CHAPTER 15

User Authentication

LEARNING OBJECTIVES

After studying this chapter, you should be able to:

- Understand the distinction between identification and verification.
- Present an overview of techniques for remote user authentication using symmetric encryption.
- Give a presentation on Kerberos.
- Explain the differences between versions 4 and 5 of Kerberos.
- Describe the use of Kerberos in multiple realms.
- Present an overview of techniques for remote user authentication using asymmetric encryption.
- Understand the need for a federated identity management system.
- Explain the use of PIV mechanisms as part of a user authentication system.

This chapter examines some of the authentication functions that have been developed to support network-based user authentication. The chapter begins with an introduction to some of the concepts and key considerations for user authentication over a network or the Internet. The next section examines user-authentication protocols that rely on symmetric encryption. This is followed by a section on one of the earliest and also one of the most widely used authentication services: Kerberos. Next, the chapter looks at user-authentication protocols that rely on asymmetric encryption. This is followed by a discussion of the X.509 user-authentication protocol. Finally, the concept of federated identity is introduced.

15.1 REMOTE USER-AUTHENTICATION PRINCIPLES

In most computer security contexts, user authentication is the fundamental building block and the primary line of defense. User authentication is the basis for most types of access control and for user accountability. RFC 4949 (*Internet Security Glossary*) defines user authentication as the process of verifying an identity claimed by or for a system entity. This process consists of two steps:

- **Identification step:** Presenting an identifier to the security system. (Identifiers should be assigned carefully, because authenticated identities are the basis for other security services, such as access control service.)
- **Verification step:** Presenting or generating authentication information that corroborates the binding between the entity and the identifier.

For example, user Alice Toklas could have the user identifier ABTOKLAS. This information needs to be stored on any server or computer system that Alice wishes to use and could be known to system administrators and other users.

A typical item of authentication information associated with this user ID is a password, which is kept secret (known only to Alice and to the system). If no one is able to obtain or guess Alice's password, then the combination of Alice's user ID and password enables administrators to set up Alice's access permissions and audit her activity. Because Alice's ID is not secret, system users can send her email, but because her password is secret, no one can pretend to be Alice.

In essence, identification is the means by which a user provides a claimed identity to the system; user authentication is the means of establishing the validity of the claim. Note that user authentication is distinct from message authentication. As defined in Chapter 12, message authentication is a procedure that allows communicating parties to verify that the contents of a received message have not been altered and that the source is authentic. This chapter is concerned solely with user authentication.

The NIST Model for Electronic User Authentication

NIST SP 800-63-2 (*Electronic Authentication Guideline,* August 2013) defines electronic user authentication as the process of establishing confidence in user identities that are presented electronically to an information system. Systems can use the authenticated identity to determine if the authenticated individual is authorized to perform particular functions, such as database transactions or access to system resources. In many cases, the authentication and transaction or other authorized function takes place across an open network such as the Internet. Equally authentication and subsequent authorization can take place locally, such as across a local area network.

SP 800-63-2 defines a general model for user authentication that involves a number of entities and procedures. We discuss this model with reference to Figure 15.1.

The initial requirement for performing user authentication is that the user must be registered with the system. The following is a typical sequence for registration. An applicant applies to a **registration authority (RA)** to become a **subscriber**

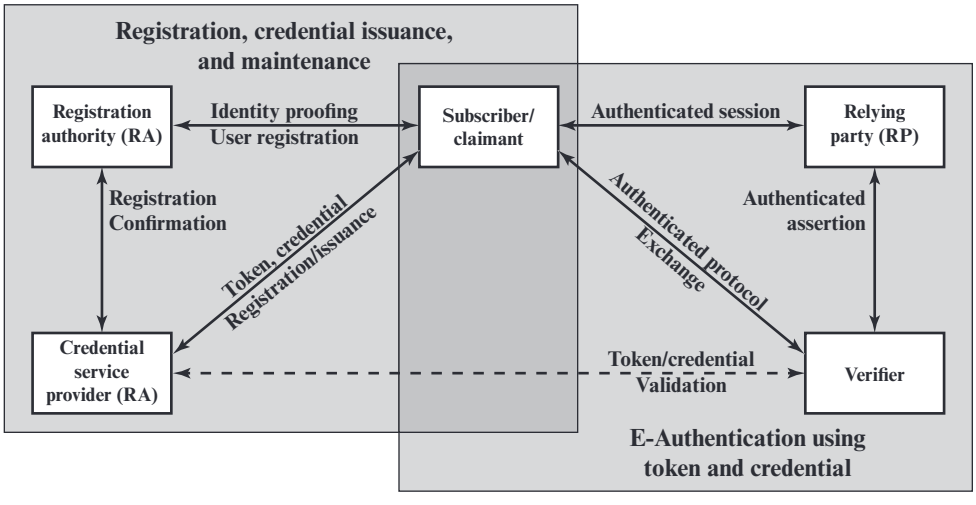


Figure 15.1 The NIST SP 800-63-2 E-Authentication Architectural Model

of a **credential service provider (CSP)**. In this model, the RA is a trusted entity that establishes and vouches for the identity of an applicant to a CSP. The CSP then engages in an exchange with the subscriber. Depending on the details of the overall authentication system, the CSP issues some sort of electronic credential to the subscriber. The **credential** is a data structure that authoritatively binds an identity and additional attributes to a token possessed by a subscriber, and can be verified when presented to the verifier in an authentication transaction. The token could be an encryption key or an encrypted password that identifies the subscriber. The token may be issued by the CSP, generated directly by the subscriber, or provided by a third party. The token and credential may be used in subsequent authentication events.

Once a user is registered as a subscriber, the actual authentication process can take place between the subscriber and one or more systems that perform authentication and, subsequently, authorization. The party to be authenticated is called a **claimant** and the party verifying that identity is called a **verifier**. When a claimant successfully demonstrates possession and control of a token to a verifier through an authentication protocol, the verifier can verify that the claimant is the subscriber named in the corresponding credential. The verifier passes on an assertion about the identity of the subscriber to the **relying party (RP)**. That assertion includes identity information about a subscriber, such as the subscriber name, an identifier assigned at registration, or other subscriber attributes that were verified in the registration process. The RP can use the authenticated information provided by the verifier to make access control or authorization decisions.

An implemented system for authentication will differ from or be more complex than this simplified model, but the model illustrates the key roles and functions needed for a secure authentication system.

Means of Authentication

There are four general means of authenticating a user's identity, which can be used alone or in combination:

- **Something the individual knows:** Examples include a password, a personal identification number (PIN), or answers to a prearranged set of questions.
- **Something the individual possesses:** Examples include cryptographic keys, electronic keycards, smart cards, and physical keys. This type of authenticator is referred to as a *token*.
- **Something the individual is (static biometrics):** Examples include recognition by fingerprint, retina, and face.
- **Something the individual does (dynamic biometrics):** Examples include recognition by voice pattern, handwriting characteristics, and typing rhythm.

All of these methods, properly implemented and used, can provide secure user authentication. However, each method has problems. An adversary may be able to guess or steal a password. Similarly, an adversary may be able to forge or steal a token. A user may forget a password or lose a token. Furthermore, there is a significant administrative overhead for managing password and token information on systems and securing such information on systems. With respect to biometric

authenticators, there are a variety of problems, including dealing with false positives and false negatives, user acceptance, cost, and convenience. For network-based user authentication, the most important methods involve cryptographic keys and something the individual knows, such as a password.

Mutual Authentication

An important application area is that of mutual authentication protocols. Such protocols enable communicating parties to satisfy themselves mutually about each other's identity and to exchange session keys. This topic was examined in Chapter 14. There, the focus was key distribution. We return to this topic here to consider the wider implications of authentication.

Central to the problem of authenticated key exchange are two issues: confidentiality and timeliness. To prevent masquerade and to prevent compromise of session keys, essential identification and session-key information must be communicated in encrypted form. This requires the prior existence of secret or public keys that can be used for this purpose. The second issue, timeliness, is important because of the threat of message replays. Such replays, at worst, could allow an opponent to compromise a session key or successfully impersonate another party. At minimum, a successful replay can disrupt operations by presenting parties with messages that appear genuine but are not.

[GONG93] lists the following examples of **replay attacks**:

1. The simplest replay attack is one in which the opponent simply copies a message and replays it later.
2. An opponent can replay a timestamped message within the valid time window. If both the original and the replay arrive within then time window, this incident can be logged.
3. As with example (2), an opponent can replay a timestamped message within the valid time window, but in addition, the opponent suppresses the original message. Thus, the repetition cannot be detected.
4. Another attack involves a backward replay without modification. This is a replay back to the message sender. This attack is possible if symmetric encryption is used and the sender cannot easily recognize the difference between messages sent and messages received on the basis of content.

One approach to coping with replay attacks is to attach a sequence number to each message used in an authentication exchange. A new message is accepted only if its sequence number is in the proper order. The difficulty with this approach is that it requires each party to keep track of the last sequence number for each claimant it has dealt with. Because of this overhead, sequence numbers are generally not used for authentication and key exchange. Instead, one of the following two general approaches is used:

- **Timestamps:** Party A accepts a message as fresh only if the message contains a **timestamp** that, in A's judgment, is close enough to A's knowledge of current time. This approach requires that clocks among the various participants be synchronized.

- **Challenge/response:** Party A, expecting a fresh message from B, first sends B a **nonce** (challenge) and requires that the subsequent message (response) received from B contain the correct nonce value.

It can be argued (e.g., [LAM92a]) that the timestamp approach should not be used for connection-oriented applications because of the inherent difficulties with this technique. First, some sort of protocol is needed to maintain synchronization among the various processor clocks. This protocol must be both fault tolerant, to cope with network errors, and secure, to cope with hostile attacks. Second, the opportunity for a successful attack will arise if there is a temporary loss of synchronization resulting from a fault in the clock mechanism of one of the parties. Finally, because of the variable and unpredictable nature of network delays, distributed clocks cannot be expected to maintain precise synchronization. Therefore, any timestamp-based procedure must allow for a window of time sufficiently large to accommodate network delays yet sufficiently small to minimize the opportunity for attack.

On the other hand, the challenge-response approach is unsuitable for a connectionless type of application, because it requires the overhead of a handshake before any connectionless transmission, effectively negating the chief characteristic of a connectionless transaction. For such applications, reliance on some sort of secure time server and a consistent attempt by each party to keep its clocks in synchronization may be the best approach (e.g., [LAM92b]).

One-Way Authentication

One application for which encryption is growing in popularity is electronic mail (email). The very nature of electronic mail, and its chief benefit, is that it is not necessary for the sender and receiver to be online at the same time. Instead, the email message is forwarded to the receiver's electronic mailbox, where it is buffered until the receiver is available to read it.

The "envelope" or header of the email message must be in the clear, so that the message can be handled by the store-and-forward email protocol, such as the Simple Mail Transfer Protocol (SMTP) or X.400. However, it is often desirable that the mail-handling protocol not require access to the plaintext form of the message, because that would require trusting the mail-handling mechanism. Accordingly, the email message should be encrypted such that the mail-handling system is not in possession of the decryption key.

A second requirement is that of **authentication**. Typically, the recipient wants some assurance that the message is from the alleged sender.

15.2 REMOTE USER-AUTHENTICATION USING SYMMETRIC ENCRYPTION

Mutual Authentication

As was discussed in Chapter 14, a two-level hierarchy of symmetric encryption keys can be used to provide confidentiality for communication in a distributed environment. In general, this strategy involves the use of a trusted key distribution center

(KDC). Each party in the network shares a secret key, known as a master key, with the KDC. The KDC is responsible for generating keys to be used for a short time over a connection between two parties, known as session keys, and for distributing those keys using the master keys to protect the distribution. This approach is quite common. As an example, we look at the Kerberos system in Section 15.3. The discussion in this subsection is relevant to an understanding of the Kerberos mechanisms.

Figure 14.3 illustrates a proposal initially put forth by Needham and Schroeder [NEED78] for secret key distribution using a KDC that, as was mentioned in Chapter 14, includes authentication features. The protocol can be summarized as follows.[1]

1. A → KDC: $ID_A \| ID_B \| N_1$
2. KDC → A: $E(K_a, [K_s \| ID_B \| N_1 \| E(K_b, [K_s \| ID_A])])$
3. A → B: $E(K_b, [K_s \| ID_A])$
4. B → A: $E(K_s, N_2)$
5. A → B: $E(K_s, f(N_2))$ where f() is a generic function that modifies the value of the nonce.

Secret keys K_a and K_b are shared between A and the KDC and B and the KDC, respectively. The purpose of the protocol is to distribute securely a session key K_s to A and B. Entity A securely acquires a new session key in step 2. The message in step 3 can be decrypted, and hence understood, only by B. Step 4 reflects B's knowledge of K_s, and step 5 assures B of A's knowledge of K_s and assures B that this is a fresh message because of the use of the nonce N_2. Recall from our discussion in Chapter 14 that the purpose of steps 4 and 5 is to prevent a certain type of replay attack. In particular, if an opponent is able to capture the message in step 3 and replay it, this might in some fashion disrupt operations at B.

Despite the handshake of steps 4 and 5, the protocol is still vulnerable to a form of replay attack. Suppose that an opponent, X, has been able to compromise an old session key. Admittedly, this is a much more unlikely occurrence than that an opponent has simply observed and recorded step 3. Nevertheless, it is a potential security risk. X can impersonate A and trick B into using the old key by simply replaying step 3. Unless B remembers indefinitely all previous session keys used with A, B will be unable to determine that this is a replay. If X can intercept the handshake message in step 4, then it can impersonate A's response in step 5. From this point on, X can send bogus messages to B that appear to B to come from A using an authenticated session key.

Denning [DENN81, DENN82] proposes to overcome this weakness by a modification to the Needham/Schroeder protocol that includes the addition of a timestamp to steps 2 and 3. Her proposal assumes that the master keys, K_a and K_b, are secure, and it consists of the following steps.

[1]The portion to the left of the colon indicates the sender and the receiver; the portion to the right indicates the contents of the message; the symbol ‖ indicates concatenation.

