AUTHENTICATION FOR DISTRIBUTED SYSTEMS

Thomas Y. C. Woo and Simon S. Lam

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712-1188

TR-91-19 May 1991

Authentication for Distributed Systems[*]

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

April 16, 1991

Abstract

A fundamental concern in building a secure distributed system is *authentication* of various local and remote entities in the system. We survey authentication issues in distributed system design. Some basic paradigms underlying the design of authentication protocols are presented. An authentication framework that can be used for the design of secure distributed systems is proposed. Three specific authentication protocols relevant to this framework are presented. We conclude with an overview of two existing authentication systems.

Keywords: Authentication, Distributed Systems, Networks, Protocols, Security.

[*]This work was supported in part by a grant from the Texas Advanced Research Program and by National Science Foundation grant no. NCR-9004464.

Contents

1 Introduction

A *distributed system* can be loosely understood to be a collection of hosts interconnected by a network. The hosts communicate by sending and receiving messages over the network. Various resources (e.g. files, printers) are distributed among the hosts and shared across the network in the form of *network services* provided by *servers*. Individual processes (*clients*), desiring access to resources, direct their service requests to the appropriate servers. Aside from such client-server computing, there are many other reasons for having a distributed system. For example, a task can be divided up into subtasks that are executed concurrently on different hosts.

A distributed system is susceptible to a variety of threats mounted by intruders as well as legitimate users of the system. Indeed, legitimate users are more powerful adversaries since they possess internal state information not usually available to an intruder (except after a successful penetration of a host). We identify two general types of threats.

The first type is called *host compromise threats*; this refers to the subversion of individual hosts in a system. Various degrees of subversion are possible, ranging from the relatively benign case of corrupting a process's state information to the extreme case of assuming total control of a host. Host compromise threats can be countered by a combination of hardware techniques (e.g. processor protection modes) and software techniques (e.g. *security kernel/reference monitor*). These techniques are outside the scope of this paper and interested readers are referred to [11] for an overview of the area of computer systems security. In this paper, we assume that each host implements a reference monitor that can be trusted to properly segregate processes.

The second type of threats is called *communication threats*; this includes threats associated with message communications, and can be further subdivided into [24]:

(T1) eavesdropping of messages transmitted over network links to extract information on private conversations,

(T2) arbitrary modification, insertion, and deletion of messages transmitted over network links to confound a receiver into accepting fabricated messages,

(T3) replay of old messages; this can be considered a combination of (T1) and (T2).

(T1) is a *passive* threat, while (T2) and (T3) are *active* threats. A passive threat is one that does not affect the system being threatened, whereas an active threat does. Therefore, passive threats are inherently undetectable by the system, and can only be dealt with using preventive measures. For active threats, a combination of prevention, detection, and recovery techniques can be employed. Additionally, there are threats of "traffic analysis" and "denial of service"; we will not consider these because they are more relevant to the general security of a distributed system than our restricted setting of authentication.

Corresponding to these threats, some basic security requirements can be formulated. For examples, *secrecy* and *integrity* are two common requirements for secure communication. Secrecy specifies that a message can be read by its intended recipients only, while integrity specifies that every message is received exactly as it was sent, or a discrepancy is detected.

A strong cryptosystem can provide a high level of assurance on both the secrecy and integrity of messages. More precisely, an encrypted message provides no information regarding the original message, hence guaranteeing secrecy; and an encrypted message, if tampered, would not decrypt into a legal message, hence guaranteeing integrity.

Replay of old messages can be countered by using *nonces* or *timestamps* [8, 19]. Nonces are information that is guaranteed to be *fresh*, i.e. has never appeared or been used before. Therefore, a reply that contains some function of a nonce sent recently should be believed to be timely—in particular, it could have been generated only after the nonce was sent. Perfect random numbers are good candidates to be nonces;

however their effectiveness is dependent upon the randomness that is practically achievable. Timestamps are values of a local clock. Their use requires at least some loose synchronization of all local clocks, and hence their effectiveness is also somewhat restricted.

The balance of this paper is organized as follows. In Section 2, we discuss what authentication means as well as the various authentication needs in distributed systems. In Section 3, two classes of paradigms of authentication protocols are presented. In Section 4, we discuss why realistic authentication protocol are difficult to design. In Section 5, we propose an authentication framework for distributed systems, and present specific authentication protocols that can be used within the framework. In Section 6, we describe authentication protocols in two existing systems: Kerberos and SPX. In Section 7, we present some conclusions.

2 What Needs to be Authenticated?

In simple terms, authentication is identification plus verification. *Identification* is the process whereby an entity claims a certain identity, while *verification* is the process whereby such a claim is checked. Thus the *correctness* of an authentication relies heavily on the verification procedure employed.

The various entities in a distributed system that can be distinctly identified are collectively referred to as *principals*. There are three main types of authentication of interest in a distributed computing system:

(A1) *message content authentication* — verifying that the content of a message received is the same as it was sent.

(A2) *message origin authentication* — verifying that the sender of a message received is as recorded in the sender field of a message.

(A3) *general identity authentication* — verifying that the identity of a principal is as claimed.

(A1) is commonly handled by tagging a key-dependent *message authentication code* (MAC) onto a message before it is sent. Integrity of the message can be confirmed upon reception by recomputing the MAC and comparing it with the one attached. (A2) can be considered as a subcase of (A3). In general, a successful general identity authentication results in a state of belief in the authenticating principal (the *verifier*) that the authenticated principal (the *claimant*) possesses the claimed identity. Hence subsequent actions performed by the claimant are attributable to the claimed identity; e.g. general identity authentication is needed for both *authorization* and *accounting* functions. In this paper, we restrict our attention to general identity authentication only.

In an environment where both host compromise threats and communication threats are possible, principals must adopt a *mutually suspicious* attitude toward one another. Therefore, *mutual authentication*, whereby both communicating principals verify each other's identity, rather than *one-way authentication*, whereby only one principal verifies the identity of the other principal, is usually required.

In a distributed computing environment, authentication is carried out using a protocol involving message exchanges. We refer to these protocols as *authentication protocols.*

Most existing systems employ only very primitive authentication measures or none at all, for examples:

- The prevalent login procedure requires users to enter their passwords in response to a system prompt. Users are then one-way authenticated by verifying the (possibly transformed) password against an internally stored table. However, no mechanism is available to the users to authenticate a system. Such a design is acceptable only when the system is trustworthy, or the probability of compromise is low.

Basic Cryptography

A cryptosystem comes with two procedures, one for *encryption* and one for *decryption*. A formal description of a cryptosystem consists of the specification of its *message space*, *key space*, *ciphertext space*, the encryption function and decryption function.

There are two broad classes of cryptosystems, namely *symmetric* and *asymmetric* cryptosystems [4]. In a symmetric cryptosystem, the encryption and decryption keys are the same and hence need to be kept secret. In an asymmetric cryptosystem, the encryption key is different from the decryption key and only the decryption key needs to be kept secret while the encryption key can be made public. Because of this, it is important that the decryption key cannot possibly be determined from the encryption key. Symmetric and asymmetric cryptosystems are also referred to as *shared key* and *public key* cryptosystems, respectively.

Knowledge of the encryption key allows one to encrypt arbitrary messages from the message space, while knowledge of the decryption key allows one to recover a message from its encrypted form. Thus, the encryption and decryption functions satisfy the following relation: $\mathcal{M}$ is the message space, $K_E \times K_D$ is the set of encryption/decryption key pairs:

$$\forall m \in \mathcal{M} : \forall (k, k^{-1}) \in K_E \times K_D : \{\{m\}_k\}_{k^{-1}} = m \tag{C1}$$

where $\{x\}_y$ denotes the encryption operation on x if y is an encryption key, and the decryption operation on x if y is a decryption key. (In the case of a symmetric cryptosystem whose encryption and decryption keys are the same, the operation performed should be clear from context.)

Two widely used cryptosystems are the Data Encryption Standard (DES) [1], a symmetric system, and RSA [3], an asymmetric system. In the case of RSA, encryption-decryption key pairs satisfy the following commutative property [2]:

$$\forall m \in \mathcal{M} : \forall (k, k^{-1}) \in K_E \times K_D : \{\{m\}_{k^{-1}}\}_k = m \tag{C2}$$

hence yielding a *signature* capability: i.e. suppose k and k^{-1} are P's asymmetric keys, then $\{m\}_{k^{-1}}$ can be used as P's signature on m since it could only have been produced by P, the only principal who knows k^{-1}. By (C2), P's signature is verifiable by any principal with knowledge of k, P's public key. Note that in (C2), the roles of k and k^{-1} are reversed; specifically, k^{-1} is used as an encryption key while k as a decryption key. To avoid confusion with the more typical roles for k and k^{-1} as exemplified in (C1), we refer to encryption by k^{-1} as a *signing* operation. In this paper, asymmetric cryptosystems are assumed to be commutative.

