has to send a nonce to the server and later gets back a key from the server, thus $2n$ messages is a lower bound. To see why these messages cannot be further combined, one client's sending a nonce and receiving a key must be done with two distinct messages. Moreover, if a client's sending a nonce is done in the same message with another client's receiving a key, then the former client has to receive the latter client's key from somewhere else with a message that cannot be combined with another message; thus we can arrange so that the latter client receives its key directly, without using the former client has an intermediate party. In other words, combining the $2n$ messages will not reduce the total number of messages required.

The following protocol achieves the lower bound. Starting from A, each client selects its nonce and passes the package down the line. (These messages also serve to notify responders.) B then sends the whole package to S. The server sends the key distribution messages in one package to B. Starting from B, each client picks up the message addressed for it and passes the rest of the package up the line. This requires a total of $2n$ messages.

A responder has to be notified before it can send out a nonce and then receives a key; thus three rounds is a lower bound.

7.8 Case G+NB+AH+SO

Compared with case G+NB+AO+SO, another $n-1$ messages are needed for the handshake; so $3n-1$ messages is a lower bound which is achieved in the following protocol. In the protocol for case G+NB+AO+SO, a client takes out its key message and puts back a handshake message before passing the package up the line. A passes the entire package of handshake messages back down the line again.

One more round is required than case G+NB+AO+SO, so four rounds is a lower bound which can be easily achieved.

7.9 Case G+NB+AO+CO

Every client (except the key chooser) has to send out a nonce and receive a key via the server. This requires $2(n-1)$ messages (recall the proof argument in case G+NB+AO+SO). The key chooser must distribute the key via the server, requiring one more message. Thus $2n-1$ messages is a lower bound, which is achieved in the following protocol.

Starting from A, every client adds a nonce to the package and passes it down the line to B. B chooses a key, and sends all $(n-1)$ nonces and the key in one encrypted message to S. The server then prepares $n-1$ key distribution messages and passes them to the client just before B in the line of clients (call it C). Starting from C, each client takes out the message for it and passes the rest of the package up the line. The total number of messages is $2n-1$.

Those clients who are not the protocol originator or the key chooser need to be notified before being able to send out their nonces. These exchanges use two rounds. Two more rounds are needed for them to receive the keys via the server, thus four rounds is a lower bound (when $n \geq 3$), which is trivial to achieve.

7.10 Case G+NB+AH+CO

Compared with case G+NB+AO+CO, $n-1$ more messages are needed for handshake; thus $3n-2$ messages is a lower bound, which is achieved with the protocol for case G+NB+AO+CO with the following modification. Together with the $n-1$ nonces, B also sends its handshake message to S, who then includes it in the package sent to C later. Starting from C, every client takes out its message and puts back in a handshake message before passing the package up the line. A passes all the handshake messages down the line again.

One more round is needed than case G+NB+AO+CO, thus five rounds is a lower bound (when $n \geq 3$) which is also easy to achieve.

7.11 Case G+NB+AO+CC

Each client has to send out a nonce to all other clients. This requires $2(n-1)$ messages because $n-1$ messages are needed for one client to send a nonce to all other clients and the last client receiving the nonce will need a further $n-1$ messages to distribute his nonce. This last client will then need n more messages to receive $n-1$ keys via the server. Thus $3n-1$ messages is a lower bound.

The protocol achieving the lower bound is as follows. Starting from A, every client adds a nonce to the package and sends it down the line. Starting from B, every client uses all the nonces in the package to prepare its key distribution message (to be sent to S), adds this message to the package, and sends it up the line again. A then forwards to S all key distribution messages, who then returns key distribution messages addressed to individual clients in a whole package to A. A picks up the message for it and passes the rest down the line. This requires a total of $3n-1$ messages.

Two rounds are needed for a responder to be notified and to send a nonce. Two more rounds are needed for the clients to send keys via the server, thus four rounds is a lower bound which is trivially achievable.

7.12 Case G+NB+AH+CC

Compared with case G+NB+AO+CC, $n-1$ more messages are needed for handshake. The lower bound of $4n-2$ messages is achieved by modifying the protocol for case G+NB+AO+CC in the following way. Starting from A, after picking up the key distribution message, every client puts back in its handshake message before passing the package down the line. B then passes the entire handshake message up the line again. One more round is needed than case G+NB+AO+CC, thus five rounds is a lower bound which is trivial to achieve.

7.13 Summary

The above results on establishing group keys are summarized in Table 7.

We can see that nonce-based cases cost $n-1$ more messages than the corresponding clock-based cases, and protocols completing handshakes cost $n-1$ more messages than similar protocols with no or only partial handshakes. Moreover, CO cases use one less message than the corresponding

msg/ round	authentication only (AO)		
	SO	CO	CC
TB	n+1/2	n/2	2n/3
NB	2n/3	2n-1/4	3n-1/4
msg/ round	authentication with handshake (AH)		
	SO	CO	CC
TB	2n/3	2n-1/3	3n-1/4
NB	3n-1/4	3n-2/5	4n-2/5

Table 7: Multiple Clients ($n \geq 3$) with One Server

SO cases, while CC cases use $n-1$ more messages than the corresponding SO cases.