1. $A \rightarrow KDC: \quad ID_A \| ID_B$
2. $KDC \rightarrow A: \quad E(K_a, [K_s \| ID_B \| T \| E(K_b, [K_s \| ID_A \| T])])$
3. $A \rightarrow B: \quad E(K_b, [K_s \| ID_A \| T])$
4. $B \rightarrow A: \quad E(K_s, N_1)$
5. $A \rightarrow B: \quad E(K_s, f(N_1))$

T is a timestamp that assures A and B that the session key has only just been generated. Thus, both A and B know that the key distribution is a fresh exchange. A and B can verify timeliness by checking that

$$|\text{Clock} - T| < \Delta t_1 + \Delta t_2$$

where Δt_1 is the estimated normal discrepancy between the KDC's clock and the local clock (at A or B) and Δt_2 is the expected network delay time. Each node can set its clock against some standard reference source. Because the timestamp T is encrypted using the secure master keys, an opponent, even with knowledge of an old session key, cannot succeed because a replay of step 3 will be detected by B as untimely.

A final point: Steps 4 and 5 were not included in the original presentation [DENN81] but were added later [DENN82]. These steps confirm the receipt of the session key at B.

The Denning protocol seems to provide an increased degree of security compared to the Needham/Schroeder protocol. However, a new concern is raised: namely, that this new scheme requires reliance on clocks that are synchronized throughout the network. [GONG92] points out a risk involved. The risk is based on the fact that the distributed clocks can become unsynchronized as a result of sabotage on or faults in the clocks or the synchronization mechanism.[2] The problem occurs when a sender's clock is ahead of the intended recipient's clock. In this case, an opponent can intercept a message from the sender and replay it later when the timestamp in the message becomes current at the recipient's site. This replay could cause unexpected results. Gong refers to such attacks as **suppress-replay attacks**.

One way to counter suppress-replay attacks is to enforce the requirement that parties regularly check their clocks against the KDC's clock. The other alternative, which avoids the need for clock synchronization, is to rely on handshaking protocols using nonces. This latter alternative is not vulnerable to a suppress-replay attack, because the nonces the recipient will choose in the future are unpredictable to the sender. The Needham/Schroeder protocol relies on nonces only but, as we have seen, has other vulnerabilities.

In [KEHN92], an attempt is made to respond to the concerns about suppress-replay attacks and at the same time fix the problems in the Needham/Schroeder protocol. Subsequently, an inconsistency in this latter protocol was noted and an improved strategy was presented in [NEUM93a].[3] The protocol is

[2]Such things can and do happen. In recent years, flawed chips were used in a number of computers and other electronic systems to track the time and date. The chips had a tendency to skip forward one day. [NEUM90]
[3]It really is hard to get these things right.

1. $A \rightarrow B$: $ID_A \| N_a$
2. $B \rightarrow KDC$: $ID_B \| N_b \| E(K_b, [ID_A \| N_a \| T_b])$
3. $KDC \rightarrow A$: $E(K_a, [ID_B \| N_a \| K_s \| T_b]) \| E(K_b, [ID_A \| K_s \| T_b]) \| N_b$
4. $A \rightarrow B$: $E(K_b, [ID_A \| K_s \| T_b]) \| E(K_s, N_b)$

Let us follow this exchange step by step.

1. A initiates the authentication exchange by generating a nonce, N_a, and sending that plus its identifier to B in plaintext. This nonce will be returned to A in an encrypted message that includes the session key, assuring A of its timeliness.
2. B alerts the KDC that a session key is needed. Its message to the KDC includes its identifier and a nonce, N_b. This nonce will be returned to B in an encrypted message that includes the session key, assuring B of its timeliness. B's message to the KDC also includes a block encrypted with the secret key shared by B and the KDC. This block is used to instruct the KDC to issue credentials to A; the block specifies the intended recipient of the credentials, a suggested expiration time for the credentials, and the nonce received from A.
3. The KDC passes on to A B's nonce and a block encrypted with the secret key that B shares with the KDC. The block serves as a "ticket" that can be used by A for subsequent authentications, as will be seen. The KDC also sends to A a block encrypted with the secret key shared by A and the KDC. This block verifies that B has received A's initial message (ID_B) and that this is a timely message and not a replay (N_a), and it provides A with a session key (K_s) and the time limit on its use (T_b).
4. A transmits the ticket to B, together with the B's nonce, the latter encrypted with the session key. The ticket provides B with the secret key that is used to decrypt $E(K_s, N_b)$ to recover the nonce. The fact that B's nonce is encrypted with the session key authenticates that the message came from A and is not a replay.

This protocol provides an effective, secure means for A and B to establish a session with a secure session key. Furthermore, the protocol leaves A in possession of a key that can be used for subsequent authentication to B, avoiding the need to contact the authentication server repeatedly. Suppose that A and B establish a session using the aforementioned protocol and then conclude that session. Subsequently, but within the time limit established by the protocol, A desires a new session with B. The following protocol ensues:

1. $A \rightarrow B$: $E(K_b, [ID_A \| K_s \| T_b]) \| N'_a$
2. $B \rightarrow A$: $N'_b \| E(K_s, N'_a)$
3. $A \rightarrow B$: $E(K_s, N'_b)$

When B receives the message in step 1, it verifies that the ticket has not expired. The newly generated nonces N'_a and N'_b assure each party that there is no replay attack.

In all the foregoing, the time specified in T_b is a time relative to B's clock. Thus, this timestamp does not require synchronized clocks, because B checks only self-generated timestamps.

One-Way Authentication

Using symmetric encryption, the decentralized key distribution scenario illustrated in Figure 14.5 is impractical. This scheme requires the sender to issue a request to the intended recipient, await a response that includes a session key, and only then send the message.

With some refinement, the KDC strategy illustrated in Figure 14.3 is a candidate for encrypted electronic mail. Because we wish to avoid requiring that the recipient (B) be on line at the same time as the sender (A), steps 4 and 5 must be eliminated. For a message with content M, the sequence is as follows:

1. $A \rightarrow KDC: \quad ID_A \| ID_B \| N_1$
2. $KDC \rightarrow A: \quad E(K_a, [K_s \| ID_B \| N_1 \| E(K_b, [K_s \| ID_A])])$
3. $A \rightarrow B: \quad E(K_b, [K_s \| ID_A]) \| E(K_s, M)$

This approach guarantees that only the intended recipient of a message will be able to read it. It also provides a level of authentication that the sender is A. As specified, the protocol does not protect against replays. Some measure of defense could be provided by including a timestamp with the message. However, because of the potential delays in the email process, such timestamps may have limited usefulness.

15.3 KERBEROS

Kerberos[4] is an authentication service developed as part of Project Athena at MIT. The problem that Kerberos addresses is this: Assume an open distributed environment in which users at workstations wish to access services on servers distributed throughout the network. We would like for servers to be able to restrict access to authorized users and to be able to authenticate requests for service. In this environment, a workstation cannot be trusted to identify its users correctly to network services. In particular, the following three threats exist:

1. A user may gain access to a particular workstation and pretend to be another user operating from that workstation.
2. A user may alter the network address of a workstation so that the requests sent from the altered workstation appear to come from the impersonated workstation.
3. A user may eavesdrop on exchanges and use a replay attack to gain entrance to a server or to disrupt operations.

In any of these cases, an unauthorized user may be able to gain access to services and data that he or she is not authorized to access. Rather than building in elaborate

[4]"In Greek mythology, a many headed dog, commonly three, perhaps with a serpent's tail, the guardian of the entrance of Hades." From *Dictionary of Subjects and Symbols in Art*, by James Hall, Harper & Row, 1979. Just as the Greek Kerberos has three heads, the modern Kerberos was intended to have three components to guard a network's gate: authentication, accounting, and audit. The last two heads were never implemented.

authentication protocols at each server, Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users. Unlike most other authentication schemes described in this book, Kerberos relies exclusively on symmetric encryption, making no use of public-key encryption.

Two versions of Kerberos are in common use. Version 4 [MILL88, STEI88] implementations still exist. Version 5 [KOHL94] corrects some of the security deficiencies of version 4 and has been issued as a proposed Internet Standard (RFC 4120 and RFC 4121).[5]

We begin this section with a brief discussion of the motivation for the Kerberos approach. Then, because of the complexity of Kerberos, it is best to start with a description of the authentication protocol used in version 4. This enables us to see the essence of the Kerberos strategy without considering some of the details required to handle subtle security threats. Finally, we examine version 5.

Motivation

If a set of users is provided with dedicated personal computers that have no network connections, then a user's resources and files can be protected by physically securing each personal computer. When these users instead are served by a centralized time-sharing system, the time-sharing operating system must provide the security. The operating system can enforce access-control policies based on user identity and use the logon procedure to identify users.

Today, neither of these scenarios is typical. More common is a distributed architecture consisting of dedicated user workstations (clients) and distributed or centralized servers. In this environment, three approaches to security can be envisioned.

1. Rely on each individual client workstation to assure the identity of its user or users and rely on each server to enforce a security policy based on user identification (ID).
2. Require that client systems authenticate themselves to servers, but trust the client system concerning the identity of its user.
3. Require the user to prove his or her identity for each service invoked. Also require that servers prove their identity to clients.

In a small, closed environment in which all systems are owned and operated by a single organization, the first or perhaps the second strategy may suffice.[6] But in a more open environment in which network connections to other machines are supported, the third approach is needed to protect user information and resources housed at the server. Kerberos supports this third approach. Kerberos assumes a distributed client/server architecture and employs one or more Kerberos servers to provide an authentication service.

---

[5]Versions 1 through 3 were internal development versions. Version 4 is the "original" Kerberos.
[6]However, even a closed environment faces the threat of attack by a disgruntled employee.

The first published report on Kerberos [STEI88] listed the following requirements.

- **Secure:** A network eavesdropper should not be able to obtain the necessary information to impersonate a user. More generally, Kerberos should be strong enough that a potential opponent does not find it to be the weak link.
- **Reliable:** For all services that rely on Kerberos for access control, lack of availability of the Kerberos service means lack of availability of the supported services. Hence, Kerberos should be highly reliable and should employ a distributed server architecture with one system able to back up another.
- **Transparent:** Ideally, the user should not be aware that authentication is taking place beyond the requirement to enter a password.
- **Scalable:** The system should be capable of supporting large numbers of clients and servers. This suggests a modular, distributed architecture.

To support these requirements, the overall scheme of Kerberos is that of a trusted third-party authentication service that uses a protocol based on that proposed by Needham and Schroeder [NEED78], which was discussed in Section 15.2. It is trusted in the sense that clients and servers trust Kerberos to mediate their mutual authentication. Assuming the Kerberos protocol is well designed, then the authentication service is secure if the Kerberos server itself is secure.[7]

Kerberos Version 4

Version 4 of Kerberos makes use of DES, in a rather elaborate protocol, to provide the authentication service. Viewing the protocol as a whole, it is difficult to see the need for the many elements contained therein. Therefore, we adopt a strategy used by Bill Bryant of Project Athena [BRYA88] and build up to the full protocol by looking first at several hypothetical dialogues. Each successive dialogue adds additional complexity to counter security vulnerabilities revealed in the preceding dialogue.

After examining the protocol, we look at some other aspects of version 4.

A Simple Authentication Dialogue In an unprotected network environment, any client can apply to any server for service. The obvious security risk is that of impersonation. An opponent can pretend to be another client and obtain unauthorized privileges on server machines. To counter this threat, servers must be able to confirm the identities of clients who request service. Each server can be required to undertake this task for each client/server interaction, but in an open environment, this places a substantial burden on each server.

---

[7]Remember that the security of the Kerberos server should not automatically be assumed but must be guarded carefully (e.g., in a locked room). It is well to remember the fate of the Greek Kerberos, whom Hercules was ordered by Eurystheus to capture as his Twelfth Labor: "Hercules found the great dog on its chain and seized it by the throat. At once the three heads tried to attack, and Kerberos lashed about with his powerful tail. Hercules hung on grimly, and Kerberos relaxed into unconsciousness. Eurystheus may have been surprised to see Hercules alive—when he saw the three slavering heads and the huge dog they belonged to he was frightened out of his wits, and leapt back into the safety of his great bronze jar." From *The Hamlyn Concise Dictionary of Greek and Roman Mythology*, by Michael Stapleton, Hamlyn, 1982.

An alternative is to use an authentication server (AS) that knows the passwords of all users and stores these in a centralized database. In addition, the AS shares a unique secret key with each server. These keys have been distributed physically or in some other secure manner. Consider the following hypothetical dialogue:

$$\begin{aligned}
&\textbf{(1)}\ \mathrm{C} \rightarrow \mathrm{AS}: && ID_C \| P_C \| ID_V \\
&\textbf{(2)}\ \mathrm{AS} \rightarrow \mathrm{C}: && Ticket \\
&\textbf{(3)}\ \mathrm{C} \rightarrow \mathrm{V}: && ID_C \| Ticket \\
&Ticket = E(K_v, [ID_C \| AD_C \| ID_V])
\end{aligned}$$

where

$$\begin{aligned}
\mathrm{C} &= \text{client} \\
\mathrm{AS} &= \text{authentication server} \\
\mathrm{V} &= \text{server} \\
ID_C &= \text{identifier of user on C} \\
ID_V &= \text{identifier of V} \\
P_C &= \text{password of user on C} \\
AD_C &= \text{network address of C} \\
K_v &= \text{secret encryption key shared by AS and V}
\end{aligned}$$

In this scenario, the user logs on to a workstation and requests access to server V. The client module C in the user's workstation requests the user's password and then sends a message to the AS that includes the user's ID, the server's ID, and the user's password. The AS checks its database to see if the user has supplied the proper password for this user ID and whether this user is permitted access to server V. If both tests are passed, the AS accepts the user as authentic and must now convince the server that this user is authentic. To do so, the AS creates a **ticket** that contains the user's ID and network address and the server's ID. This ticket is encrypted using the secret key shared by the AS and this server. This ticket is then sent back to C. Because the ticket is encrypted, it cannot be altered by C or by an opponent.

With this ticket, C can now apply to V for service. C sends a message to V containing C's ID and the ticket. V decrypts the ticket and verifies that the user ID in the ticket is the same as the unencrypted user ID in the message. If these two match, the server considers the user authenticated and grants the requested service.

Each of the ingredients of message (3) is significant. The ticket is encrypted to prevent alteration or forgery. The server's ID (ID_V) is included in the ticket so that the server can verify that it has decrypted the ticket properly. ID_C is included in the ticket to indicate that this ticket has been issued on behalf of C. Finally, AD_C serves to counter the following threat. An opponent could capture the ticket transmitted in message (2), then use the name ID_C and transmit a message of form (3) from another workstation. The server would receive a valid ticket that matches the user ID and grant access to the user on that other workstation. To prevent this attack, the AS includes in the ticket the network address from which the original request came. Now the ticket is valid only if it is transmitted from the same workstation that initially requested the ticket.