Since in practice, symmetric cryptosystems can operate much faster than asymmetric ones, asymmetric cryptosystems are often used only for initialization/control functions, while symmetric cryptosystems can be used for both initializations and actual data transfer.

References

[1] "Data Encryption Standard," FIPS Pub 46, National Bureau of Standards, Washington, D.C., January 15, 1977.

[2] W. Diffe and M.E. Hellman, "Privacy and Authentication: An Introduction to Cryptography," *Proceedings of the IEEE*, Vol. 67, No. 3, pp. 397–427, March 1979.

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

[4] G.J. Simmons, "Symmetric and Asymmetric Encryption," *ACM Computing Survey*, Vol. 11, No. 4 pp. 305–330, 1979.

- In a typical client-server interaction, the server, on accepting a client's request, has to trust that (1) the resident host of the client has correctly authenticated the client, and (2) the identity supplied in the request actually corresponds to the client. Such trust is valid only if the system's hosts are trustworthy and its communication channels are secure.

The above measures are seriously inadequate for the following reasons: First, the notion of trust in distributed systems is poorly understood; a satisfactory formal explication of trust has yet to be proposed. Second, the proliferation of large-scale distributed systems spanning multiple administrative domains has given rise to extremely complex trust relationships.

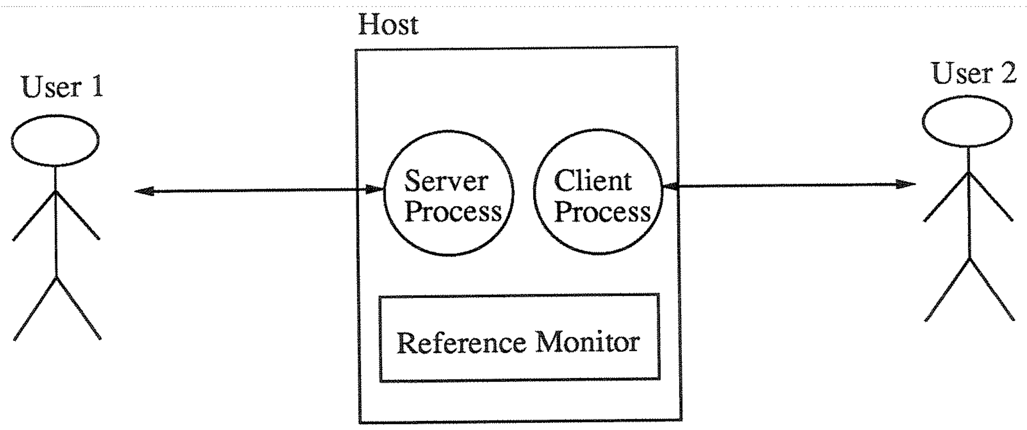

Figure 1: Principals in a Distributed System

In a distributed computing system, the entities that require identification are hosts, users and processes [15]; hence they constitute the principals involved in an authentication. We describe them in the following (see Fig 1):

- *hosts* — a host is an addressable entity at the network level, which is distinguished from its underlying supporting hardware. For example a host H running on workstation A can be moved to run on workstation B if desired by performing the bootstrapping sequence for H on B. A host is usually identified by its host name (e.g. a domain name) or its network address (e.g. an IP address), whereas a particular host hardware is usually identified by its factory assigned serial number e.g. a workstation on a Ethernet can be identified by the unique Ethernet address of its Ethernet adaptor board.

- *users* — users are the entities ultimately responsible for all system activities. In other words, users initiate and are accountable for all system activities. Most access control and accounting functions are based on users. (For completeness, a special user called *root* can be postulated, who is accountable for system-level activities, e.g. process scheduling.) Typical users include human users, as well as accounts maintained in the user database. Note that users are considered to be outside the system boundary.

- *processes* — processes are created by the system to represent users. They lie within the system boundary and each of them is associated with a unique user. A process requests and consumes resources on behalf of its associated user. Processes can be broadly divided into two classes: client processes and server processes. Client processes are *service consumers* who obtain services from the server processes, the *service providers.* A particular process can act as both a client and a server. For example, print servers are usually created by (hence associated with) the user *root*, and act as servers for printing requests by other processes, but as clients when requesting files from file servers.

Corresponding to the various principals, we identify the following major types of authentication exchanges in a distributed system:

- *host-host* — cooperation between hosts is often required for host-level activities. For example, *topology maintenance* is typically done in a distributed manner such that individual hosts exchange link information and update their internal topology maps accordingly. Another example is *remote bootstrapping.* A host upon reinitialization, must be able to identify a trustworthy *boot server* to supply the information (e.g. a copy of the operating system) required for correct initialization.
- *user-host* — a user gains access to a distributed system by logging onto a host in the system. In an open access environment where hosts are scattered across unrestricted areas, a host can be arbitrarily compromised and hence mutual authentication is required between the user and the host.
- *process-process* — two main subclasses can be identified:
  1. peer process communication — peer processes must be satisfied with each other's identity before private communication can begin.
  2. client-server communication — an access decision concerning a client's request can be made only when the client's identity is affirmed; a client is willing to surrender valuable information to a server only after it has verified the identity of the server.

  As shown in later sections, these two classes of communication authentication are closely related, and can be handled by similar protocols.

In the sequel, we will use authentication to refer to general identity authentication.

3 Paradigms of Authentication Protocols

Authentication in distributed systems is invariably performed using protocols. A *protocol* is a precisely defined sequence of *communication* and *computation* steps. A communication step transfers messages from one principal (the sender) to another (the receiver), while a computation step updates the internal state of a principal. Two distinct states can be identified upon termination of the protocol, one signifying successful authentication and the other failure.

Although the goal of any authentication is to verify the claimed identity of a principal, the specific success and failure states are highly protocol dependent. For example, the success of an authentication during the connection establishment phase of a communication protocol is usually indicated by the distribution of a fresh *session key* between two mutually authenticated peer processes; whereas in a user login authentication, success usually results in the creation of a login process on behalf of the user.

We present protocols in the following format: a communication step whereby P sends a message M to Q is represented as:

$$P \rightarrow Q : \qquad M$$

whereas a computation step of P is written as:

$$P : \qquad \ldots$$

where "..." is some specification of the computation step. As an example, the typical login protocol between a host H and a user U is given below: f is a *one-way* function, that is, given y it is computationally infeasible to find an x such that $f(x) = y$.

Approaches to Authentication

All authentication procedures involve the checking of some known information about a claimed identity against information acquired from the claimant during the identity verification procedure. Such checking can be based on the following three approaches [2]:

- Proof by *knowledge* — the claimant demonstrates knowledge of some information regarding the claimed identity that can only be known/produced by a principal with the claimed identity. For example, the knowledge of a password is used in most login procedures. A proof by knowledge can be conducted by a direct demonstration of the knowledge e.g. typing in a password, or by an indirect demonstration e.g. correctly computing replies to challenges posed by the verifier. Direct demonstration is not preferable from a security viewpoint since a compromised verifier can record the submitted knowledge and later impersonate the claimant by presenting the recorded knowledge. Indirect demonstration can be designed to induce high confidence in the verifier, without leaving any clue on how the claimant's replies are computed. For example, in [1], a *zero-knowledge* protocol for proof of identity is proposed. This protocol allows a claimant C to prove to a verifier V that C knows how to compute replies to challenges posed by V without revealing what the replies are. These protocols are provably secure (under complexity assumptions). However, additional refinements are needed before they can be applied in practical systems.
- Proof by *possession* — the claimant produces an item that can only be possessed by a principal with the claimed identity. For example, an ID badge. For this to work, the item has to be unforgeable and be safely guarded to avoid theft.
- Proof by *property* — this involves the direct measurement of certain properties of the claimant by the verifier. For example, various biometric techniques are used: finger print, retina print, and so on. The measured property has to be distinguishing, i.e. unique among all possible principals, to achieve correct identification.

Proof by knowledge and possession (and combinations thereof) can be applied to all types of authentication needs in a secure distributed system while proof by property is generally limited to the authentication of human users by a host equipped with specialized measuring instruments.

References

[1] U. Feige, A. Fiat and A. Shamir, "Zero Knowledge Proofs of Identity," *Proceedings of the ACM Symposium on Theory of Computing*, pp. 210–217, 1987.

[2] K. Shankar, "The Total Computer Security Problem," *IEEE Computer*, Vol. 10, No. 6, pp. 50–73, June 1977.