We can also observe that the results on lower bound on messages in the above table are consistent with our previous analysis (see Table 2) if $n = 2$.

Results on lower bound on rounds are identical to previous analysis, except in the cases of G+NB+AO+CO and G+NB+AH+CO where one more round is needed when $n \geq 3$.

8 Conclusions and Future Work

We have conducted a systematic study of the optimality (in terms of the numbers of messages and rounds) of network authentication protocols. Our analysis is based on a set of typical security-related assumptions in distributed systems and is aimed at a set of typical authentication scenarios. Our results not only confirm some widely held beliefs regarding the relative merits of some authentication techniques but also can guide system and protocol designers in providing optimal authentication services or flexible authentication systems with which clients can, according to individual need, make dynamic trade-offs between efficiency parameters.

For future work, it is desirable to identify protocols that are both message-optimal and round-optimal, such as the protocols in cases 1, 3, 4, 7, 9, and 10. It seems unlikely that there exist message-optimal and round-optimal protocols for all cases, especially the group key cases; thus some impossibility results will be very interesting.

Another possibility is to investigate metrics or scenarios other than those already discussed. For example, how many encryptions are needed in any particular scenario? Since it has been shown that mutual authentication can be achieved based on one-way hash functions [Gong 89], what is a suitable definition of encryption? Or, is there an optimal sequence of messages in the sense that more and/or higher order of beliefs [Burrows 89] can be achieved with the same number of messages? Minimizing interactions with the (probably heavily loaded) server can also be beneficial.

Finally, we note that the efficiency of a protocol cannot be fully characterized independently of its implementation details. For example, depending on the actual message sizes and the network environment, adding a few extra bytes may cost very little, or it may cost a lot by introducing additional message fragmentation at lower levels. There are other considerations that cannot be easily abstracted. For example, approaches using timestamp or verifier-issued timestamps (as nonces) reduce the server states per-connection and thus increase performance [Neuman 93]. Also, a protocol that can piggyback its last message on the opening message of a subsequent communication may yield better overall performance than another protocol that cannot piggyback, even if they have the same numbers of messages and rounds.

Acknowledgments

I started working on these lower bounds as a research student under Professors D.J. Wheeler and R.M. Needham at Cambridge University, who reviewed the original manuscript ("Minima in Authentication Protocols," unpublished, March 1989). I also benefited from discussions with many other colleagues. Cliff Neuman and Stuart Stubblebine of USC/ISI provided detailed comments on a more recent draft.

References

[Birrell 85] A.D. Birrell, "Secure Communications Using Remote Procedure Calls," *ACM Transactions on Computer Systems*, Vol.3, No.1, February 1985, pp.1-14.

[Bird 93] B. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung, "Systematic Design of a Family of Attack-Resistant Authentication Protocols," to appear in *IEEE Journal on Selected Areas in Communications*, 1993.

[Burrows 89] M. Burrows, M. Abadi, and R.M. Needham, "A Logic for Authentication," DEC System Research Center Technical Report No. 39, February 1989.

[Denning 81] D.E. Denning and G.M. Sacco, "Timestamps in Key Distribution Protocols," *Communications of the ACM*, Vol.24, No.8, August 1981, pp.533-536.

[Diffie 76] W. Diffie and M.E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, Vol. IT-22, No.6, November 1976, pp.644-654.

[Gong 89] L. Gong, "Using One-Way Functions for Authentication," *ACM Computer Communication Review*, Vol.19, No.5, October 1989, pp.8-11.

[Gong 92] L. Gong, "A Security Risk of Depending on Synchronized Clocks," *ACM Operating Systems Review*, Vol.26, No.1, January 1992, pp.49-53.

[Gong 93] L. Gong, "Increasing Availability and Security of an Authentication Service," *IEEE Journal on Selected Areas in Communications*, Vol.11, No.5, June 1993.

[Lakatos 76] I. Lakatos, "Proofs and Refutations," J. Worrall and E. Zahar (Eds.), Cambridge University Press, 1976.

[NBS 77] U.S. National Bureau of Standards, "Data Encryption Standard," U.S. Federal Information Processing Standards Publication, FIPS PUB 46, January 1977.

[Needham 78] R.M. Needham and M.D. Schroeder, "Using Encryption for Authentication in Large Networks of Computers," *Communications of the ACM*, Vol.21, No.12, December 1978, pp.993-999.

[Needham 87] R.M. Needham and M.D. Schroeder, "Authentication Revisited," *ACM Operating Systems Review*, Vol.21, No.1, January 1987, p.7.

[Neuman 93] B.C. Neuman and S.G. Stubblebine, "A Note on the Use of Timestamps as Nonces," *ACM Operating Systems Review*, Vol.27, No.2, April 1993, pp.10-14.

[Otway 87] D. Otway and O. Rees, "Efficient and Timely Mutual Authentication," *ACM Operating Systems Review*, Vol.21, No.1, January 1987, pp.8-10.

[Rivest 78] R.L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, Vol.21, No.2, February 1978, pp.120-126.

[Yahalom 93] R. Yahalom, "Optimality of Multi-Domain Protocols" (draft), April 1993, to appear in this proceedings.