A More Secure Authentication Dialogue Although the foregoing scenario solves some of the problems of authentication in an open network environment, problems remain. Two in particular stand out. First, we would like to minimize the number of times that a user has to enter a password. Suppose each ticket can be used only once. If user C logs on to a workstation in the morning and wishes to check his or her mail at a mail server, C must supply a password to get a ticket for the mail server. If C wishes to check the mail several times during the day, each attempt requires reentering the password. We can improve matters by saying that tickets are reusable. For a single logon session, the workstation can store the mail server ticket after it is received and use it on behalf of the user for multiple accesses to the mail server.

However, under this scheme, it remains the case that a user would need a new ticket for every different service. If a user wished to access a print server, a mail server, a file server, and so on, the first instance of each access would require a new ticket and hence require the user to enter the password.

The second problem is that the earlier scenario involved a plaintext transmission of the password [message (1)]. An eavesdropper could capture the password and use any service accessible to the victim.

To solve these additional problems, we introduce a scheme for avoiding plaintext passwords and a new server, known as the **ticket-granting server (TGS)**. The new (but still hypothetical) scenario is as follows.

Once per user logon session:

(1) C → AS: $ID_C \| ID_{tgs}$

(2) AS → C: $\mathrm{E}(K_c, Ticket_{tgs})$

Once per type of service:

(3) C → TGS: $ID_C \| ID_V \| Ticket_{tgs}$

(4) TGS → C: $Ticket_v$

Once per service session:

(5) C → V: $ID_C \| Ticket_v$

$$Ticket_{tgs} = \mathrm{E}(K_{tgs}, [ID_C \| AD_C \| ID_{tgs} \| TS_1 \| Lifetime_1])$$

$$Ticket_v = \mathrm{E}(K_v, [ID_C \| AD_C \| ID_v \| TS_2 \| Lifetime_2])$$

The new service, TGS, issues tickets to users who have been authenticated to AS. Thus, the user first requests a ticket-granting ticket ($Ticket_{tgs}$) from the AS. The client module in the user workstation saves this ticket. Each time the user requires access to a new service, the client applies to the TGS, using the ticket to authenticate itself. The TGS then grants a ticket for the particular service. The client saves each service-granting ticket and uses it to authenticate its user to a server each time a particular service is requested. Let us look at the details of this scheme:

1. The client requests a ticket-granting ticket on behalf of the user by sending its user's ID to the AS, together with the TGS ID, indicating a request to use the TGS service.

2. The AS responds with a ticket that is encrypted with a key that is derived from the user's password (K_c), which is already stored at the AS. When this response arrives at the client, the client prompts the user for his or her password, generates the key, and attempts to decrypt the incoming message. If the correct password is supplied, the ticket is successfully recovered.

Because only the correct user should know the password, only the correct user can recover the ticket. Thus, we have used the password to obtain credentials from Kerberos without having to transmit the password in plaintext. The ticket itself consists of the ID and network address of the user, and the ID of the TGS. This corresponds to the first scenario. The idea is that the client can use this ticket to request multiple service-granting tickets. So the ticket-granting ticket is to be reusable. However, we do not wish an opponent to be able to capture the ticket and use it. Consider the following scenario: An opponent captures the login ticket and waits until the user has logged off his or her workstation. Then the opponent either gains access to that workstation or configures his workstation with the same network address as that of the victim. The opponent would be able to reuse the ticket to spoof the TGS. To counter this, the ticket includes a timestamp, indicating the date and time at which the ticket was issued, and a lifetime, indicating the length of time for which the ticket is valid (e.g., eight hours). Thus, the client now has a reusable ticket and need not bother the user for a password for each new service request. Finally, note that the ticket-granting ticket is encrypted with a secret key known only to the AS and the TGS. This prevents alteration of the ticket. The ticket is reencrypted with a key based on the user's password. This assures that the ticket can be recovered only by the correct user, providing the authentication.

Now that the client has a ticket-granting ticket, access to any server can be obtained with steps 3 and 4.

3. The client requests a service-granting ticket on behalf of the user. For this purpose, the client transmits a message to the TGS containing the user's ID, the ID of the desired service, and the ticket-granting ticket.
4. The TGS decrypts the incoming ticket using a key shared only by the AS and the TGS (K_{tgs}) and verifies the success of the decryption by the presence of its ID. It checks to make sure that the lifetime has not expired. Then it compares the user ID and network address with the incoming information to authenticate the user. If the user is permitted access to the server V, the TGS issues a ticket to grant access to the requested service.

The service-granting ticket has the same structure as the ticket-granting ticket. Indeed, because the TGS is a server, we would expect that the same elements are needed to authenticate a client to the TGS and to authenticate a client to an application server. Again, the ticket contains a timestamp and lifetime. If the user wants access to the same service at a later time, the client can simply use the previously acquired service-granting ticket and need not bother the user for a password. Note that the ticket is encrypted with a secret key (K_v) known only to the TGS and the server, preventing alteration.

Finally, with a particular service-granting ticket, the client can gain access to the corresponding service with step 5.

5. The client requests access to a service on behalf of the user. For this purpose, the client transmits a message to the server containing the user's ID and the service-granting ticket. The server authenticates by using the contents of the ticket.

This new scenario satisfies the two requirements of only one password query per user session and protection of the user password.

The Version 4 Authentication Dialogue Although the foregoing scenario enhances security compared to the first attempt, two additional problems remain. The heart of the first problem is the lifetime associated with the ticket-granting ticket. If this lifetime is very short (e.g., minutes), then the user will be repeatedly asked for a password. If the lifetime is long (e.g., hours), then an opponent has a greater opportunity for replay. An opponent could eavesdrop on the network and capture a copy of the ticket-granting ticket and then wait for the legitimate user to log out. Then the opponent could forge the legitimate user's network address and send the message of step (3) to the TGS. This would give the opponent unlimited access to the resources and files available to the legitimate user.

Similarly, if an opponent captures a service-granting ticket and uses it before it expires, the opponent has access to the corresponding service.

Thus, we arrive at an additional requirement. A network service (the TGS or an application service) must be able to prove that the person using a ticket is the same person to whom that ticket was issued.

The second problem is that there may be a requirement for servers to authenticate themselves to users. Without such authentication, an opponent could sabotage the configuration so that messages to a server were directed to another location. The false server would then be in a position to act as a real server and capture any information from the user and deny the true service to the user.

We examine these problems in turn and refer to Table 15.1, which shows the actual Kerberos protocol. Figure 15.2 provides a simplified overview.

Table 15.1 Summary of Kerberos Version 4 Message Exchanges

(1)	$\mathbf{C} \rightarrow \mathbf{AS}$	$ID_c \| ID_{tgs} \| TS_1$
(2)	$\mathbf{AS} \rightarrow \mathbf{C}$	$\mathrm{E}(K_c, [K_{c,tgs} \| ID_{tgs} \| TS_2 \| Lifetime_2 \| Ticket_{tgs}])$
		$Ticket_{tgs} = \mathrm{E}(\mathrm{K}_{tgs}, [\mathrm{K}_{c,tgs} \| \mathrm{ID}_C \| \mathrm{AD}_C \| \mathrm{ID}_{tgs} \| \mathrm{TS}_2 \| \mathrm{Lifetime}_2])$

(a) Authentication Service Exchange to obtain ticket-granting ticket

(3)	$\mathbf{C} \rightarrow \mathbf{TGS}$	$ID_v \| Ticket_{tgs} \| Authenticator_c$
(4)	$\mathbf{TGS} \rightarrow \mathbf{C}$	$\mathrm{E}(K_{c,tgs}, [K_{c,v} \| ID_v \| TS_4 \| Ticket_v])$
		$Ticket_{tgs} = \mathrm{E}(\mathrm{K}_{tgs}, [\mathrm{K}_{c,tgs} \| \mathrm{ID}_C \| \mathrm{AD}_C \| \mathrm{ID}_{tgs} \| \mathrm{TS}_2 \| \mathrm{Lifetime}_2])$
		$Ticket_v = \mathrm{E}(\mathrm{K}_v, [\mathrm{K}_{c,v} \| \mathrm{ID}_C \| \mathrm{AD}_C \| \mathrm{ID}_v \| \mathrm{TS}_4 \| \mathrm{Lifetime}_4])$
		$Authenticator_c = \mathrm{E}(\mathrm{K}_{c,tgs}, [\mathrm{ID}_C \| \mathrm{AD}_C \| \mathrm{TS}_3])$

(b) Ticket-Granting Service Exchange to obtain service-granting ticket

(5)	$\mathbf{C} \rightarrow \mathbf{V}$	$Ticket_v \| Authenticator_c$
(6)	$\mathbf{V} \rightarrow \mathbf{C}$	$\mathrm{E}(K_{c,v}, [TS_5 + 1])$ (for mutual authentication)
		$Ticket_v = \mathrm{E}(\mathrm{K}_v, [\mathrm{K}_{c,v} \| \mathrm{ID}_C \| \mathrm{AD}_C \| \mathrm{ID}_v \| \mathrm{TS}_4 \| \mathrm{Lifetime}_4])$
		$Authenticator_c = \mathrm{E}(\mathrm{K}_{c,v}, [\mathrm{ID}_C \| \mathrm{AD}_C \| \mathrm{TS}_5])$

(c) Client/Server Authentication Exchange to obtain service

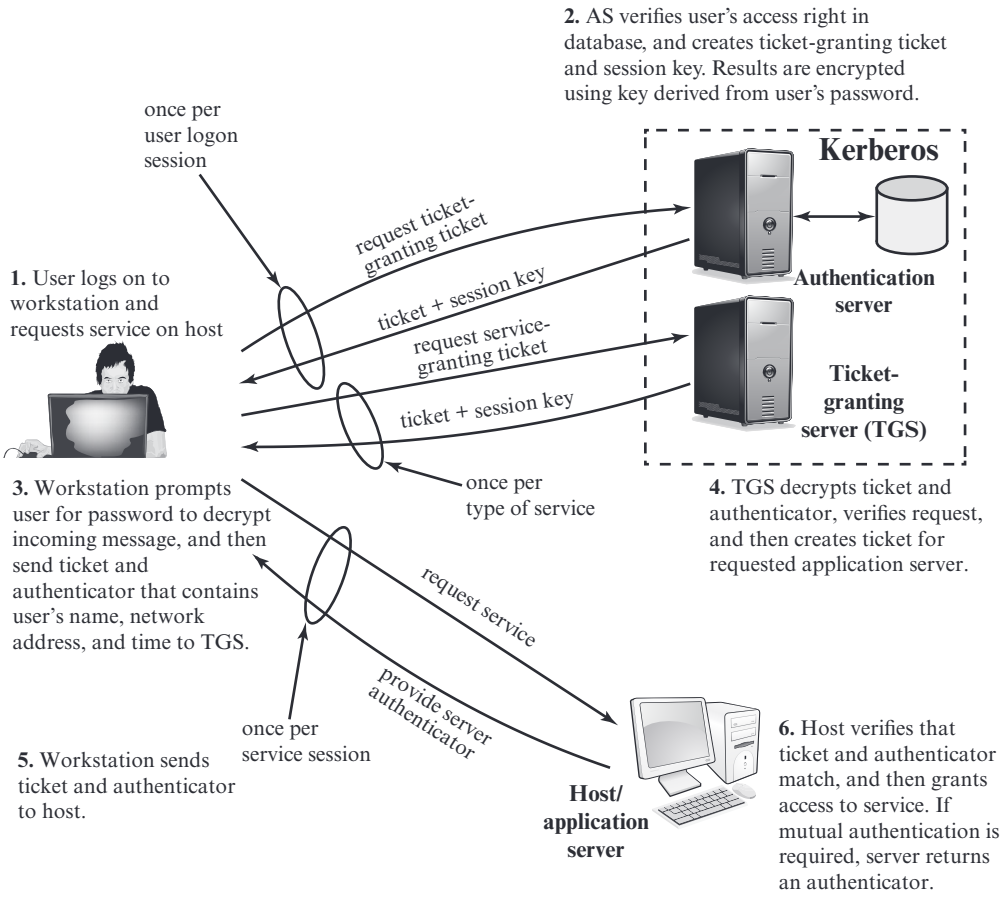


Figure 15.2 Overview of Kerberos

First, consider the problem of captured ticket-granting tickets and the need to determine that the ticket presenter is the same as the client for whom the ticket was issued. The threat is that an opponent will steal the ticket and use it before it expires. To get around this problem, let us have the AS provide both the client and the TGS with a secret piece of information in a secure manner. Then the client can prove its identity to the TGS by revealing the secret information—again in a secure manner. An efficient way of accomplishing this is to use an encryption key as the secure information; this is referred to as a session key in Kerberos.

Table 15.1a shows the technique for distributing the session key. As before, the client sends a message to the AS requesting access to the TGS. The AS responds with a message, encrypted with a key derived from the user's password (K_c), that contains the ticket. The encrypted message also contains a copy of the session key, $K_{c,tgs}$, where the subscripts indicate that this is a session key for C and TGS. Because this session key is inside the message encrypted with K_c, only the user's client can read it. The same session key is included in the ticket, which can be read only by the TGS. Thus, the session key has been securely delivered to both C and the TGS.

Note that several additional pieces of information have been added to this first phase of the dialogue. Message (1) includes a timestamp, so that the AS knows that the message is timely. Message (2) includes several elements of the ticket in a form accessible to C. This enables C to confirm that this ticket is for the TGS and to learn its expiration time.

Armed with the ticket and the session key, C is ready to approach the TGS. As before, C sends the TGS a message that includes the ticket plus the ID of the requested service [message (3) in Table 15.1b]. In addition, C transmits an authenticator, which includes the ID and address of C's user and a timestamp. Unlike the ticket, which is reusable, the authenticator is intended for use only once and has a very short lifetime. The TGS can decrypt the ticket with the key that it shares with the AS. This ticket indicates that user C has been provided with the session key $K_{c,tgs}$. In effect, the ticket says, "Anyone who uses $K_{c,tgs}$ must be C." The TGS uses the session key to decrypt the authenticator. The TGS can then check the name and address from the authenticator with that of the ticket and with the network address of the incoming message. If all match, then the TGS is assured that the sender of the ticket is indeed the ticket's real owner. In effect, the authenticator says, "At time TS_3, I hereby use $K_{c,tgs}$." Note that the ticket does not prove anyone's identity but is a way to distribute keys securely. It is the authenticator that proves the client's identity. Because the authenticator can be used only once and has a short lifetime, the threat of an opponent stealing both the ticket and the authenticator for presentation later is countered.