$U \rightarrow H$	:	U
$H \rightarrow U$	:	"Please enter password"
$U \rightarrow H$	:	p
H	:	compute $x = f(p)$
	:	retrieve user record $(U, f(password_U))$ from user database
	:	if $x = f(password_U)$ then accept; otherwise reject

As in the domain of communication protocols where a large body of existing protocols can be systematically constructed and analyzed by focusing on a few key ideas (i.e., 2-way handshake, 3-way handshake, alternating-bit and sliding windows), we next examine several basic ideas that underlie authentication protocol design.

Since authentication protocols make direct use of cryptosystems, their basic design principles also follow closely the type of cryptosystem used. Specifically, we identify two basic principles for authentication, one based on symmetric cryptosystems and the other on asymmetric cryptosystems.

Note that protocols presented in this section are intended to illustrate basic design principles only. A realistic protocol is necessarily a refinement of these basic paradigms and addresses weaker environment assumptions and/or stronger postconditions. Also, a realistic protocol may make use of both symmetric and asymmetric cryptosystems.

3.1 Protocols Based upon Symmetric Cryptosystems

In a symmetric cryptosystem, knowledge of the shared key enables a principal to encrypt and decrypt arbitrary messages; without such knowledge it is infeasible to obtain the encrypted version of a message, or to decrypt an encrypted message. Hence authentication protocols can be designed based on the following principle:

> *The ability to correctly encrypt a message using a key that is believed to be known only by a principal with the claimed identity (and the verifier) constitutes a sufficient proof of claimed identity by the claimant.* (SYM)

Thus (SYM) embodies the proof by knowledge principle for authentication, i.e., a principal's knowledge is indirectly demonstrated through its ability to encrypt. Using (SYM), we immediately obtain the following basic protocol: k is a symmetric key shared between P and Q.

P	:	create $m =$"I am P."
	:	compute $m' = \{m\}_k$
$P \rightarrow Q$	:	m,m'
Q	:	verify $\{m\}_k \overset{?}{=} m'$
	:	if equal then accept; otherwise reject

Clearly, the design principle (SYM) is sound only if the underlying cryptosystem is strong (i.e. it is impossible to find the encrypted version of a message without knowing the key) and the key is secret (i.e. shared only between the real principal and the verifier). Note that the above protocol performs only one-way authentication, mutual authentication can be achieved by reversing the roles of P and Q. However, a major weakness is its vulnerability toward replays. More precisely, an adversary could masquerade as P by recording the message m' and later replay it to Q.

As mentioned before, replay attacks can be countered using nonces or timestamps. We modify the above protocol by adding a *challenge and response* step using nonces: n is a nonce.

$P \rightarrow Q$	:	"I am P."
$Q \rightarrow P$	:	n
P	:	compute $n' = \{n\}_k$
$P \rightarrow Q$	:	n'
Q	:	verify $\{n\}_k \overset{?}{=} n'$
	:	if equal then accept; otherwise reject

Replay is foiled by the freshness of n. Thus even if a eavesdropper has monitored all previous authentication conversations between P and Q, it would still be unable to produce the correct n'. (This also points out the need for the cryptosystem to withstand *known plaintext* attack, i.e. the cryptosystem is still unbreakable given the knowledge of some plaintext-ciphertext pairs.) The challenge and response step can be repeated any number of times until the desired level of confidence is reached by Q.

The above protocol is impractical as a general large-scale solution because each principal is required to remember the secret key for every other principal he would ever want to authenticate. This presents major initialization (i.e. the predistribution of secret keys) and storage problems. Moreover, the compromise of any one principal can potentially compromise the entire system. These problems can be significantly reduced by postulating a centralized *authentication server* A that shares a secret key k_X with every principal X in the system [19]. The basic authentication protocol then becomes:

$P \rightarrow Q$	:	"I am P."
$Q \rightarrow P$	:	n
P	:	compute $n' = \{n\}_{k_P}$
$P \rightarrow Q$	:	n'
Q	:	compute $n'' = \{P, n'\}_{k_Q}$
$Q \rightarrow A$	:	n''
A	:	recover (P, n') from n'' by decrypting with k_Q
	:	compute $m = \{\{n'\}_{k_P}\}_{k_Q}$
$A \rightarrow Q$	:	m
Q	:	verify $\{n\}_{k_Q} \overset{?}{=} m$
	:	if equal then accept; otherwise reject

Thus the verification step by Q is preceded by a *key translation* step by A. The protocol correctness now also rests on the trustworthiness of A, i.e. it would properly decrypt using P's key and reencrypt using Q's key. The initialization and storage problems are greatly alleviated since each principal now needs to keep only one key. The risk of compromise is mostly shifted to A, whose security can be guaranteed by various measures e.g. physical security and encrypting all stored keys using a master key.

3.2 Protocols Based upon Asymmetric Cryptosystems

In an asymmetric cryptosystem, each principal P publishes his public key k_P and keeps secret his private key k_P^{-1}. Thus only P can generate $\{m\}_{k_P^{-1}}$ for any message m by signing it using k_P^{-1}. $\{m\}_{k_P^{-1}}$ can be verified by any principal with knowledge of k_P (assuming a commutative asymmetric cryptosystem). Thus the basic design principle is:

> *The ability to correctly sign a message using the private key of the principal with the claimed identity constitutes a sufficient proof of the claimed identity by the claimant.* (ASYM)

(ASYM) follows the proof by knowledge principle for authentication, in that a principal's knowledge is indirectly demonstrated through its capability to sign. Using (ASYM), we obtain a basic protocol as follows: n is a nonce.

$P \rightarrow Q$	:	"I am P."
$Q \rightarrow P$	:	n
P	:	compute $n' = \{n\}_{k_P^{-1}}$
$P \rightarrow Q$	:	n'
Q	:	verify $n \stackrel{?}{=} \{n'\}_{k_P}$
	:	if equal then accept; otherwise reject

The above protocol depends on the guarantee that $\{n\}_{k_P^{-1}}$ cannot be produced without the knowledge of k_P^{-1} and the correctness of k_P as published by P and kept by Q.

As in the case of symmetric keys, the initialization and storage problems can be alleviated by postulating a centralized *certification authority* A that maintains a database of all published public keys [19]. The above protocol can then be modified as follows:

$P \rightarrow Q$	:	"I am P."
$Q \rightarrow P$	:	n
P	:	compute $n' = \{n\}_{k_P^{-1}}$
$P \rightarrow Q$	:	n'
$Q \rightarrow A$	:	"I need P's public key."
A	:	retrieve $c = \{P, k_P\}_{k_A^{-1}}$ from key database
$A \rightarrow Q$	:	P, c
Q	:	recover (P, k_P) from c by decrypting with k_A
	:	verify $n \stackrel{?}{=} \{n'\}_{k_P}$
	:	if equal then accept; otherwise reject

Thus c, called a *public key certificate*, represents a certified statement by A that P's public key is k_P. Other information such as a specified lifetime and the classification of principal P (for mandatory access control) can also be included in the certificate; such information is omitted here. Each principal in the system only needs to keep a copy of the public key k_A of A.

In the above protocol, A is an example of an *on-line* certification authority. In other words, A supports interactive queries and is actively involved in authentication exchanges. A certification authority can also operate *off-line* such that a public key certificate is issued to each principal only once. The certificate is kept by the principal and is forwarded during an authentication exchange, thus eliminating the need to query A interactively. Forgery is impossible since a certificate is signed by the certification authority.

3.3 Notion of Trust

It is easy to see that correctness of both the symmetric and asymmetric protocols presented above requires more than the existence of secure communication channels between principals and the appropriate authentication servers (or certification authorities). In fact, such correctness is critically dependent on the ability of the servers (authorities) to faithfully follow the protocols. Each principal bases its judgment on its own observations (messages sent and received) and its trust on the server's judgment.

In some sense, the trust required of a certification authority is less than the trust required of an authentication server, as all information (except its own private key) kept by the authority is public.

Furthermore, a certification authority has no way of masquerading as a principal since the private key of a principal is never shared.

Our formal understanding of trust in a distributed system is at best inadequate. In particular, a formal understanding of authentication would require both a formal specification of trust and a rigorous reasoning method wherein trust is one of the basic elements.

4 Authentication Protocol Failures

Despite the apparent simplicity of the basic design principles for authentication protocols, realistic authentication protocols are notoriously difficult to design. Several protocols were published and later found to exhibit subtle security problems [3, 4, 8, 19].

There are several reasons for this difficulty. First, most realistic cryptosystems satisfy additional algebraic identities other than those in (C1) and (C2) (see Box). These extra properties may generate undesirable effects when combined with the logic used in a protocol [18]. Second, even assuming that the underlying cryptosystem is perfect, unexpected interaction among the protocol steps themselves can lead to subtle logical flaws. Third, assumptions regarding the environment and the capabilities of an adversary are never explicitly specified, thus rendering it extremely difficult to determine when a protocol is applicable and what final states are achieved.