The reply from the TGS in message (4) follows the form of message (2). The message is encrypted with the session key shared by the TGS and C and includes a session key to be shared between C and the server V, the ID of V, and the timestamp of the ticket. The ticket itself includes the same session key.

C now has a reusable service-granting ticket for V. When C presents this ticket, as shown in message (5), it also sends an authenticator. The server can decrypt the ticket, recover the session key, and decrypt the authenticator.

If mutual authentication is required, the server can reply as shown in message (6) of Table 15.1. The server returns the value of the timestamp from the authenticator, incremented by 1, and encrypted in the session key. C can decrypt this message to recover the incremented timestamp. Because the message was encrypted by the session key, C is assured that it could have been created only by V. The contents of the message assure C that this is not a replay of an old reply.

Finally, at the conclusion of this process, the client and server share a secret key. This key can be used to encrypt future messages between the two or to exchange a new random session key for that purpose.

Figure 15.3 illustrates the Kerberos exchanges among the parties. Table 15.2 summarizes the justification for each of the elements in the Kerberos protocol.

Kerberos Realms and Multiple Kerberi A full-service Kerberos environment consisting of a Kerberos server, a number of clients, and a number of application servers requires the following:

1. The Kerberos server must have the user ID and hashed passwords of all participating users in its database. All users are registered with the Kerberos server.
2. The Kerberos server must share a secret key with each server. All servers are registered with the Kerberos server.

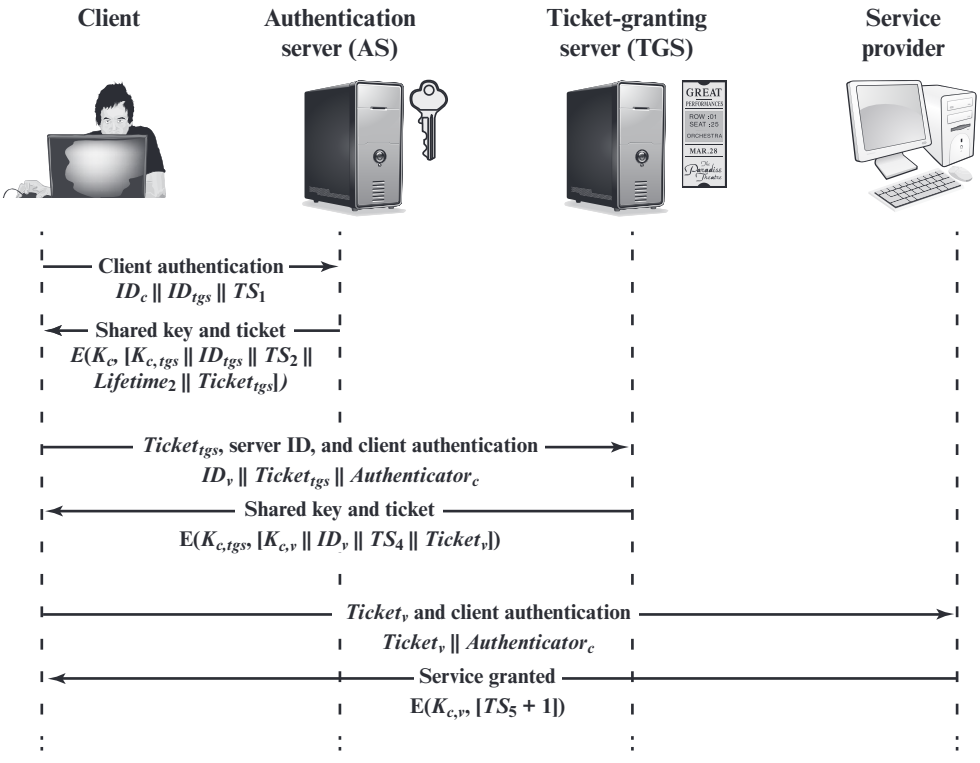


Figure 15.3 Kerberos Exchanges

Table 15.2 Rationale for the Elements of the Kerberos Version 4 Protocol

Message (1)	Client requests ticket-granting ticket.
ID_C	Tells AS identity of user from this client.
ID_{tgs}	Tells AS that user requests access to TGS.
TS_1	Allows AS to verify that client's clock is synchronized with that of AS.
Message (2)	AS returns ticket-granting ticket.
K_c	Encryption is based on user's password, enabling AS and client to verify password, and protecting contents of message (2).
$K_{c,\,tgs}$	Copy of session key accessible to client created by AS to permit secure exchange between client and TGS without requiring them to share a permanent key.
ID_{tgs}	Confirms that this ticket is for the TGS.
TS_2	Informs client of time this ticket was issued.
$Lifetime_2$	Informs client of the lifetime of this ticket.
$Ticket_{tgs}$	Ticket to be used by client to access TGS.

(a) Authentication Service Exchange

Message (3)	Client requests service-granting ticket.
ID_V	Tells TGS that user requests access to server V.
$Ticket_{tgs}$	Assures TGS that this user has been authenticated by AS.
$Authenticator_c$	Generated by client to validate ticket.

(*Continued*)

Table 15.2 Continued

Message (4)	TGS returns service-granting ticket.
$K_{c,\,tgs}$	Key shared only by C and TGS protects contents of message (4).
$K_{c,\,v}$	Copy of session key accessible to client created by TGS to permit secure exchange between client and server without requiring them to share a permanent key.
ID_V	Confirms that this ticket is for server V.
TS_4	Informs client of time this ticket was issued.
$Ticket_V$	Ticket to be used by client to access server V.
$Ticket_{tgs}$	Reusable so that user does not have to reenter password.
K_{tgs}	Ticket is encrypted with key known only to AS and TGS, to prevent tampering.
$K_{c,\,tgs}$	Copy of session key accessible to TGS used to decrypt authenticator, thereby authenticating ticket.
ID_C	Indicates the rightful owner of this ticket.
AD_C	Prevents use of ticket from workstation other than one that initially requested the ticket.
ID_{tgs}	Assures server that it has decrypted ticket properly.
TS_2	Informs TGS of time this ticket was issued.
$Lifetime_2$	Prevents replay after ticket has expired.
$Authenticator_c$	Assures TGS that the ticket presenter is the same as the client for whom the ticket was issued has very short lifetime to prevent replay.
$K_{c,\,tgs}$	Authenticator is encrypted with key known only to client and TGS, to prevent tampering.
ID_C	Must match ID in ticket to authenticate ticket.
AD_C	Must match address in ticket to authenticate ticket.
TS_3	Informs TGS of time this authenticator was generated.

(b) Ticket-Granting Service Exchange

Message (5)	Client requests service.
$Ticket_V$	Assures server that this user has been authenticated by AS.
$Authenticator_c$	Generated by client to validate ticket.
Message (6)	Optional authentication of server to client.
$K_{c,\,v}$	Assures C that this message is from V.
$TS_5 + 1$	Assures C that this is not a replay of an old reply.
$Ticket_v$	Reusable so that client does not need to request a new ticket from TGS for each access to the same server.
K_v	Ticket is encrypted with key known only to TGS and server, to prevent tampering.
$K_{c,\,v}$	Copy of session key accessible to client; used to decrypt authenticator, thereby authenticating ticket.
ID_C	Indicates the rightful owner of this ticket.
AD_C	Prevents use of ticket from workstation other than one that initially requested the ticket.
ID_V	Assures server that it has decrypted ticket properly.
TS_4	Informs server of time this ticket was issued.
$Lifetime_4$	Prevents replay after ticket has expired.
$Authenticator_c$	Assures server that the ticket presenter is the same as the client for whom the ticket was issued; has very short lifetime to prevent replay.
$K_{c,\,v}$	Authenticator is encrypted with key known only to client and server, to prevent tampering.
ID_C	Must match ID in ticket to authenticate ticket.
AD_C	Must match address in ticket to authenticate ticket.
TS_5	Informs server of time this authenticator was generated.

(c) Client/Server Authentication Exchange

Such an environment is referred to as a **Kerberos realm**. The concept of **realm** can be explained as follows. A Kerberos realm is a set of managed nodes that share the same Kerberos database. The Kerberos database resides on the Kerberos master computer system, which should be kept in a physically secure room. A read-only copy of the Kerberos database might also reside on other Kerberos computer systems. However, all changes to the database must be made on the master computer system. Changing or accessing the contents of a Kerberos database requires the Kerberos master password. A related concept is that of a **Kerberos principal**, which is a service or user that is known to the Kerberos system. Each Kerberos principal is identified by its principal name. Principal names consist of three parts: a service or user name, an instance name, and a realm name.

Networks of clients and servers under different administrative organizations typically constitute different realms. That is, it generally is not practical or does not conform to administrative policy to have users and servers in one administrative domain registered with a Kerberos server elsewhere. However, users in one realm may need access to servers in other realms, and some servers may be willing to provide service to users from other realms, provided that those users are authenticated.

Kerberos provides a mechanism for supporting such interrealm authentication. For two realms to support interrealm authentication, a third requirement is added:

3. The Kerberos server in each interoperating realm shares a secret key with the server in the other realm. The two Kerberos servers are registered with each other.

The scheme requires that the Kerberos server in one realm trust the Kerberos server in the other realm to authenticate its users. Furthermore, the participating servers in the second realm must also be willing to trust the Kerberos server in the first realm.

With these ground rules in place, we can describe the mechanism as follows (Figure 15.4): A user wishing service on a server in another realm needs a ticket for that server. The user's client follows the usual procedures to gain access to the local TGS and then requests a ticket-granting ticket for a remote TGS (TGS in another realm). The client can then apply to the remote TGS for a service-granting ticket for the desired server in the realm of the remote TGS.

The details of the exchanges illustrated in Figure 15.4 are as follows (compare Table 15.1).

(1) C → AS: $ID_c \| ID_{tgs} \| TS_1$

(2) AS → C: $E(K_c, [K_{c,\,tgs} \| ID_{tgs} \| TS_2 \| Lifetime_2 \| Ticket_{tgs}])$

(3) C → TGS: $ID_{tgsrem} \| Ticket_{tgs} \| Authenticator_c$

(4) TGS → C: $E(K_{c,tgs}, [K_{c,\,tgsrem} \| ID_{tgsrem} \| TS_4 \| Ticket_{tgsrem}])$

(5) C → TGS_{rem}: $ID_{vrem} \| Ticket_{tgsrem} \| Authenticator_c$

(6) TGS_{rem} → C: $E(K_{c,tgsrem}, [K_{c,\,vrem} \| ID_{vrem} \| TS_6 \| Ticket_{vrem}])$

(7) C → V_{rem}: $Ticket_{\text{vrem}} \| Authenticator_c$

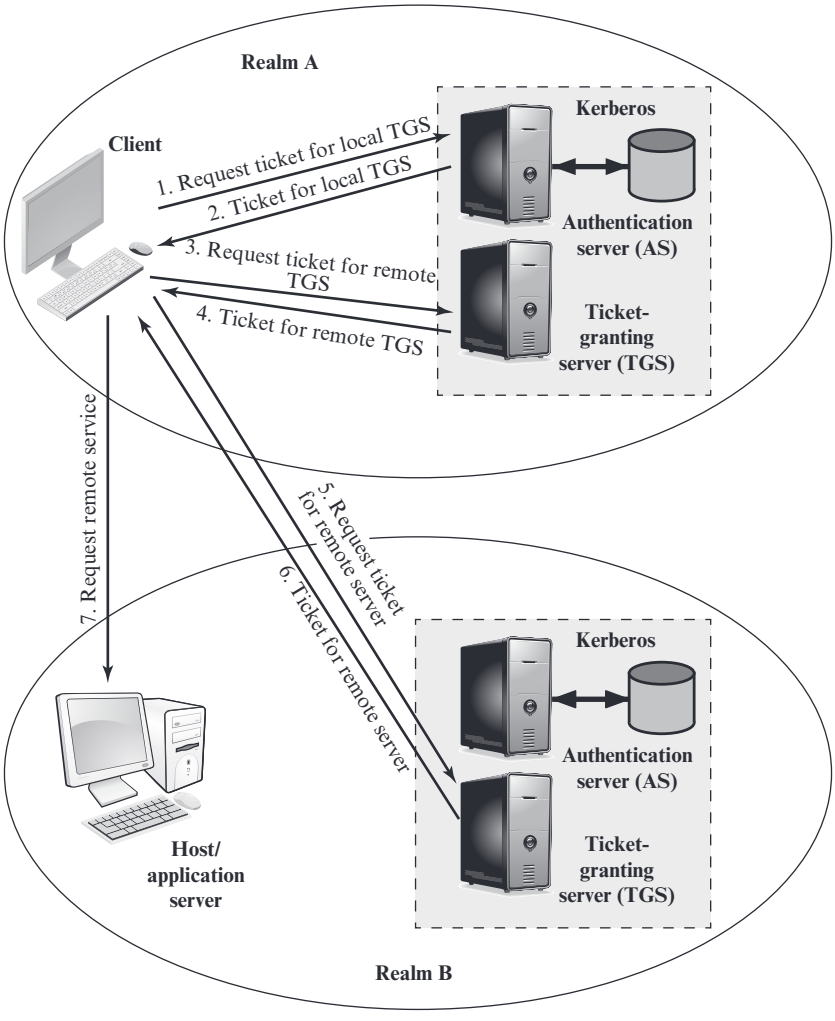


Figure 15.4 Request for Service in Another Realm

The ticket presented to the remote server (V_{rem}) indicates the realm in which the user was originally authenticated. The server chooses whether to honor the remote request.

One problem presented by the foregoing approach is that it does not scale well to many realms. If there are N realms, then there must be $N(N - 1)/2$ secure key exchanges so that each Kerberos realm can interoperate with all other Kerberos realms.

Kerberos Version 5

Kerberos version 5 is specified in RFC 4120 and provides a number of improvements over version 4 [KOHL94]. To begin, we provide an overview of the changes from version 4 to version 5 and then look at the version 5 protocol.

Differences between Versions 4 and 5 Version 5 is intended to address the limitations of version 4 in two areas: environmental shortcomings and technical deficiencies. Let us briefly summarize the improvements in each area.[8]

Kerberos version 4 was developed for use within the Project Athena environment and, accordingly, did not fully address the need to be of general purpose. This led to the following **environmental shortcomings**.

1. **Encryption system dependence:** Version 4 requires the use of DES. Export restriction on DES as well as doubts about the strength of DES were thus of concern. In version 5, ciphertext is tagged with an encryption-type identifier so that any encryption technique may be used. Encryption keys are tagged with a type and a length, allowing the same key to be used in different algorithms and allowing the specification of different variations on a given algorithm.
2. **Internet protocol dependence:** Version 4 requires the use of Internet Protocol (IP) addresses. Other address types, such as the ISO network address, are not accommodated. Version 5 network addresses are tagged with type and length, allowing any network address type to be used.
3. **Message byte ordering:** In version 4, the sender of a message employs a byte ordering of its own choosing and tags the message to indicate least significant byte in lowest address or most significant byte in lowest address. This techniques works but does not follow established conventions. In version 5, all message structures are defined using Abstract Syntax Notation One (ASN.1) and Basic Encoding Rules (BER), which provide an unambiguous byte ordering.
4. **Ticket lifetime:** Lifetime values in version 4 are encoded in an 8-bit quantity in units of five minutes. Thus, the maximum lifetime that can be expressed is $2^8 \times 5 = 1280$ minutes (a little over 21 hours). This may be inadequate for some applications (e.g., a long-running simulation that requires valid Kerberos credentials throughout execution). In version 5, tickets include an explicit start time and end time, allowing tickets with arbitrary lifetimes.
5. **Authentication forwarding:** Version 4 does not allow credentials issued to one client to be forwarded to some other host and used by some other client. This capability would enable a client to access a server and have that server access another server on behalf of the client. For example, a client issues a request to a print server that then accesses the client's file from a file server, using the client's credentials for access. Version 5 provides this capability.
6. **Interrealm authentication:** In version 4, interoperability among N realms requires on the order of N^2 Kerberos-to-Kerberos relationships, as described earlier. Version 5 supports a method that requires fewer relationships, as described shortly.

[8]The following discussion follows the presentation in [KOHL94].

Apart from these environmental limitations, there are **technical deficiencies** in the version 4 protocol itself. Most of these deficiencies were documented in [BELL90], and version 5 attempts to address these. The deficiencies are the following.

1. **Double encryption:** Note in Table 15.1 [messages (2) and (4)] that tickets provided to clients are encrypted twice—once with the secret key of the target server and then again with a secret key known to the client. The second encryption is not necessary and is computationally wasteful.
2. **PCBC encryption:** Encryption in version 4 makes use of a nonstandard mode of DES known as **propagating cipher block chaining (PCBC)**.[9] It has been demonstrated that this mode is vulnerable to an attack involving the interchange of ciphertext blocks [KOHL89]. PCBC was intended to provide an integrity check as part of the encryption operation. Version 5 provides explicit integrity mechanisms, allowing the standard CBC mode to be used for encryption. In particular, a checksum or hash code is attached to the message prior to encryption using CBC.
3. **Session keys:** Each ticket includes a session key that is used by the client to encrypt the authenticator sent to the service associated with that ticket. In addition, the session key may subsequently be used by the client and the server to protect messages passed during that session. However, because the same ticket may be used repeatedly to gain service from a particular server, there is the risk that an opponent will replay messages from an old session to the client or the server. In version 5, it is possible for a client and server to negotiate a subsession key, which is to be used only for that one connection. A new access by the client would result in the use of a new subsession key.
4. **Password attacks:** Both versions are vulnerable to a password attack. The message from the AS to the client includes material encrypted with a key based on the client's password.[10] An opponent can capture this message and attempt to decrypt it by trying various passwords. If the result of a test decryption is of the proper form, then the opponent has discovered the client's password and may subsequently use it to gain authentication credentials from Kerberos. This is the same type of password attack described in Chapter 21, with the same kinds of countermeasures being applicable. Version 5 does provide a mechanism known as preauthentication, which should make password attacks more difficult, but it does not prevent them.

The Version 5 Authentication Dialogue Table 15.3 summarizes the basic version 5 dialogue. This is best explained by comparison with version 4 (Table 15.1).

First, consider the **authentication service exchange**. Message (1) is a client request for a ticket-granting ticket. As before, it includes the ID of the user and the TGS. The following new elements are added:

[9]This is described in Appendix T.
[10]Appendix T describes the mapping of passwords to encryption keys.

Table 15.3 Summary of Kerberos Version 5 Message Exchanges

(1)	**C → AS**	$Options \| ID_c \| Realm_c \| ID_{tgs} \| Times \| Nonce_1$
(2)	**AS → C**	$Realm_C \| ID_C \| Ticket_{tgs} \| \mathrm{E}(K_c, [K_{c,tgs} \| Times \| Nonce_1 \| Realm_{tgs} \| ID_{tgs}])$
		$Ticket_{tgs} = \mathrm{E}(K_{tgs}, [Flags \| K_{c,tgs} \| Realm_c \| ID_C \| AD_C \| Times])$

(a) Authentication Service Exchange to obtain ticket-granting ticket

(3)	**C → TGS**	$Options \| ID_v \| Times \| Nonce_2 \| Ticket_{tgs} \| Authenticator_c$
(4)	**TGS → C**	$Realm_c \| ID_C \| Ticket_v \| \mathrm{E}(K_{c,tgs}, [K_{c,v} \| Times \| Nonce_2 \| Realm_v \| ID_v])$
		$Ticket_{tgs} = \mathrm{E}(K_{tgs}, [Flags \| K_{c,tgs} \| Realm_c \| ID_C \| AD_C \| Times])$
		$Ticket_v = \mathrm{E}(K_v, [Flags \| K_{c,v} \| Realm_c \| ID_C \| AD_C \| Times])$
		$Authenticator_c = \mathrm{E}(K_{c,tgs}, [ID_C \| Realm_c \| TS_1])$

(b) Ticket-Granting Service Exchange to obtain service-granting ticket

(5)	**C → V**	$Options \| Ticket_v \| Authenticator_c$
(6)	**V → C**	$\mathrm{E}_{K_{c,v}}[TS_2 \| Subkey \| Seq\ \#]$
		$Ticket_v = \mathrm{E}(K_v, [Flag \| K_{c,v} \| Realm_c \| ID_C \| AD_C \| Times])$
		$Authenticator_c = \mathrm{E}(K_{c,v}, [ID_C \| Relam_c \| TS_2 \| Subkey \| Seq\ \#])$

(c) Client/Server Authentication Exchange to obtain service

- **Realm:** Indicates realm of user
- **Options:** Used to request that certain flags be set in the returned ticket
- **Times:** Used by the client to request the following time settings in the ticket:
  - —**from**: the desired start time for the requested ticket
  - —**till**: the requested expiration time for the requested ticket
  - —**rtime**: requested renew-till time
- **Nonce:** A random value to be repeated in message (2) to assure that the response is fresh and has not been replayed by an opponent

Message (2) returns a ticket-granting ticket, identifying information for the client, and a block encrypted using the encryption key based on the user's password. This block includes the session key to be used between the client and the TGS, times specified in message (1), the nonce from message (1), and TGS identifying information. The ticket itself includes the session key, identifying information for the client, the requested time values, and flags that reflect the status of this ticket and the requested options. These flags introduce significant new functionality to version 5. For now, we defer a discussion of these flags and concentrate on the overall structure of the version 5 protocol.

Let us now compare the **ticket-granting service exchange** for versions 4 and 5. We see that message (3) for both versions includes an authenticator, a ticket, and the name of the requested service. In addition, version 5 includes requested times and options for the ticket and a nonce—all with functions similar to those of message (1). The authenticator itself is essentially the same as the one used in version 4.

Message (4) has the same structure as message (2). It returns a ticket plus information needed by the client, with the information encrypted using the session key now shared by the client and the TGS.

Finally, for the **client/server authentication exchange**, several new features appear in version 5. In message (5), the client may request as an option that mutual authentication is required. The authenticator includes several new fields:

- **Subkey:** The client's choice for an encryption key to be used to protect this specific application session. If this field is omitted, the session key from the ticket ($K_{c,v}$) is used.
- **Sequence number:** An optional field that specifies the starting sequence number to be used by the server for messages sent to the client during this session. Messages may be sequence numbered to detect replays.

If mutual authentication is required, the server responds with message (6). This message includes the timestamp from the authenticator. Note that in version 4, the timestamp was incremented by one. This is not necessary in version 5, because the nature of the format of messages is such that it is not possible for an opponent to create message (6) without knowledge of the appropriate encryption keys. The subkey field, if present, overrides the subkey field, if present, in message (5). The optional sequence number field specifies the starting sequence number to be used by the client.

Ticket Flags The flags field included in tickets in version 5 supports expanded functionality compared to that available in version 4. Table 15.4 summarizes the flags that may be included in a ticket.

Table 15.4 Kerberos Version 5 Flags

INITIAL	This ticket was issued using the AS protocol and not issued based on a ticket-granting ticket.
PRE-AUTHENT	During initial authentication, the client was authenticated by the KDC before a ticket was issued.
HW-AUTHENT	The protocol employed for initial authentication required the use of hardware expected to be possessed solely by the named client.
RENEWABLE	Tells TGS that this ticket can be used to obtain a replacement ticket that expires at a later date.
MAY-POSTDATE	Tells TGS that a postdated ticket may be issued based on this ticket-granting ticket.
POSTDATED	Indicates that this ticket has been postdated; the end server can check the authtime field to see when the original authentication occurred.
INVALID	This ticket is invalid and must be validated by the KDC before use.
PROXIABLE	Tells TGS that a new service-granting ticket with a different network address may be issued based on the presented ticket.
PROXY	Indicates that this ticket is a proxy.
FORWARDABLE	Tells TGS that a new ticket-granting ticket with a different network address may be issued based on this ticket-granting ticket.
FORWARDED	Indicates that this ticket has either been forwarded or was issued based on authentication involving a forwarded ticket-granting ticket.

The INITIAL flag indicates that this ticket was issued by the AS, not by the TGS. When a client requests a service-granting ticket from the TGS, it presents a ticket-granting ticket obtained from the AS. In version 4, this was the only way to obtain a service-granting ticket. Version 5 provides the additional capability that the client can get a service-granting ticket directly from the AS. The utility of this is as follows: A server, such as a password-changing server, may wish to know that the client's password was recently tested.

The PRE-AUTHENT flag, if set, indicates that when the AS received the initial request [message (1)], it authenticated the client before issuing a ticket. The exact form of this preauthentication is left unspecified. As an example, the MIT implementation of version 5 has encrypted timestamp preauthentication, enabled by default. When a user wants to get a ticket, it has to send to the AS a preauthentication block containing a random confounder, a version number, and a timestamp all encrypted in the client's password-based key. The AS decrypts the block and will not send a ticket-granting ticket back unless the timestamp in the preauthentication block is within the allowable time skew (time interval to account for clock drift and network delays). Another possibility is the use of a smart card that generates continually changing passwords that are included in the preauthenticated messages. The passwords generated by the card can be based on a user's password but be transformed by the card so that, in effect, arbitrary passwords are used. This prevents an attack based on easily guessed passwords. If a smart card or similar device was used, this is indicated by the HW-AUTHENT flag.

When a ticket has a long lifetime, there is the potential for it to be stolen and used by an opponent for a considerable period. If a short lifetime is used to lessen the threat, then overhead is involved in acquiring new tickets. In the case of a ticket-granting ticket, the client would either have to store the user's secret key, which is clearly risky, or repeatedly ask the user for a password. A compromise scheme is the use of renewable tickets. A ticket with the RENEWABLE flag set includes two expiration times: One for this specific ticket and one that is the latest permissible value for an expiration time. A client can have the ticket renewed by presenting it to the TGS with a requested new expiration time. If the new time is within the limit of the latest permissible value, the TGS can issue a new ticket with a new session time and a later specific expiration time. The advantage of this mechanism is that the TGS may refuse to renew a ticket reported as stolen.

A client may request that the AS provide a ticket-granting ticket with the MAY-POSTDATE flag set. The client can then use this ticket to request a ticket that is flagged as POSTDATED and INVALID from the TGS. Subsequently, the client may submit the postdated ticket for validation. This scheme can be useful for running a long batch job on a server that requires a ticket periodically. The client can obtain a number of tickets for this session at once, with spread out time values. All but the first ticket are initially invalid. When the execution reaches a point in time when a new ticket is required, the client can get the appropriate ticket validated. With this approach, the client does not have to repeatedly use its ticket-granting ticket to obtain a service-granting ticket.

In version 5, it is possible for a server to act as a proxy on behalf of a client, in effect adopting the credentials and privileges of the client to request a service from another server. If a client wishes to use this mechanism, it requests a ticket-granting

ticket with the PROXIABLE flag set. When this ticket is presented to the TGS, the TGS is permitted to issue a service-granting ticket with a different network address; this latter ticket will have its PROXY flag set. An application receiving such a ticket may accept it or require additional authentication to provide an audit trail.[11]

The proxy concept is a limited case of the more powerful forwarding procedure. If a ticket is set with the FORWARDABLE flag, a TGS can issue to the requestor a ticket-granting ticket with a different network address and the FORWARDED flag set. This ticket then can be presented to a remote TGS. This capability allows a client to gain access to a server on another realm without requiring that each Kerberos maintain a secret key with Kerberos servers in every other realm. For example, realms could be structured hierarchically. Then a client could walk up the tree to a common node and then back down to reach a target realm. Each step of the walk would involve forwarding a ticket-granting ticket to the next TGS in the path.

15.4 REMOTE USER-AUTHENTICATION USING ASYMMETRIC ENCRYPTION

Mutual Authentication

In Chapter 14, we presented one approach to the use of public-key encryption for the purpose of session-key distribution (Figure 14.9). This protocol assumes that each of the two parties is in possession of the current public key of the other. It may not be practical to require this assumption.

A protocol using timestamps is provided in [DENN81]:

1. A → AS: $ID_A \| ID_B$
2. AS → A: $E(PR_{as}, [ID_A \| PU_a \| T]) \| E(PR_{as}, [ID_B \| PU_b \| T])$
3. A → B: $E(PR_{as}, [ID_A \| PU_a \| T]) \| E(PR_{as}, [ID_B \| PU_b \| T]) \| E(PU_b, E(PR_a, [K_s \| T]))$

In this case, the central system is referred to as an authentication server (AS), because it is not actually responsible for secret-key distribution. Rather, the AS provides public-key certificates. The session key is chosen and encrypted by A; hence, there is no risk of exposure by the AS. The timestamps protect against replays of compromised keys.