We illustrate the difficulty by showing below an authentication protocol proposed in [19] that was found to contain a subtle weakness [4, 8]: k_P and k_Q are symmetric keys shared between P and A, and Q and A, respectively, where A is an authentication server. Let k be a session key.

$$\begin{array}{lll}
(1) & P \rightarrow A: & P, Q, n_P \\
(2) & A \rightarrow P: & \{n_P, Q, k, \{k, P\}_{k_Q}\}_{k_P} \\
(3) & P \rightarrow Q: & \{k, P\}_{k_Q} \\
(4) & Q \rightarrow P: & \{n_Q\}_k \\
(5) & P \rightarrow Q: & \{n_Q + 1\}_k
\end{array}$$

The message $\{k, P\}_{k_Q}$ in step (3) can only be decrypted and hence understood by Q. Step (4) reflects Q's knowledge of k while step (5) assures Q of P's knowledge of k; hence the authentication handshake is based entirely on knowledge of k. A subtle weakness of the protocol arises from the fact that the message $\{k, P\}_{k_Q}$ sent in step (3) contains no information for Q to verify its freshness.[1] In fact, this is the first message sent to Q notifying it of P's intention to establish a secure connection. An adversary who has compromised an old session key k' can impersonate P by replaying the recorded message $\{k', P\}_{k_Q}$ in step (3), and subsequently executing the steps (4) and (5) using k'.

To avoid protocol failures, formal methods may be employed in the design and verification of authentication protocols. A formal design method should embody the basic design principles as illustrated in the previous section while informal reasoning such as:

> "If you believe that only you and Bob know k, then any message you receive encrypted with k should be believed to have been sent by Bob originally."

should be formalized within a verification method.

Early attempts at formal verification of security protocols followed mainly an algebraic approach [9, 10]. Specifically, messages exchanged in the protocol are viewed as terms in an algebra and various identities involving the encryption and decryption operators (e.g. (C1), (C2)) were taken to be term rewriting rules.

[1]Note that k is known to be fresh only by P and A.

A protocol is *secure* if it is impossible to derive certain terms (e.g. the term containing the key) from the terms obtainable by an adversary. The algebraic approach is limited since it deals only with one aspect of security, namely secrecy. Recently, various logical approaches have been proposed to study authentication protocols [3, 2] Most of these logics adopt a modal basis, with *belief* and *knowledge* being their central notions. The logical approaches appear to be more general, but they currently lack a rigorous foundation as compared to the more well-established logics e.g. first-order logic and temporal logic. In particular, a satisfactory semantic model for these logical systems has not been developed. Clearly, much research is still needed to obtain sound design methods and to formally understand issues of authentication.

5 An Authentication Framework

We have so far presented various basic concepts of authentication. In this section, we synthesize these concepts into a specific authentication framework that can be incorporated into the design of secure distributed systems.

In particular, we identify five aspects of secure distributed system design and the associated authentication needs. For these five particular aspects, we demonstrate how authentication protocols can be used to address specific needs. This section should not be regarded as exhaustive in scope, because in an actual distributed system security framework, other issues may have to be addressed as well. The five aspects are described below:

- Host initializations — all process executions take place inside hosts. Some hosts (e.g. workstations) also act as entry points to the system by allowing user logins. The overall security of a distributed system is highly dependent on the security of each of the hosts. However, in an open network environment, not all hosts can be physically protected. Thus resistance to compromise must be built into a host's software to ensure its secure operation. This suggests the importance of the integrity of host software. In particular, for a host that employs remote initialization, loading it with the correct host software is necessary for its proper functioning. In fact, one way to compromise a public host is to reboot the host with incorrect initialization information. Authentication can be used to implement *secure bootstrapping.*

- User logins — login is the point where a user initiates its activities within the system. The identity of the user is established at this point and all subsequent activities of the user are attributed to this established identity. In particular, all access control decisions and accounting functions would be based on this established identity. Therefore correct identification of users is crucial to the functioning of a secure system. On the other hand, any host in an open environment is susceptible to compromise, thus a user should not engage in any activity with a host without first ascertaining the host's identity. A mutual user-host authentication can be used to achieve the required guarantees.

- Peer communications — an advantage of distributed systems over a centralized one is the ability to distribute a task over multiple hosts so as to achieve a higher throughput or more balanced utilizations. Correctness of such a distributed task depends on the ability of peer processes participating in the task to correctly identify each other. Authentication can be used here to identify friend or foe.

- Client-server interactions — the client-server model provides an attractive paradigm for constructing distributed systems. Servers are only willing to provide service to authorized clients while clients are interested in dealing with the legitimate servers only. Authentication can be used to verify a potential consumer-supplier relationship.

- Inter-domain communication — most distributed systems are not centrally owned or administered; for example, a campus-wide distributed system is often an interconnection of individually administered departmental subsystems. Additional authentication mechanisms are required for the identification of principals across subsystems.

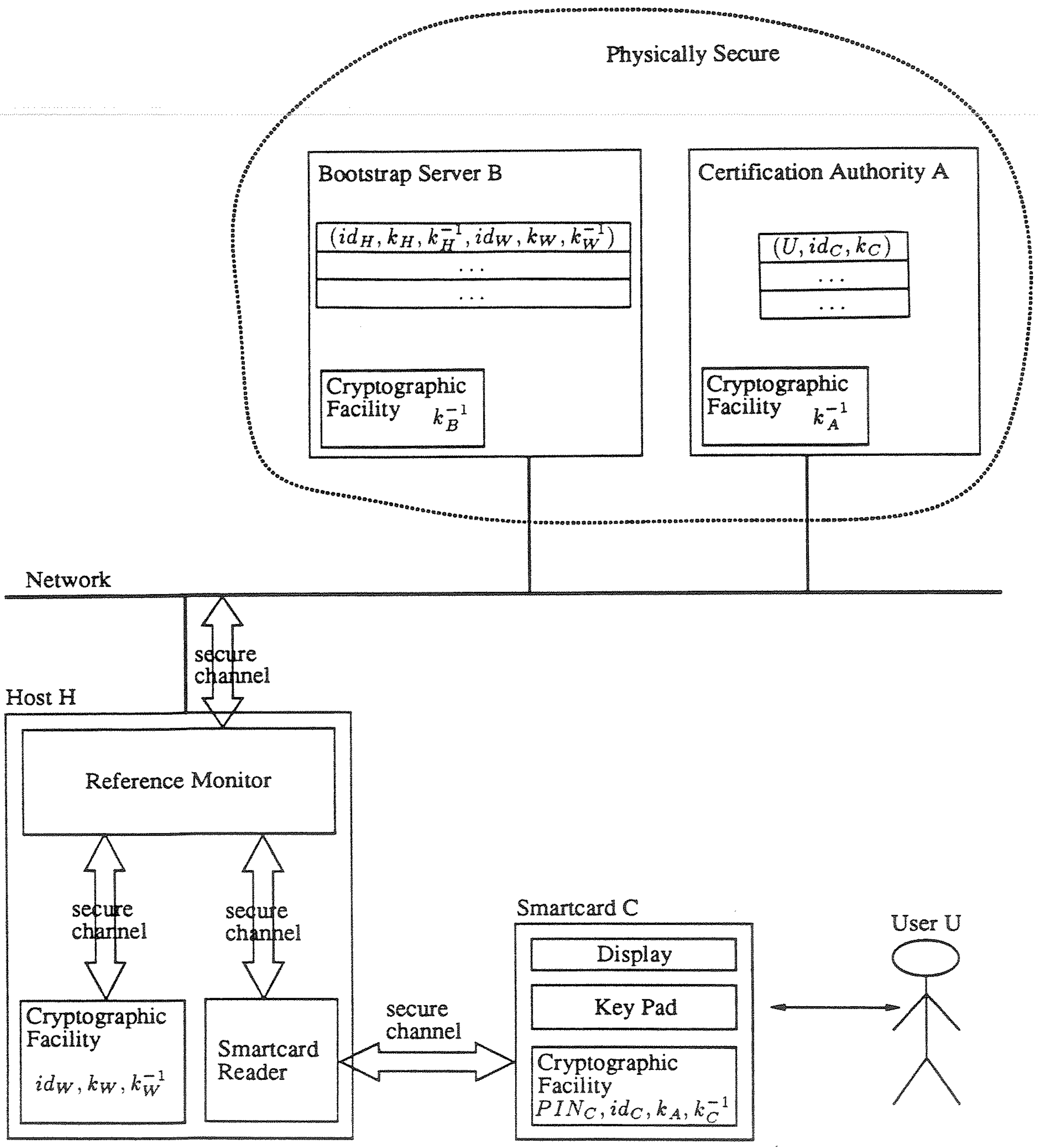
Physically Secure
Bootstrap Server B
$(id_H, k_H, k_H^{-1}, id_W, k_W, k_W^{-1})$
...
...
Cryptographic Facility k_B^{-1}
Certification Authority A
(U, id_C, k_C)
...
...
Cryptographic Facility k_A^{-1}
Network
secure channel
Host H
Reference Monitor
secure channel
secure channel
Cryptographic Facility id_W, k_W, k_W^{-1}
Smartcard Reader
secure channel
Smartcard C
Display
Key Pad
Cryptographic Facility $PIN_C, id_C, k_A, k_C^{-1}$
User U

Figure 2: Distributed System Configuration

In the kind of malicious environments postulated in our threats model, some basic assumptions about the system must be satisfied to achieve any reasonable level of security. We list a set of assumptions below (for other possible assumptions, see [1, 15]). These assumptions are also depicted in Figure 2.