This protocol is compact but, as before, requires the synchronization of clocks. Another approach, proposed by Woo and Lam [WOO92a], makes use of nonces. The protocol consists of the following steps.

1. A → KDC: $ID_A \| ID_B$
2. KDC → A: $E(PR_{\text{auth}}, [ID_B \| PU_b])$
3. A → B: $E(PU_b, [N_a \| ID_A])$
4. B → KDC: $ID_A \| ID_B \| E(PU_{\text{auth}}, N_a)$
5. KDC → B: $E(PR_{\text{auth}}, [ID_A \| PU_a]) \| E(PU_b, E(PR_{\text{auth}}, [N_a \| K_s \| ID_B]))$

[11] For a discussion of some of the possible uses of the proxy capability, see [NEUM93b].

6. $B \rightarrow A$: $E(PU_a, [E(PR_{auth}, [(N_a \| K_s \| ID_B)]) \| N_b])$
7. $A \rightarrow B$: $E(K_s, N_b)$

In step 1, A informs the KDC of its intention to establish a secure connection with B. The KDC returns to A a copy of B's public-key certificate (step 2). Using B's public key, A informs B of its desire to communicate and sends a nonce N_a (step 3). In step 4, B asks the KDC for A's public-key certificate and requests a session key; B includes A's nonce so that the KDC can stamp the session key with that nonce. The nonce is protected using the KDC's public key. In step 5, the KDC returns to B a copy of A's public-key certificate, plus the information $\{N_a, K_s, ID_B\}$. This information basically says that K_s is a secret key generated by the KDC on behalf of B and tied to N_a; the binding of K_s and N_a will assure A that K_s is fresh. This triple is encrypted using the KDC's private key to allow B to verify that the triple is in fact from the KDC. It is also encrypted using B's public key so that no other entity may use the triple in an attempt to establish a fraudulent connection with A. In step 6, the triple $\{N_a, K_s, ID_B\}$, still encrypted with the KDC's private key, is relayed to A, together with a nonce N_b generated by B. All the foregoing are encrypted using A's public key. A retrieves the session key K_s, uses it to encrypt N_b, and returns it to B. This last message assures B of A's knowledge of the session key.

This seems to be a secure protocol that takes into account the various attacks. However, the authors themselves spotted a flaw and submitted a revised version of the algorithm in [WOO92b]:

1. $A \rightarrow KDC$: $ID_A \| ID_B$
2. $KDC \rightarrow A$: $E(PR_{auth}, [ID_B \| PU_b])$
3. $A \rightarrow B$: $E(PU_b, [N_a \| ID_A])$
4. $B \rightarrow KDC$: $ID_A \| ID_B \| E(PU_{auth}, N_a)$
5. $KDC \rightarrow B$: $E(PR_{auth}, [ID_A \| PU_a]) \| E(PU_b, E(PR_{auth}, [N_a \| K_s \| ID_A \| ID_B]))$
6. $B \rightarrow A$: $E(PU_a, [N_b \| E(PR_{auth}, [N_a \| K_s \| ID_A \| ID_B])])$
7. $A \rightarrow B$: $E(K_s, N_b)$

The identifier of A, ID_A, is added to the set of items encrypted with the KDC's private key in steps 5 and 6. This binds the session key K_s to the identities of the two parties that will be engaged in the session. This inclusion of ID_A accounts for the fact that the nonce value N_a is considered unique only among all nonces generated by A, not among all nonces generated by all parties. Thus, it is the pair $\{ID_A, N_a\}$ that uniquely identifies the connection request of A.

In both this example and the protocols described earlier, protocols that appeared secure were revised after additional analysis. These examples highlight the difficulty of getting things right in the area of authentication.

One-Way Authentication

We have already presented public-key encryption approaches that are suited to electronic mail, including the straightforward encryption of the entire message for confidentiality (Figure 12.1b), authentication (Figure 12.1c), or both (Figure 12.1d). These approaches require that either the sender know the recipient's public key

(confidentiality), the recipient know the sender's public key (authentication), or both (confidentiality plus authentication). In addition, the public-key algorithm must be applied once or twice to what may be a long message.

If confidentiality is the primary concern, then the following may be more efficient:

$$A \rightarrow B: \quad E(PU_b, K_s) \| E(K_s, M)$$

In this case, the message is encrypted with a one-time secret key. A also encrypts this one-time key with B's public key. Only B will be able to use the corresponding private key to recover the one-time key and then use that key to decrypt the message. This scheme is more efficient than simply encrypting the entire message with B's public key.

If authentication is the primary concern, then a digital signature may suffice, as was illustrated in Figure 13.2:

$$A \rightarrow B: \quad M \| E(PR_a, H(M))$$

This method guarantees that A cannot later deny having sent the message. However, this technique is open to another kind of fraud. Bob composes a message to his boss Alice that contains an idea that will save the company money. He appends his digital signature and sends it into the email system. Eventually, the message will get delivered to Alice's mailbox. But suppose that Max has heard of Bob's idea and gains access to the mail queue before delivery. He finds Bob's message, strips off his signature, appends his, and requeues the message to be delivered to Alice. Max gets credit for Bob's idea.

To counter such a scheme, both the message and signature can be encrypted with the recipient's public key:

$$A \rightarrow B: \quad E(PU_b, [M \| E(PR_a, H(M))])$$

The latter two schemes require that B know A's public key and be convinced that it is timely. An effective way to provide this assurance is the digital certificate, described in Chapter 14. Now we have

$$A \rightarrow B: \quad M \| E(PR_a, H(M)) \| E(PR_{as}, [T \| ID_A \| PU_a])$$

In addition to the message, A sends B the signature encrypted with A's private key and A's certificate encrypted with the private key of the authentication server. The recipient of the message first uses the certificate to obtain the sender's public key and verify that it is authentic and then uses the public key to verify the message itself. If confidentiality is required, then the entire message can be encrypted with B's public key. Alternatively, the entire message can be encrypted with a one-time secret key; the secret key is also transmitted, encrypted with B's public key. This approach is explored in Chapter 19.

15.5 FEDERATED IDENTITY MANAGEMENT

Federated identity management is a relatively new concept dealing with the use of a common identity management scheme across multiple enterprises and numerous applications and supporting many thousands, even millions, of users. We begin our overview with a discussion of the concept of identity management and then examine federated identity management.

Identity Management

Identity management is a centralized, automated approach to provide enterprise-wide access to resources by employees and other authorized individuals. The focus of identity management is defining an identity for each user (human or process), associating attributes with the identity, and enforcing a means by which a user can verify identity. The central concept of an identity management system is the use of single sign-on (SSO).

SSO enables a user to access all network resources after a single authentication.

Typical services provided by a federated identity management system include the following:

- **Point of contact:** Includes authentication that a user corresponds to the user name provided, and management of user/server sessions.
- **SSO protocol services:** Provides a vendor-neutral security token service for supporting a single sign on to federated services.
- **Trust services:** Federation relationships require a trust relationship-based federation between business partners. A trust relationship is represented by the combination of the security tokens used to exchange information about a user, the cryptographic information used to protect these security tokens, and optionally the identity mapping rules applied to the information contained within this token.
- **Key services:** Management of keys and certificates.
- **Identity services:** services that provide the interface to local data stores, including user registries and databases, for identity-related information management.
- **Authorization:** Granting access to specific services and/or resources based on the authentication.
- **Provisioning:** Includes creating an account in each target system for the user, enrollment or registration of user in accounts, establishment of access rights or credentials to ensure the privacy and integrity of account data.
- **Management:** Services related to runtime configuration and deployment.

Note that Kerberos contains a number of the elements of an identity management system.

Figure 15.5 illustrates entities and data flows in a generic identity management architecture. A **principal** is an identity holder. Typically, this is a human user that seeks access to resources and services on the network. User devices, agent processes, and server systems may also function as principals. Principals authenticate themselves to an **identity provider**. The identity provider associates authentication information with a principal, as well as attributes and one or more identifiers.

Increasingly, digital identities incorporate attributes other than simply an identifier and authentication information (such as passwords and biometric information). An **attribute service** manages the creation and maintenance of such attributes. For example, a user needs to provide a shipping address each time an order is placed at a new Web merchant, and this information needs to be revised when the user moves. Identity management enables the user to provide this information once, so that it is maintained in a single place and released to data consumers in accordance with

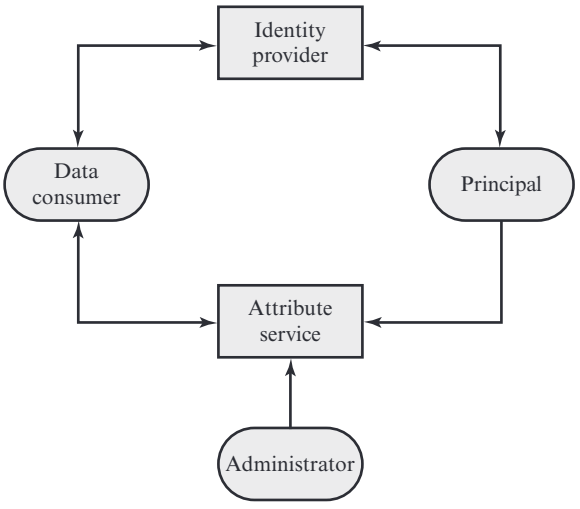


Figure 15.5 Generic Identity Management Architecture

authorization and privacy policies. Users may create some of the attributes to be associated with their digital identity, such as an address. **Administrators** may also assign attributes to users, such as roles, access permissions, and employee information.

Data consumers are entities that obtain and employ data maintained and provided by identity and attribute providers, which are often used to support authorization decisions and to collect audit information. For example, a database server or file server is a data consumer that needs a client's credentials so as to know what access to provide to that client.

Identity Federation

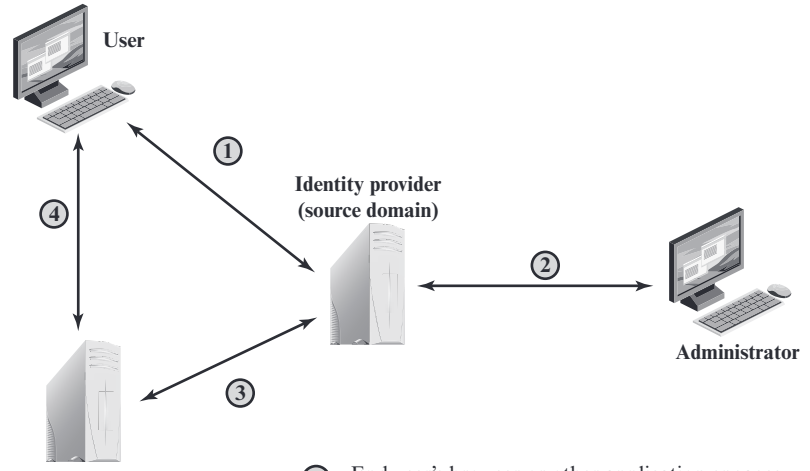
Identity federation is, in essence, an extension of identity management to multiple security domains. Such domains include autonomous internal business units, external business partners, and other third-party applications and services. The goal is to provide the sharing of digital identities so that a user can be authenticated a single time and then access applications and resources across multiple domains. Because these domains are relatively autonomous or independent, no centralized control is possible. Rather, the cooperating organizations must form a federation based on agreed standards and mutual levels of trust to securely share digital identities.

Federated identity management refers to the agreements, standards, and technologies that enable the portability of identities, identity attributes, and entitlements across multiple enterprises and numerous applications and supporting many thousands, even millions, of users. When multiple organizations implement interoperable federated identity schemes, an employee in one organization can use a single sign-on to access services across the federation with trust relationships associated with the identity. For example, an employee may log onto her corporate intranet and be authenticated to perform authorized functions and access authorized services on that intranet. The employee could then access their health benefits from an outside health-care provider without having to reauthenticate.

Beyond SSO, federated identity management provides other capabilities. One is a standardized means of representing attributes. Increasingly, digital identities incorporate attributes other than simply an identifier and authentication information (such as passwords and biometric information). Examples of attributes include account numbers, organizational roles, physical location, and file ownership. A user may have multiple identifiers; for example, each identifier may be associated with a unique role with its own access permissions.

Another key function of federated identity management is identity mapping. Different security domains may represent identities and attributes differently. Further, the amount of information associated with an individual in one domain may be more than is necessary in another domain. The federated identity management protocols map identities and attributes of a user in one domain to the requirements of another domain.

Figure 15.6 illustrates entities and data flows in a generic federated identity management architecture.



1. End user's browser or other application engages in an authentication dialogue with identity provider in the same domain. End user also provides attribute values associated with user's identity.
2. Some attributes associated with an identity, such as allowable roles, may be provided by an administrator in the same domain.
3. A service provider in a remote domain, which the user wishes to access, obtains identity information, authentication information, and associated attributes from the identity provider in the source domain.
4. Service provider opens session with remote user and enforces access control restrictions based on user's identity and attributes.

Figure 15.6 Federated Identity Operation

The identity provider acquires attribute information through dialogue and protocol exchanges with users and administrators. For example, a user needs to provide a shipping address each time an order is placed at a new Web merchant, and this information needs to be revised when the user moves. Identity management enables the user to provide this information once, so that it is maintained in a single place and released to data consumers in accordance with authorization and privacy policies.

Service providers are entities that obtain and employ data maintained and provided by identity providers, often to support authorization decisions and to collect audit information. For example, a database server or file server is a data consumer that needs a client's credentials so as to know what access to provide to that client. A service provider can be in the same domain as the user and the identity provider. The power of this approach is for federated identity management, in which the service provider is in a different domain (e.g., a vendor or supplier network).

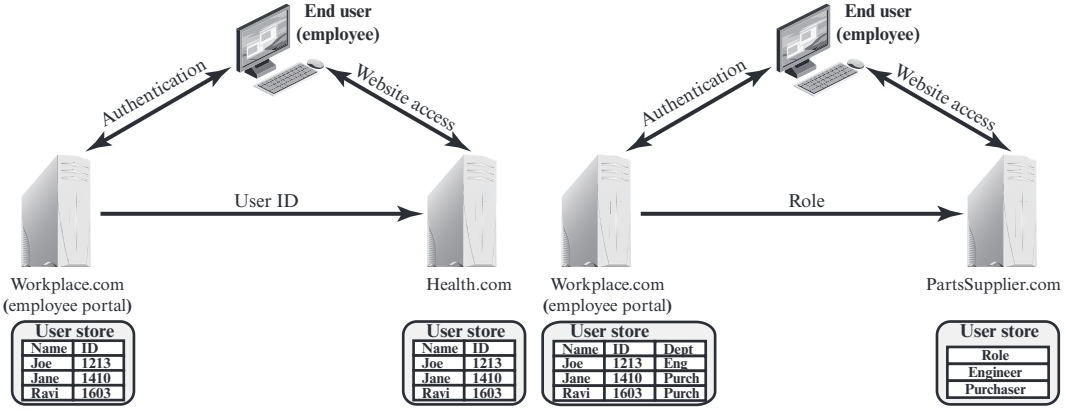
Standards Federated identity management uses a number of standards as the building blocks for secure identity exchange across different domains or heterogeneous systems. In essence, organizations issue some form of security tickets for their users that can be processed by cooperating partners. Identity federation standards are thus concerned with defining these tickets, in terms of content and format, providing protocols for exchanging tickets and performing a number of management tasks. These tasks include configuring systems to perform attribute transfers and identity mapping, and performing logging and auditing functions. The key standards are as follows:

- **The Extensible Markup Language (XML):** A markup language that uses sets of embedded tags or labels to characterize text elements within a document so as to indicate their appearance, function, meaning, or context. XML documents appear similar to HTML (Hypertext Markup Language) documents that are visible as Web pages, but provide greater functionality. XML includes strict definitions of the data type of each field, thus supporting database formats and semantics. XML provides encoding rules for commands that are used to transfer and update data objects.
- **The Simple Object Access Protocol (SOAP):** A minimal set of conventions for invoking code using XML over HTTP. It enables applications to request services from one another with XML-based requests and receive responses as data formatted with XML. Thus, XML defines data objects and structures, and SOAP provides a means of exchanging such data objects and performing remote procedure calls related to these objects. See [ROS06] for an informative discussion.
- **WS-Security:** A set of SOAP extensions for implementing message integrity and confidentiality in Web services. To provide for secure exchange of SOAP messages among applications, WS-Security assigns security tokens to each message for use in authentication.
- **Security Assertion Markup Language (SAML):** An XML-based language for the exchange of security information between online business partners. SAML conveys authentication information in the form of assertions about subjects. Assertions are statements about the subject issued by an authoritative entity.

The challenge with federated identity management is to integrate multiple technologies, standards, and services to provide a secure, user-friendly utility. The key, as in most areas of security and networking, is the reliance on a few mature standards widely accepted by industry. Federated identity management seems to have reached this level of maturity.

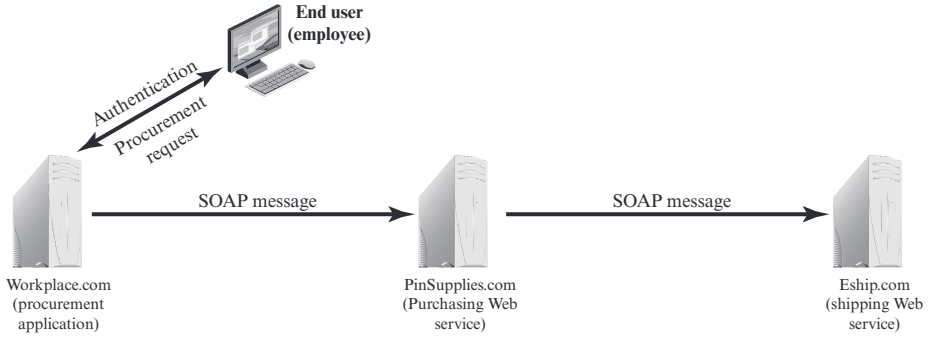
Examples To get some feel for the functionality of identity federation, we look at three scenarios, taken from [COMP06].

In the first scenario (Figure 15.7a), Workplace.com contracts with Health.com to provide employee health benefits. An employee uses a Web interface to sign on to Workplace.com and goes through an authentication procedure there. This enables the employee to access authorized services and resources at Workplace.com. When the employee clicks on a link to access health benefits, her browser is redirected to Health.com. At the same time, the Workplace.com software passes the user's identifier to Health.com in a secure manner. The two organizations are part of a federation that cooperatively exchanges user identifiers. Health.com maintains user identities



(a) Federation based on account linking

(b) Federation based on roles



(c) Chained Web services

Figure 15.7 Federated Identity Scenarios

for every employee at Workplace.com and associates with each identity health-benefits information and access rights. In this example, the linkage between the two companies is based on account information and user participation is browser based.

Figure 15.7b shows a second type of browser-based scheme. PartsSupplier.com is a regular supplier of parts to Workplace.com. In this case, a role-based access-control (RBAC) scheme is used for access to information. An engineer of Workplace.com authenticates at the employee portal at Workplace.com and clicks on a link to access information at PartsSupplier.com. Because the user is authenticated in the role of an engineer, he is taken to the technical documentation and troubleshooting portion of PartsSupplier.com's Web site without having to sign on. Similarly, an employee in a purchasing role signs on at Workplace.com and is authorized, in that role, to place purchases at PartsSupplier.com without having to authenticate to PartsSupplier.com. For this scenario, PartsSupplier.com does not have identity information for individual employees at Workplace.com. Rather, the linkage between the two federated partners is in terms of roles.

The scenario illustrated in Figure 15.7c can be referred to as document based rather than browser based. In this third example, Workplace.com has a purchasing agreement with PinSupplies.com, and PinSupplies.com has a business relationship with E-Ship.com. An employee of Workplace.com signs on and is authenticated to make purchases. The employee goes to a procurement application that provides a list of Workplace.com's suppliers and the parts that can be ordered. The user clicks on the PinSupplies button and is presented with a purchase order Web page (HTML page). The employee fills out the form and clicks the submit button. The procurement application generates an XML/SOAP document that it inserts into the envelope body of an XML-based message. The procurement application then inserts the user's credentials in the envelope header of the message, together with Workplace.com's organizational identity. The procurement application posts the message to the PinSupplies.com's purchasing Web service. This service authenticates the incoming message and processes the request. The purchasing Web service then sends a SOAP message to its shipping partner to fulfill the order. The message includes a PinSupplies.com security token in the envelope header and the list of items to be shipped as well as the end user's shipping information in the envelope body. The shipping Web service authenticates the request and processes the shipment order.

15.6 PERSONAL IDENTITY VERIFICATION

User authentication based on the possession of a smart card is becoming more widespread. A smart card has the appearance of a credit card, has an electronic interface, and may use a variety of authentication protocols.

A smart card contains within it an entire microprocessor, including processor, memory, and I/O ports. Some versions incorporate a special co-processing circuit for cryptographic operation to speed the task of encoding and decoding messages or generating digital signatures to validate the information transferred. In some cards, the I/O ports are directly accessible by a compatible reader by means of exposed electrical contacts. Other cards rely instead on an embedded antenna for wireless communication with the reader.

A typical smart card includes three types of memory. Read-only memory (ROM) stores data that does not change during the card's life, such as the card number and the cardholder's name. Electrically erasable programmable ROM (EEPROM) holds application data and programs, such as the protocols that the card can execute. It also holds data that may vary with time. For example, in a telephone card, the EEPROM holds the talk time remaining. Random access memory (RAM) holds temporary data generated when applications are executed.

For the practical application of smart card authentication, a wide range of vendors must conform to standards that cover smart card protocols, authentication and access control formats and protocols, database entries, message formats, and so on. An important step in this direction is FIPS 201-2 (*Personal Identity Verification [PIV] of Federal Employees and Contractors*, June 2012). The standard defines a reliable, government-wide PIV system for use in applications such as access to federally controlled facilities and information systems. The standard specifies a PIV system within which common identification credentials can be created and later used to verify a claimed identity. The standard also identifies Federal government-wide requirements for security levels that are dependent on risks to the facility or information being protected. The standard applies to private-sector contractors as well, and serves as a useful guideline for any organization.

PIV System Model

Figure 15.8 illustrates the major components of FIPS 201-2 compliant systems. The PIV front end defines the physical interface to a user who is requesting access to a facility, which could be either physical access to a protected physical area or logical access to an information system. The **PIV front-end subsystem** supports up to three-factor authentication; the number of factors used depends on the level of security required. The front end makes use of a smart card, known as a PIV card, which is a dual-interface contact and contactless card. The card holds a cardholder photograph, X.509 certificates, cryptographic keys, biometric data, and a cardholder unique identifier (CHUID). Certain cardholder information may be read-protected and require a personal identification number (PIN) for read access by the card reader. The biometric reader, in the current version of the standard, is a fingerprint reader or an iris scanner.

The standard defines three assurance levels for verification of the card and the encoded data stored on the card, which in turn leads to verifying the authenticity of the person holding the credential. A level of *some confidence* corresponds to use of the card reader and PIN. A level of *high confidence* adds a biometric comparison of a fingerprint captured and encoded on the card during the card-issuing process and a fingerprint scanned at the physical access point. A *very high confidence* level requires that the process just described is completed at a control point attended by an official observer.

The other major component of the PIV system is the **PIV card issuance and management subsystem**. This subsystem includes the components responsible for identity proofing and registration, card and key issuance and management, and the various repositories and services (e.g., public key infrastructure [PKI] directory, certificate status servers) required as part of the verification infrastructure.

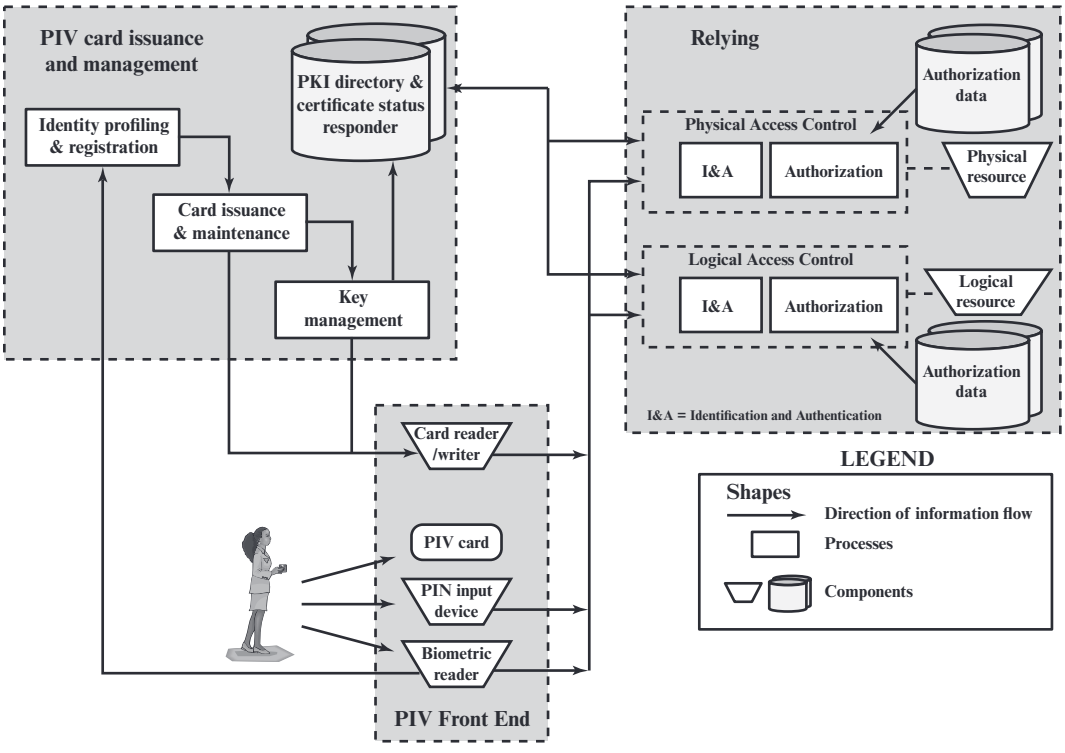


Figure 15.8 FIPS 201 PIV System Model

The PIV system interacts with a **relying subsystem**, which includes components responsible for determining a particular PIV cardholder's access to a physical or logical resource. FIPS 201-2 standardizes data formats and protocols for interaction between the PIV system and the relying system.

Unlike the typical card number/facility code encoded on most access control cards, the FIPS 201 CHUID takes authentication to a new level, through the use of an expiration date (a required CHUID data field) and an optional CHUID digital signature. A digital signature can be checked to ensure that the CHUID recorded on the card was digitally signed by a trusted source and that the CHUID data have not been altered since the card was signed. The CHUID expiration date can be checked to verify that the card has not expired. This is independent from whatever expiration date is associated with cardholder privileges. Reading and verifying the CHUID alone provides only some assurance of identity because it authenticates the card data, not the cardholder. The PIN and biometric factors provide identity verification of the individual.

PIV Documentation

The PIV specification is quite complex, and NIST has issued a number of documents that cover a broad range of PIV topics. These are as follows:

- **FIPS 201-2—Personal Identity Verification (PIV) of Federal Employees and Contractors:** Specifies the physical card characteristics, storage media, and data elements that make up the identity credentials resident on the PIV card.
- **SP 800-73-3—Interfaces for Personal Identity Verification:** Specifies the interfaces and card architecture for storing and retrieving identity credentials from a smart card, and provides guidelines for the use of authentication mechanisms and protocols.
- **SP 800-76-2—Biometric Data Specification for Personal Identity Verification:** Describes technical acquisition and formatting specifications for the biometric credentials of the PIV system.
- **SP 800-78-3—Cryptographic Algorithms and Key Sizes for Personal Identity Verification:** Identifies acceptable symmetric and asymmetric encryption algorithms, digital signature algorithms, and message digest algorithms, and specifies mechanisms to identify the algorithms associated with PIV keys or digital signatures.
- **SP 800-104—A Scheme for PIV Visual Card Topography:** Provides additional recommendations on the PIV card color-coding for designating employee affiliation.
- **SP 800-116—A Recommendation for the Use of PIV Credentials in Physical Access Control Systems (PACS):** Describes a risk-based approach for selecting appropriate PIV authentication mechanisms to manage physical access to Federal government facilities and assets.
- **SP 800-79-1—Guidelines for the Accreditation of Personal Identity Verification Card Issuers:** Provides guidelines for accrediting the reliability of issuers of PIV cards that collect, store, and disseminate personal identity credentials and issue smart cards.
- **SP 800-96—PIV Card to Reader Interoperability Guidelines:** Provides requirements that facilitate interoperability between any card and any reader.