- Each host hardware W has a unique built-in immutable identity id_W, and contains a tamper-proof cryptographic facility that encapsulates the public key k_W and the private key k_W^{-1} of W. The cryptographic facility can communicate with the host reference monitor via a secure channel. Each host that supports user logins also has a smartcard reader that can communicate with the host reference monitor via a secure channel. Lastly, the host reference monitor has a secure channel to the network interface.

- Each legitimate user U is issued a smartcard C that has a unique built-in immutable identity id_C. Each smartcard C is capable of performing encryption and decryption, and encapsulates its private key k_C^{-1}, the public key for the certification authority k_A (see below) and a pin number PIN_C for its legitimate holder. (The pin number is assigned by a card issuing procedure.) The channel between the smartcard and the smartcard reader is secure. Each smartcard has its own display, its own keypad, and a clock.

- A physically-secure centralized bootstrap server B exists that maintains a database of all host information. More precisely, for each host H, it keeps a record $(id_H, k_H, k_H^{-1}, id_W, k_W, k_W^{-1})$ specifying the unique hardware W that can be initialized as H. All records in the database can be encrypted under a secret master key for added security. B has a public key k_B and a private key k_B^{-1}.

- A physically-secure centralized certification authority A exists that maintains a database of information on all principals. More precisely, for each user U, A keeps a record (U, id_C, k_C), binding U to its smartcard C. For each host H, A keeps a record (id_H, id_W), specifying the hardware W that H is supposed to run on. Also, for each server S, A keeps a record of its public key certificate (S, k_S). The certification authority A has a public key k_A and a private key k_A^{-1}.

Each of the above assumptions is achievable with current technology. In particular, the technology of battery-powered credit-card-sized smartcard with a built-in LCD display, keypad, and capable of performing specialized computations has had steady progress in recent years. Also, specialized cryptographic facilities and smartcard readers for hosts are already included as options from many vendors. The use of a smartcard or other forms of computation aid is essential in realizing mutual authentication between a host and a user, as unaided human users simply cannot carry out the intensive computations required by an authentication protocol.

The bootstrap server and the certification authority are assumed to be centralized to simplify our presentation. Decentralized servers/authorities can be supported by adding authentication between the servers/authorities themselves (see Section 5.5). Such authentication can be carried out in a hierarchical manner as suggested in X.509 [7].

While there is a certification authority in our authentication framework, there is no authentication server. We made this choice because the level of trust needed in a certification authority (which distributes public key certificates) is deemed to be less than that of an authentication server (as discussed in Section 3.3).

In the following subsections, we present some specific protocols designed to address authentication needs in our framework. The protocols presented are not meant to be definitive or optimal (i.e. least number of messages or the weakest initial assumptions). They serve to illustrate possible solution approaches.

5.1 Secure Bootstrapping

The following secure bootstrapping protocol is initiated when some host hardware attempts a remote initialization. This could take place after a voluntary shutdown, a system crash, or a malicious attack by an adversary who attempts to penetrate the host. The secure bootstrap protocol allows a reinitialized host to attain a "safe" state prior to resuming normal operation. In particular, a correctly loaded reference monitor is ready to assume control of the host in this state.

Suppose that the hosts and the bootstrap server B are on the same broadcast network, hence allowing the message in step (1) below to be received by B. In the following protocol, n_W and n_B are nonces, k is a session key, OS is a copy of the operating system, and T is a timestamp.

(1)	$W \rightarrow$ all	:	"boot", id_W, $\{n_W, id_W\}_{k_W}$
(2)	B	:	retrieve record $(id_H, k_H, k_H^{-1}, id_W, k_W, k_W^{-1})$ for W from database
		:	recover n_W from $\{n_W, id_W\}_{k_W}$ by decrypting with k_W^{-1}
		:	generate a random key k
		:	compute $m = \{n_W, k_A, k_B, k\}_{k_W}$
(3)	$B \rightarrow W$	:	m
(4)	W	:	recover (n_W, k_A, k_B, k) from m by decrypting with k_W^{-1}
(5)	$W \rightarrow B$	:	$\{n_W, \text{"ready"}\}_k$
(6)	$B \rightarrow W$	:	$\{n_W, n_B, id_H, \{k_H^{-1}\}_{k_W}, OS\}_k$
(7)	$W \rightarrow B$	:	$\{\{n_B\}_{k_H^{-1}}\}_k$
(8)	$B \rightarrow W$	:	$\{id_H, id_W, k_H, T\}_{k_B^{-1}}$
(9)	H	:	validate certificate $\{id_H, id_W, k_H, T\}_{k_B^{-1}}$ by encrypting with k_B

The basic idea of the protocol is as follows: In step (1), W announces its intention to reboot by broadcasting a boot request. Only B who has knowledge of k_W^{-1} can recover the nonce n_W. In step (2), B generates a fresh key k to be used for loading OS. In step (4), W ascertains that m is not a replay by checking the component n_W, since only B could have composed message m. Thus k_A, k_B and k in m can be safely taken to be respectively the public key of certification authority A, the public key of B and the session key to be used for loading OS. At this point, B has been authenticated by W.

When the message "ready" encrypted with k is received in step (5), B is certain that the original boot request actually came from W since only W can decrypt m to retrieve k. Hence, B and W have mutually authenticated each other.

Step (6) is the actual loading of OS and transfer of the private key k_H^{-1} of host H. The checksum of OS is included as part of OS, and it should be recomputed by W to detect any tampering of OS in transit. W acknowledges the receipt of k_H^{-1} and OS by returning the nonce n_B signed with k_H^{-1}. B verifies that the correct n_B is returned. Then in step (8), a *license* signed by B affirming the binding of host id_H with public key k_H and hardware id_W is sent to H. This license is retained by H as a proof of successful bootstrapping and of its identity. The timestamp field T within the license denotes its expiration date.

If secrecy of OS is not required, OS can be transferred without being encrypted first. However, the checksum of OS must be sent in encrypted form.

5.2 User-Host Authentication

User-host authentication is performed when a user U walks up to a host H and attempts to log in. The authentication requires the use of a smartcard C. A successful authentication provides a guarantee to host H that U is the legitimate holder of C and a guarantee to user U that H is a "safe" host to use, i.e., the host holds a valid license (which may have been obtained through secure bootstrapping), and possesses knowledge of the private key k_H^{-1}.

In most systems, the end result of a successful user authentication is the creation of a login process by the host's reference monitor on behalf of the user. The login process is a proxy for the user, and all requests generated by the login process are taken as if they are directly made by the user. However, a remote host/server has no way of confirming such proxy status, except to trust the authentication capability and integrity of the local host. Such trust is unacceptable in a potentially malicious environment since a compromised host can simply claim the existence of user login processes to obtain unauthorized services.

A solution to alleviate this trust requirement is to have a user explicitly *delegate* its authority to its login host [1, 15]. The delegation can be done by having the user's smartcard sign a *login certificate* to the login host upon the successful termination of a user-host authentication protocol. The login certificate asserts the proxy status of the host with respect to the user, and can be presented by the host in future authentication exchanges.

Because of the possibility of forgery, the possession of a login certificate should not be taken as sufficient proof of delegation. The host is also required to demonstrate the knowledge of a private delegation key k_d^{-1} whose public component k_d is named in the certificate. Also, to reduce the potential impact of a host compromise, the login certificate is given only a finite lifetime by including in it an expiration timestamp.

We present such a user-host authentication protocol below: we assume that the host holds a valid license $\{id_H, id_W, k_H, T\}_{k_B^{-1}}$ as would be the case if the host has executed the secure bootstrapping protocol. In the following, n_C is a nonce, k_d is the public delegation key, whose private counterpart k_d^{-1} is kept secret by the host, T_c is a timestamp denoting the expiration date of the login certificate.