In addition there are other documents that deal with conformance testing and codes for identifiers.

PIV Credentials and Keys

The PIV card contains a number of mandatory and optional data elements that serve as identity credentials with varying levels of strength and assurance. These credentials are used singly or in sets to authenticate the holder of the PIV card to achieve the level of assurance required for a particular activity or transaction. The mandatory data elements are the following:

- **Personal Identification Number (PIN):** Required to activate the card for privileged operation.
- **Cardholder Unique Identifier (CHUID):** Includes the Federal Agency Smart Credential Number (FASC-N) and the Global Unique Identification Number (GUID), which uniquely identify the card and the cardholder.

- **PIV Authentication Key:** Asymmetric key pair and corresponding certificate for user authentication.
- **Two fingerprint templates:** For biometric authentication.
- **Electronic facial image:** For biometric authentication.
- **Asymmetric Card Authentication Key:** Asymmetric key pair and corresponding certificate used for card authentication.

Optional elements include the following:

- **Digital Signature Key:** Asymmetric key pair and corresponding certificate that supports document signing and signing of data elements such as the CHUID.
- **Key Management Key:** Asymmetric key pair and corresponding certificate supporting key establishment and transport.
- **Symmetric Card Authentication Key:** For supporting physical access applications.
- **PIV Card Application Administration Key:** Symmetric key associated with the card management system.
- **One or two iris images:** For biometric authentication.

Table 15.5 lists the algorithm and key size requirements for PIV key types.

Authentication

Using the electronic credentials resident on a PIV card, the card supports the following authentication mechanisms:

- **CHUID:** The cardholder is authenticated using the signed CHUID data element on the card. The PIN is not required. This mechanism is useful in environments where a low level of assurance is acceptable and rapid contactless authentication is necessary.

Table 15.5 PIV Algorithms and Key Sizes

PIV Key Type	Algorithms	Key Sizes (bits)	Application
PIV Authentication Key	RSA	2048	Supports card and cardholder authentication for an interoperable environment
	ECDSA	256	
Card Authentication Key	3TDEA	168	Supports card authentication for physical access
	AES	128, 192, or 256	
	RSA	2048	Supports card authentication for an interoperable environment
	ECDSA	256	
Digital Signature Key	RSA	2048 or 3072	Supports document signing and nonce signing
	ECDSA	256 or 384	
Key Management Key	RSA	2048	Supports key establishment and transport
	ECDH	256 or 384	

- **Card Authentication Key:** The PIV card is authenticated using the Card Authentication Key in a challenge response protocol. The PIN is not required. This mechanism allows contact (via card reader) or contactless (via radio waves) authentication of the PIV card without the holder's active participation, and provides a low level of assurance.
- **BIO:** The cardholder is authenticated by matching his or her fingerprint sample(s) to the signed biometric data element in an environment without a human attendant in view. The PIN is required to activate the card. This mechanism achieves a high level of assurance and requires the cardholder's active participation is submitting the PIN as well as the biometric sample.
- **BIO-A:** The cardholder is authenticated by matching his or her fingerprint sample(s) to the signed biometric data element in an environment with a human attendant in view. The PIN is required to activate the card. This mechanism achieves a very high level of assurance when coupled with full trust validation of the biometric template retrieved from the card, and requires the cardholder's active participation is submitting the PIN as well as the biometric sample.
- **PKI:** The cardholder is authenticated by demonstrating control of the PIV authentication private key in a challenge response protocol that can be validated using the PIV authentication certificate. The PIN is required to activate the card. This mechanism achieves a very high level of identity assurance and requires the cardholder's knowledge of the PIN.

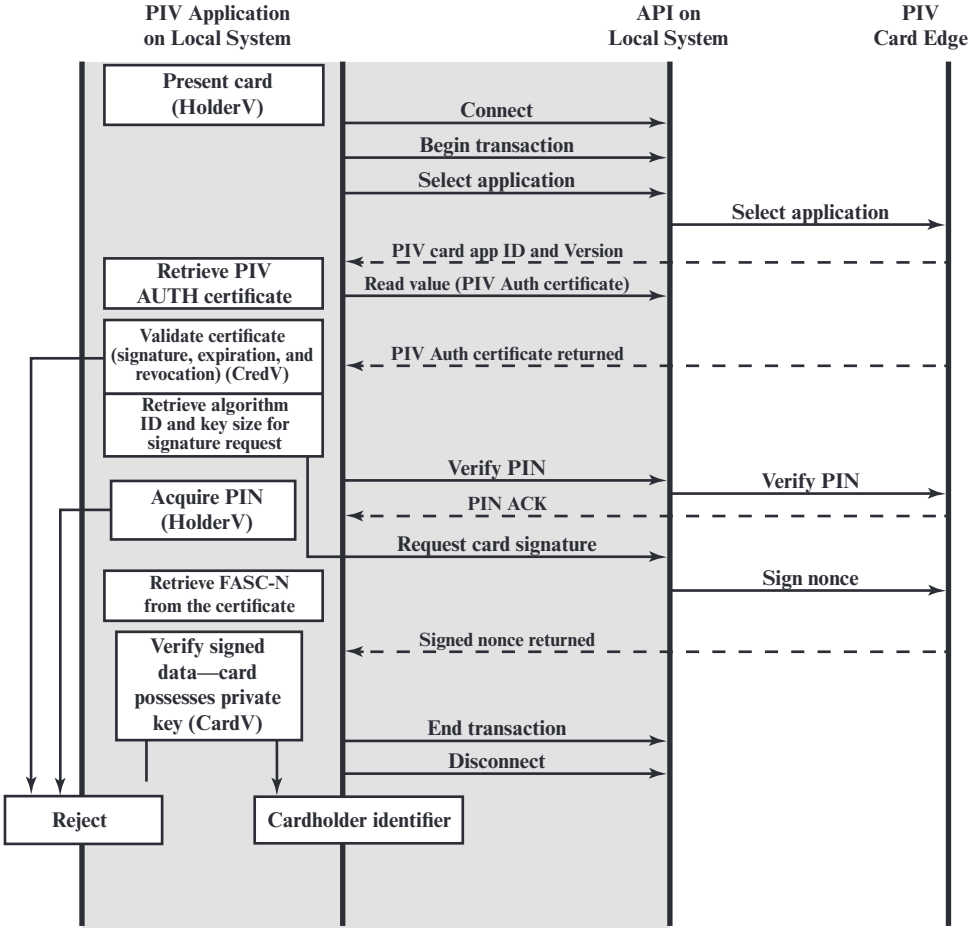
In each of the above use cases, except the symmetric Card Authentication Key use case, the source and the integrity of the corresponding PIV credential are validated by verifying the digital signature on the credential, with the signature being provided by a trusted entity.

A variety of protocols can be constructed for each of these authentication types. SP 800-78-3 gives examples for each type. Figure 15.9 illustrates an authentication scenario that includes the use of the PIV Authentication Key. This scenario provides a high level of assurance. This scenario would be appropriate for authentication of a user who possesses a PIV card and seeks access to a computer resource. The computer, designated *local system* in the figure, includes PIV application software and communicates to the card via an application program interface that enables the use of relatively high-level procedure calls. These high-level commands are converted into PIV commands that are issued to the card through a physical interface via a card reader or via a wireless interface. In either case, SP 800-73 refers to the card command interface as the PIV card edge.

The process begins when the local system detects the card either through an attached card reader or wirelessly. It then selects an application on the card for authentication. The local system then requests the public-key certificate for the card's PIV Authentication Key. If the certificate is valid (i.e., has a valid signature, has not expired or been revoked), authentication continues. Otherwise the card is rejected. The next step is for the local system to request that the cardholder enter the PIN for the card. If the submitted PIN matches the PIN stored on the card, the card returns a positive acknowledgment; otherwise the card returns a failure message.

The local system either continues or rejects the card accordingly. The next phase is a challenge-response protocol. The local system sends a nonce to be signed by the PIV, and the PIV returns the signature. The local system uses the PIV authentication public key to verify the signature. If the signature is valid, the cardholder is accepted as being identified. Otherwise the local system rejects the card.

The scenario of Figure 15.9 accomplishes three types of authentication. The combination of possession of the card and knowledge of the PIN service authenticates the cardholder. The PIV Authentication Key certificate validates the card's credentials. The challenge-response protocol authenticates the card.



CardV = Card validation
CredV = Credential validation
HolderV = Cardholder validation
FASC-N = Federal Agency Smart Credential Number

Figure 15.9 Authentication Using PIV Authentication Key

15.7 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

authentication	Kerberos	relying party (RP)
authentication server	Kerberos realm	replay attack
claimant	mutual authentication	subscriber
credential	nonce	suppress-replay attack
credential service provider (CSP)	one-way authentication	ticket
federated identity management	personal identity verification (PIV)	ticket-granting server (TGS)
identity management	realm	timestamp
	registration authority (RA)	verifier

Review Questions

15.1 What are the steps involved in an authentication process?
15.2 List three general approaches to dealing with replay attacks.
15.3 What is a suppress-replay attack?
15.4 What problem was Kerberos designed to address?
15.5 What are three threats associated with user authentication over a network or Internet?
15.6 List three approaches to secure user authentication in a distributed environment.
15.7 What four requirements were defined for Kerberos?
15.8 What entities constitute a full-service Kerberos environment?
15.9 In the context of Kerberos, what is a realm?
15.10 What are the mandatory elements to authenticate a PIV card holder?

Problems

15.1 In Section 15.4, we outlined the public-key scheme proposed in [WOO92a] for the distribution of secret keys. The revised version includes ID_A in steps 5 and 6. What attack, specifically, is countered by this revision?

15.2 The protocol referred to in Problem 15.1 can be reduced from seven steps to five, having the following sequence:
- **a.** A → B:
- **b.** A → KDC:
- **c.** KDC → B:
- **d.** B → A:
- **e.** A → B:

Show the message transmitted at each step. *Hint:* The final message in this protocol is the same as the final message in the original protocol.

15.3 Reference the suppress-replay attack described in Section 15.2 to answer the following.
- **a.** Give an example of an attack when a party's clock is ahead of that of the KDC.
- **b.** Give an example of an attack when a party's clock is ahead of that of another party.

15.4 There are three typical ways to use nonces as challenges. Suppose N_a is a nonce generated by A, A and B share key K, and f() is a function (such as an increment). The three usages are

Usage 1	Usage 2	Usage 3
(1) A → B: N_a	(1) A → B: E(K, N_a)	(1) A → B: E(K, N_a)
(2) B → A: E(K, N_a)	(2) B → A: N_a	(2) B → A: E(K, f(N_a))

Describe situations for which each usage is appropriate.

15.5 Show that a random error in one block of ciphertext is propagated to all subsequent blocks of plaintext in PCBC mode (See Figure T.2 in Appendix T).

15.6 Suppose that, in PCBC mode, blocks C_i and C_{i+1} are interchanged during transmission. Show that this affects only the decrypted blocks P_i and P_{i+1} but not subsequent blocks.

15.7 In addition to providing a standard for public-key certificate formats, X.509 specifies an authentication protocol. The original version of X.509 contains a security flaw. The essence of the protocol is as follows.

$$\begin{aligned} &\text{A} \rightarrow \text{B:} \quad \text{A}\ \{t_A, r_A, ID_B\} \\ &\text{B} \rightarrow \text{A:} \quad \text{B}\ \{t_B, r_B, ID_A, r_A\} \\ &\text{A} \rightarrow \text{B:} \quad \text{A}\ \{r_B\} \end{aligned}$$

where t_A and t_B are timestamps, r_A and r_B are nonces and the notation X{Y} indicates that the message Y is transmitted, encrypted, and signed by X.

The text of X.509 states that checking timestamps t_A and t_B is optional for three-way authentication. But consider the following example: Suppose A and B have used the preceding protocol on some previous occasion, and that opponent C has intercepted the preceding three messages. In addition, suppose that timestamps are not used and are all set to 0. Finally, suppose C wishes to impersonate A to B. C initially sends the first captured message to B:

$$\text{C} \rightarrow \text{B:} \quad \text{A}\ \{0, r_A, ID_B\}$$

B responds, thinking it is talking to A but is actually talking to C:

$$\text{B} \rightarrow \text{C:} \quad \text{B}\ \{0, r'_B, ID_A, r_A\}$$

C meanwhile causes A to initiate authentication with C by some means. As a result, A sends C the following:

$$\text{A} \rightarrow \text{C:} \quad \text{A}\ \{0, r'_A, ID_C\}$$

C responds to A using the same nonce provided to C by B:

$$\text{C} \rightarrow \text{A:} \quad \text{C}\ \{0, r'_B, ID_A, r'_A\}$$

A responds with

$$\text{A} \rightarrow \text{C:} \quad \text{A}\ \{r'_B\}$$

This is exactly what C needs to convince B that it is talking to A, so C now repeats the incoming message back out to B.

$$\text{C} \rightarrow \text{B:} \quad \text{A} \{r'_B\}$$

So B will believe it is talking to A whereas it is actually talking to C. Suggest a simple solution to this problem that does not involve the use of timestamps.

15.8 Consider a one-way authentication technique based on asymmetric encryption:

$$\text{A} \rightarrow \text{B:} \quad ID_A$$
$$\text{B} \rightarrow \text{A:} \quad R_1$$
$$\text{A} \rightarrow \text{B:} \quad \text{E}(PR_a, R_1)$$

a. Explain the protocol.
b. What type of attack is this protocol susceptible to?

15.9 Consider a one-way authentication technique based on asymmetric encryption:

$$A \rightarrow B\text{: } ID_A \| E(PU_B, R_A)$$

$$B \rightarrow A\text{: } R_A$$

a. Explain the protocol.
b. What type of attack is this protocol susceptible to?

15.10 In Kerberos, when Bob receives a Ticket from Alice, how does he know it is not genuine?

15.11 In Kerberos, how does Bob know that the received token is not corresponding to Alice's?

15.12 In Kerberos, how does Alice know that a reply to an earlier message is from Bob?

15.13 In Kerberos, what does the Ticket contain that allows Alice and Bob to talk securely?