(1)	$C \rightarrow H$	:	$id_C,\ n_C$
(2)	$H \rightarrow A$	:	$id_C, \{id_H, id_W, k_H, T\}_{k_B^{-1}}$
(3)	A	:	check timestamp of certificate
		:	if timestamp expired, abort
(4)	$A \rightarrow H$	:	$\{id_H, id_W, k_H, T\}_{k_A^{-1}}, \{U, id_C, k_C\}_{k_A^{-1}}$
(5)	H	:	generate new delegation key pair (k_d, k_d^{-1})
(6)	$H \rightarrow C$	:	$\{id_H, id_W, k_H, T\}_{k_A^{-1}}, \{U, k_d, n_C\}_{k_H^{-1}}$
(7)	$C \rightarrow U$	:	id_H, id_W
(8)	U	:	verify if id_H/id_W is the host desired
		:	if not, abort
(9)	$U \rightarrow C$	:	PIN
(10)	C	:	verify $PIN \overset{?}{=} PIN_C$
		:	if not equal, abort
(11)	$C \rightarrow H$	:	$\{U, id_H, k_d, T_c\}_{k_C^{-1}}$
(12)	H	:	verify correct delegation by decrypting the login certificate $\{U, id_H, k_d, T_c\}_{k_C^{-1}}$ with k_C

The protocol proceeds as follows: A user walks up to a host and inserts his/her smartcard in the card reader. The card's identity id_C and a nonce n_C are sent through the card reader to the host in step (1). In step (2) H requests user information associated with id_C from certification authority A. Since the license held by H was signed by B and hence not decipherable by C, a key translation is requested by H in the same step. (Note that these licenses can be cached by H and need not be requested for every user authentication.) After receiving a reply from A in step (4), H knows both the legitimate holder U of the card C and the public key k_C associated with the smartcard. Knowledge of U can be used to enforce local discretionary control to provide service (or not), while k_C is needed to verify the authenticity of C. In step (5), H generates a new delegation key pair (k_d, k_d^{-1}). H keeps k_d^{-1} private, to be used as proof of a successful delegation from U to H.

In step (6), H returns the nonce n_C with the public delegation key k_d, and a copy of its license to C. In step (7), C retrieves (id_H, id_W), the identity of H, from the license by decrypting it with k_A. A check is made to ensure that the timestamp in the license has not expired. The identity (id_H, id_W) is then displayed on the card's own screen. If the user decides to proceed, he/she enters on the card's keypad the pin number assigned to him in the card's issuing procedure. In step (10), the pin number entered is verified against the one stored in the card. If they are equal, C signs a login certificate binding the user U with the host id_H and the public delegation key k_d. This is sent to H in step (11). The host (and others) can verify the validity of the login certificate using k_C, the card's public key.

When user U logs out, the host erases its copy of the private delegation key k_d^{-1} to void the delegation from U. In the case that H is compromised after the delegation, the effect of the login certificate is limited by its lifetime, T_c.

5.3 Peer-Peer Authentication

Peer-peer mutual authentication and cryptographic parameters negotiation (e.g. session key agreement) are performed in the connection establishment phase of a secure connection-oriented protocol.

The following protocol mutually authenticates peers P and Q, and establishes a new session key for their future communication. Below, n_P and n_Q are nonces, while k is a fresh session key.

$$\begin{array}{lll}
(1) & P \rightarrow A: & P, Q \\
(2) & A \rightarrow P: & \{Q, k_Q\}_{k_A^{-1}} \\
(3) & P \rightarrow Q: & \{n_P, P\}_{k_Q} \\
(4) & Q \rightarrow A: & Q, P, \{n_P\}_{k_A} \\
(5) & A \rightarrow Q: & \{P, k_P\}_{k_A^{-1}}, \{\{n_P, k, Q\}_{k_A^{-1}}\}_{k_Q} \\
(6) & Q \rightarrow P: & \{\{n_P, k, Q\}_{k_A^{-1}}, n_Q\}_{k_P} \\
(7) & P \rightarrow Q: & \{n_Q\}_k
\end{array}$$

In step (1), P informs A of its intention to establish a secure connection with Q. In step (2), A returns to P a copy of Q's public key certificate. In step (3), P informs Q of its desire to communicate along with a nonce n_P. In step (4), Q asks A for P's public key certificate and requests a session key at the same time. In order for Q to subsequently prove to P that the session key k is actually from A (not Q's own creation), A sends a signed statement containing the key k, n_P and Q's name. This basically says that k is a key generated by A on behalf of Q's request identified by n_P. The binding of k and n_P assures P that k is fresh. In step (6), A's signed copy of (n_P, k, Q) is relayed to P together with a nonce n_Q generated by Q. P's knowledge of the new session key k is indicated to Q by the receipt of n_Q in step (7).

5.4 Client-Server Authentication

Since both clients and servers are implemented as processes, the basic protocol for peer-peer authentication in Section 5.3 can be applied here as well. However, several issues peculiar to client-server interactions need to be addressed. They are described next.

In a general-purpose distributed computing environment, new services (hence servers) are made available dynamically. Thus instead of informing clients of every service available, most implementations use a *service broker* to keep track of and direct clients to appropriate service providers. Thus a client would first contact the service broker using a *purchase protocol* which performs the necessary mutual authentication prior to the granting of a *ticket*. The ticket is later used by the client to redeem services from the actual server using a *redemption protocol.*

Authentication performed by the purchase protocol proceeds as the protocol for peer to peer authentication, while in the redemption protocol authentication is based upon possession of a ticket and some information recorded in the ticket. Such a ticket contains the names of the client and the server, a key and a timestamp to indicate lifetime (similar to a login certificate). A ticket can only be used between the specified client and server. A prime example of this approach is the Kerberos authentication system discussed in Subsection 6.1.

Another special issue of client-server authentication is *proxy authentication* [13, 14, 22]. It is quite common that in the course of satisfying a client's request, a server needs to access other servers on behalf of the client. For example, a database server, upon accepting a query from a client, may need to access the file server to retrieve certain information on the client's behalf. A straightforward solution would be to require the file server to directly authenticate the client. However, this may not be feasible in general; for instance, in a long chain of service requests, the client may not be aware of some request made by one of the servers in the chain, and hence is not in a position to perform the required authentication. An alternative solution is to extend the concept of *delegation* previously used in user-host authentication [13]. Specifically, a client can forward a signed *delegation certificate* affirming the delegation of its rights to a server along with its service request. The server is allowed to delegate to another server by signing its own delegation certificate as well as relaying the client's certificate. In general, for a service request involving a sequence of servers, delegation can be propagated to the final server in the sequence through intermediate servers, forming a *delegation chain.*

Various refinements are possible to extend the delegation scheme described. For example, *restricted delegation* can be accomplished by explicitly specifying a set of rights and/or objects in a delegation certificate.

5.5 Inter-domain Authentication

Up to now, we have assumed a centralized certification authority that is trusted by all principals. However, a realistic distributed system is often composed of subsystems that are independently administered by different authorities. We use *domain* to refer to such an independently administered subsystem. Each domain D maintains its own certification authority A_D that has jurisdiction over all principals within the domain. *Intra-domain* authentication refers to an authentication exchange between two principals belonging to the same domain, whereas *inter-domain* authentication refers to an authentication exchange that involves two principals belonging to different domains.

Using the previously described protocols, A_D is sufficient for all intra-domain authentications for each domain D. However, a certification authority has no way of verifying a request from a remote principal, even if the request is certified by a remote certification authority. Hence, additional mechanisms are required in the case of inter-domain authentication,

To allow inter-domain authentication, two issues need to be addressed: *naming* and *trust.* Naming is concerned with ensuring that principals are uniquely identifiable across domains, so that each authentication request can be attributed to a unique principal. A global naming system spanning all domains can be used to provide globally unique names to all principals. A good example of this is the Domain Name System used in Internet.

Trust refers to the willingness of a local certification authority to accept a certification made by a remote certification authority regarding a remote principal. Such trust relationships must be explicitly established between domains, which can be achieved in several ways:

- by sharing an inter-domain key between certification authorities that are willing to trust each other,
- by installing the public keys of all trusted remote certification authorities in a local certification authority's database, and
- by introducing an inter-domain certification authority for authenticating domain-level certification authorities. In general a hierarchical organization corresponding to that of the naming system can be imposed on the certification authorities. In this case, an authentication exchange between two principals P and Q involves multiple certification authorities on a path in the hierarchical organization between P and Q [5, 12]. The path is referred to as a *certification path.*

6 Case Studies

We study two authentication services, namely: Kerberos and SPX. Both were designed to address primarily client-server authentication needs. Their services are generally available to an application program through a programming interface. While Kerberos uses a symmetric cryptosystem, SPX is based on the use of both symmetric and asymmetric cryptosystems.

6.1 Kerberos

Kerberos is an authentication system designed for use with MIT's Project Athena [21]. The goal of Project Athena is to create an educational computing environment based on high-performance workstations, high-speed networking, and servers of various types. A large-scale (10,000 workstations/1000 servers) open network computing environment is envisioned where individual workstations can be privately owned and operated. Therefore, a workstation cannot be trusted to identify its users correctly to network services. Kerberos is not a complete authentication framework required for secure distributed computing in general; specifically, it addresses only issues of client-server interactions.

We will limit our discussion to the Kerberos authentication protocols and omit various administrative issues concerning the use of Kerberos.

Kerberos's design is based on the use of a symmetric cryptosystem together with trusted third-party authentication servers; it is a refinement of ideas presented in [19]. The basic components of Kerberos include authentication servers (*Kerberos servers*) and *ticket-granting servers* (TGSs). A database is maintained that contains information on each principal; in particular, it stores a copy of each principal's key shared with Kerberos. For a user principal U, its shared key k_U is computed from its password $password_U$; specifically $k_U = f(password_U)$ for some one-way function f. The database is read by Kerberos servers and TGSs in the course of authentication.

There are two main protocols in Kerberos. One is used in authenticating a user login and installing an initial ticket at the login host. We refer to this as the *credential initialization protocol*. The other is a client-server authentication protocol, and is used when a client requests services from a server. We describe both protocols below.

Kerberos servers are used in the credential initialization protocol. Let U be a user who attempts to log in host H, and f be the one-way function for computing k_U from U's password. The protocol can be specified as follows:[2]

$U \rightarrow H$	:	U
$H \rightarrow$ Kerberos	:	U, TGS
Kerberos	:	retrieve k_U and k_{TGS} from database
	:	generate a new session key k
	:	create ticket-granting ticket $tick_{TGS} = \{U, TGS, k, T, L\}_{k_{TGS}}$
Kerberos $\rightarrow H$	:	$\{TGS, k, T, L, tick_{TGS}\}_{k_U}$
$H \rightarrow U$	:	"Password?"
$U \rightarrow H$	:	$passwd$
H	:	compute $\ell = f(passwd)$
	:	recover $k, tick_{TGS}$ by decrypting $\{TGS, k, T, L, tick_{TGS}\}_{k_U}$ with ℓ
	:	if decryption fails, abort login
	:	otherwise retain $tick_{TGS}$ and k
	:	erase $passwd$ from memory

[2]Kerberos in the following protocol refers to a Kerberos server.

If *passwd* is not the valid password of U, ℓ would not be identical to k_U and decryption in the last step would fail.[3] Upon successful authentication, the host obtains a new session key k and a copy of the *ticket-granting* ticket $tick_{TGS} = \{U, TGS, k, T, L\}_{k_{TGS}}$ where T is a timestamp, L is the ticket's lifetime. The ticket-granting ticket is used to request server tickets from a TGS; note that $tick_{TGS}$ is encrypted with k_{TGS}, the shared key between TGS and Kerberos.

A ticket by itself does not constitute sufficient proof of identity, since it is susceptible to interception or copying. Therefore a principal, when presenting a ticket, is also required to demonstrate knowledge of the session key k named in the ticket. An *authenticator* is used to provide such a demonstration (see below). The protocol for a client C to request network service from a server S is as follows: T_1 and T_2 are timestamps.

(1)	$C \rightarrow$ TGS	:	$S, tick_{TGS}, \{C, T_1\}_k$
(2)	TGS	:	recover k from $tick_{TGS}$ by decrypting with k_{TGS}
		:	recover T_1 from $\{C, T_1\}_k$ by decrypting with k
		:	check timeliness of T_1 with respect to local clock
		:	create server ticket $tick_S = \{C, S, k', T', L'\}_{k_S}$
(3)	TGS $\rightarrow C$	:	$\{S, k', T', L', tick_S\}_k$
(4)	C	:	recover $k', tick_S$ by decrypting with k
(5)	$C \rightarrow S$	:	$tick_S$,$\{C, T_2\}_{k'}$
(6)	S	:	recover k' from $tick_S$ by decrypting with k_S
		:	recover T_2 from $\{C, T_2\}_{k'}$ by decrypting with k'
		:	check timeliness of T_2 with respect to local clock
(7)	$S \rightarrow C$	:	$\{T_2 + 1\}_{k'}$

In step (1), client C presents its ticket-granting ticket $tick_{TGS}$ to TGS to request a ticket for server S.[4] C's knowledge of k is demonstrated using the authenticator $\{C, T_1\}_k$. In step (2), TGS decrypts $tick_{TGS}$, recovers k and uses it to verify the authenticator. If both decryptions in step (2) are successful and T_1 is timely, TGS creates a ticket $tick_S$ for server S and returns it to C. Holding $tick_S$, C repeats the authentication sequence with S. Thus, in step (5), C presents S with $tick_S$ and a new authenticator. In step (6), S performs verifications similar to those by TGS in step (2). Finally, step (7) assures C of the server's identity. Note that this protocol requires "loosely synchronized" local clocks for the verification of timestamps.

Kerberos can also be used for authentication across administrative/organizational domains. Each administrative/organizational domain is called a *realm*. Each user belongs to some realm, one that is identified by a field in the user's id. Services registered in a realm will only accept tickets issued by an authentication server for that realm.

To support cross-realm authentication, an *inter-realm key* is shared between two realms. The TGS of one realm can be registered as a principal in another realm by using the shared inter-realm key. Thus a user can obtain a ticket-granting ticket for contacting a remote TGS from its local TGS. When the ticket-granting ticket is presented to the remote TGS, it can be decrypted by the remote TGS using the appropriate inter-realm key to ascertain that it was issued by the user's local TGS. In general, an *authentication path* spanning multiple intermediate realms is possible.

Kerberos is still an evolving system. The latest version being Version V5 [16]. Various limitations of previous versions of Kerberos were discussed in [6], and some, but not all, have been remedied.

[3] In practice, f may not be 1-1. It suffices to require that given two distinct elements x and y, the probability of $f(x)$ being equal to $f(y)$ is negligible.

[4] Note that each client process is associated with a unique user (the user who created the process). It inherits the id of the user and the ticket-granting ticket issued to the user during login.

6.2 SPX

SPX is another authentication service intended for open network environments [23]. It is a major component of the Digital Distributed System Security Architecture [12]. SPX offers functionalities similar to those of Kerberos; specifically, SPX also has a credential initialization protocol and a client-server authentication protocol. In addition, SPX has an *enrollment protocol* that is used to register new principals. In this subsection, we focus only on the first two protocols and will omit the enrollment protocol and most other administrative issues.

Corresponding to Kerberos servers and TGSs, SPX has a Login Enrollment Agent Facility (LEAF) and a Certificate Distribution Center (CDC). LEAF is similar to a Kerberos server, and is used in the credential initialization protocol. CDC is an on-line depository of public key certificates (for principals and certification authorities) as well as encrypted private keys of principals. Note that CDC needs not be trustworthy as everything stored in it is encrypted and can be verified independently by principals.

Besides LEAF and CDC, there are also certification authorities (CAs) in SPX. The CAs are organized hierarchically. All CAs operate off-line and are selectively trusted by principals. Their function is to issue public key certificates (binding names and public keys of principals). Global trust is not needed in SPX. Typically, each CA has jurisdiction over just a subset of all principals, while each principal P trusts only a subset of all CAs, referred to as the *trusted authorities of* P. Scalability of the system is greatly enhanced by the absence of global trust and on-line trusted components.

The credential initialization protocol is performed when a user logs in. It installs a ticket and a set of trusted-authority certificates for the user upon successful login. We present the protocol below: U is a user who attempts to log in host H, *passwd* is the password entered by U, T is a timestamp, L is the lifetime of a ticket, n is a nonce, h_1 and h_2 are publicly-known one-way functions, k is a (DES) session key, k_U, k_{LEAF}, k_A are respectively the public keys of U, the LEAF server, and a trusted authority A of U, and k_U^{-1} and k_{LEAF}^{-1} are respectively the private keys of U and LEAF:

(1)	$U \rightarrow H$	:	$U, passwd$
(2)	$H \rightarrow \mathrm{LEAF}$	:	$U, \{T, n, h_1(passwd)\}_{k_{\mathrm{LEAF}}}$
(3)	$\mathrm{LEAF} \rightarrow \mathrm{CDC}$	:	U
(4)	$\mathrm{CDC} \rightarrow \mathrm{LEAF}$	:	$\{\{k_U^{-1}\}_{h_2(password_U)}, h_1(password_U)\}_k, \{k\}_{k_{\mathrm{LEAF}}}$
(5)	LEAF	:	recover k by decrypting with k_{LEAF}^{-1}
		:	recover $\{k_U^{-1}\}_{h_2(password_U)}$ and $h_1(password_U)$ by decrypting with k
		:	verify $h_1(passwd) \stackrel{?}{=} h_1(password_U)$
		:	if not equal, abort
(6)	$\mathrm{LEAF} \rightarrow H$	:	$\{\{k_U^{-1}\}_{h_2(password_U)}\}_n$
(7)	H	:	recover k_U^{-1} by decrypting first with n and then with $h_2(passwd)$
		:	generate (RSA) delegation key pair (k_d, k_d^{-1})
		:	create ticket $tick_U = \{L, U, k_d\}_{k_U^{-1}}$
(8)	$H \rightarrow \mathrm{CDC}$	:	U
(9)	$\mathrm{CDC} \rightarrow H$	:	$\{A, k_A\}_{k_U^{-1}}$

In step (1), user U enters its id and password. In step (2), H applies the one-way function h_1 to the password entered by U and sends the result along with a timestamp T and a nonce n in a message to LEAF. LEAF, on receiving the message from H, forwards a request to CDC for U's private key. The private key of U is stored as a record $(\{k_U^{-1}\}_{h_2(password_U)}, h_1(password_U))$ in CDC; note that compromise of CDC would not reveal these private keys. In step (4), CDC sends the requested private-key record to LEAF using a temporary session key k. In step (5), LEAF recovers both $\{k_U^{-1}\}_{h_2(password_U)}$ and $h_1(password_U)$ from

CDC's reply. LEAF then verifies *passwd* by checking $h_1(passwd)$ against $h_1(password_U)$; if they are not equal, the login session is aborted and the abortion logged. Note that $h_1(password_U)$ is never revealed to any principal except LEAF, thus password guessing attacks would require contacting LEAF for each guess or compromising LEAF's private key.

Having determined the password to be valid, LEAF sends the first part of the private-key record encrypted by n to H in step (6). (The nonce n sent in step (2) is used as a symmetric key for encryption.) In step (7), H recovers k_U^{-1} by decrypting the reply from LEAF first with n and then with $h_2(passwd)$. H then generates a pair of delegation keys and create a ticket $tick_U$. In step (8), H requests the public key certificate for a trusted authority of U from CDC. CDC replies with the certificate in step (9). In fact, multiple certificates can be returned in step (9) if U trusts more than one CA. These trusted authorities' certificates were previously deposited in the CDC by U using the enrollment protocol.

The authentication exchange protocol between a client C and a server S is described in the following. To simplify the protocol specification such that a single public key certificate is sent in step (2) and also in step (5), we made the following assumption: Let the public key certificate of C be signed by A_C such that A_C is one of the trusted authorities of S. Similarly, let the public key certificate of S be signed by A_S such that A_S is one of the trusted authorities of C. Below, T is a timestamp, and k is a (DES) session key:

(1)	$C \rightarrow \text{CDC}$	:	S
(2)	$\text{CDC} \rightarrow C$	:	$\{S, k_S\}_{k_{A_S}^{-1}}$
(3)	$C \rightarrow S$	:	$T, \{k\}_{k_S}, tick_C, \{k_d^{-1}\}_k$
(4)	$S \rightarrow \text{CDC}$	:	C
(5)	$\text{CDC} \rightarrow S$	:	$\{C, k_C\}_{k_{A_C}^{-1}}$
(7)	S	:	validate $tick_C$ by encrypting with k_C
(6)	$S \rightarrow C$	:	$\{T+1\}_k$

In step (1), C requests the public key certificate of S from CDC. In step (2), CDC returns the requested certificate. C can verify the public key certificate by decrypting it with k_{A_S}, which is the public key of A_S obtained by C when it executed the credential intialization protocol. In step (3), $tick_C$ and the private delegation key k_d^{-1} generated in step (7) of the credential initialization protocol, as well as a new session key k are sent to S. Only S can recover k from $\{k\}_{k_S}$ and subsequently decrypt $\{k_d^{-1}\}_k$ to recover k_d^{-1}. Possession of $tick_C$ and knowledge of the private delegation key constitute sufficient proof of delegation from C to S. However, if such delegation from C to S is not needed, $\{\{k\}_{k_S}\}_{k_d^{-1}}$ is sent in step (3) instead of $\{k_d^{-1}\}_k$; this acts as an authenticator for proving C's knowledge of k_d^{-1} without revealing it. In steps (4) and (5), S requests the public key certificate of C, which is used to verify $tick_C$ in step (7). Finally, S returns $\{T+1\}_k$ to C and mutual authentication between C and S is completed.

SPX is a relatively recent proposal. Hence, no extensive study of its security properties has been done. It is necessary that such a detailed study be performed before its general adoption.

Although SPX offers services similar to those of Kerberos, its elimination of on-line trusted authentication servers and the extensive use of hierarchical trust relationships are intended to make SPX scalable for very large distributed systems.

7 Conclusion

With the growth in scale of distributed systems, security has become one of the major concerns and limiting factors in their design. For example, security has been strongly advocated as one of the major design constraints in both the Athena [21] and Andrew [20] projects. Most existing distributed systems, however, have been designed without a well-defined security framework, and their use of authentication is nonexistent or limited to only the most critical applications.

Various authentication needs for distributed systems have been described in this paper, and some specific protocols are presented. Most of them are practically feasible in today's technology and their adoption and use should be just a matter of need.

To cope with the complexity of understanding and managing security, a formal approach should be used. A formal approach allows the precise specification of a security framework and rigorous analysis. A basis for developing such an approach can be found in [17].

Acknowledgements We thank Clifford Neuman (University of Washington) and John Kohl (MIT) who reviewed the section on Kerberos. We are also grateful to the anonymous referees for their constructive comments.

References

[1] M. Abadi, M. Burrows, C. Kaufman and B.W. Lampson, "Authentication and Delegation with Smart-cards," *Techincal Report 67*, System Research Center, Digital Corporation, October 22, 1990.

[2] P. Bieber, "A Logic of Communication in Hostile Environment," *Proceedings of the Computer Security Foundations Workshop*, pp. 14–22, 1990.

[3] M. Burrows, M. Abadi and R. Needham, "A Logic of Authentication," *ACM Transactions on Computer Systems*, Vol. 8, No. 1, pp. 18–36, February 1990.

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

[5] A.D. Birrel, B.W. Lampson, R.M. Needham and M.D. Schroder, "A Global Authentication Service without Global Trust," *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pp. 223–230, 1986.

[6] S.M. Bellovin and M. Merritt, "Limitations of the Kerberos Authentication System," *Proceedings of the USENIX* Conference, Winter 1991.

[7] "CCITT X.509 The Directory-Authentication Framework," see also ISO 9594-8.

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

[9] D. Dolev and A.C. Yao, "On the Security of Public Key Protocols," *IEEE Transactions on Information Theory*, Vol. IT-30, No. 2, pp. 198–208, March 1983.

[10] S. Even and O. Goldreich, "On the Security of Multi-Party Protocols," *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pp. 34–39, 1983.

[11] M. Gasser, *Building a Secure Computer System*, Van Nostrand Reinhold Company, New York 1988.

[12] M. Gasser, A. Goldstein, C. Kaufman and B.W. Lampson, "The Digital Distributed System Security Architecture," *Proceedings of the National Computer Security Conference*, pp. 305–319, 1989.

[13] M. Gasser and E. McDermott, "An Architecture for Practical Delegation in a Distributed System," *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pp. 20–30, 1990.

[14] P.A. Karger, "Authentication and Discretionary Access Control in Computer Networks," *Computer Networks and ISDN Systems*, Vol. 10, pp. 27–37, 1985.

[15] J. Linn, "Practical Authentication for Distributed Computing," *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pp. 31–40, 1990.

[16] "RFC: Kerberos Version 5 Draft 4,", Network Working Group, MIT Project Athena, December 20, 1990.

[17] S.S. Lam, A.U. Shankar and T.Y.C. Woo, "Applying a Theory of Modules and Interfaces to Security Verification," *Proceedings of the IEEE Symposium on Research in Security and Privacy*, May 1991.

[18] J.H. Moore, "Protocol Failures in Cryptosystems," *Proceedings of the IEEE*, Vol. 76, No. 5, pp. 594–602, May 1988.

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

[20] M. Satyanarayanan, "Integrating Security in a Large Distributed System," *ACM Transactions on Computer Systems*, Vol. 7, No. 3, pp. 247–280, August 1989.

[21] J.G. Steiner, C. Neuman and J.I. Schiller, "*Kerberos*: An Authentication Service for Open Network Systems," *Proceedings of the USENIX* Conference, Winter 1988.

[22] K.R. Sollins, "Cascaded Authentication," *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pp. 156–163, 1988.

[23] J.J. Tardo and K. Alagappan, "SPX: Global Authentication Using Public Key Certificates," *Proceedings of the IEEE Symposium on Research in Security and Privacy*, May 1991.

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