CHAPTER 22

Internet Authentication Applications

This chapter examines some of the authentication functions that have been developed to support network-baed authentication and digital signatures.

We begin by looking at one of the earliest and also one of the most widely used services: Kerberos. Next, we examine the X.509 directory authentication service. This standard is important as part of the directory service that it supports but is also a basic building block used in other standards, such as S/MIME, discussed in Chapter 21. Next, we examine the concept of a public-key infrastructure (PKI). Finally, we introduce the concept of federated identity management

22.1 KERBEROS

There are a number of approaches that organizations can use to secure networked servers and hosts. Systems that use one-time passwords thwart any attempt to guess or capture a user's password. These systems require special equipment such as smart cards or synchronized password generators to operate and have been slow to gain acceptance for general networking use. Another approach is the use of biometric systems. These are automated methods of verifying or recognizing identity on the basis some physiological characteristic, such as a fingerprint or iris pattern, or a behavioral characteristic, such as handwriting or keystroke patterns. Again, these systems require specialized equipment.

Another way to tackle the problem is the use of authentication software tied to a secure authentication server. This is the approach taken by Kerberos. Kerberos, initially developed at MIT, is a software utility available both in the public domain and in commercially supported versions. Kerberos has been issued as an Internet standard and is the defacto standard for remote authentication.

The overall scheme of Kerberos is that of a trusted third-party authentication service. It is trusted in the sense that clients and servers trust Kerberos to mediate their mutual authentication. In essence, Kerberos requires that a user prove his or her identity for each service invoked and, optionally, requires servers to prove their identity to clients.

The Kerberos Protocol

Kerberos makes use of a protocol that involves clients, application servers, and a Kerberos server. That the protocol is complex reflects that fact that there are many ways for an opponent to penetrate security. Kerberos is designed to counter a variety of threats to the security of a client/server dialogue.

The basic idea is simple. 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. An alternative is to use an authentication server (AS) that knows the passwords of all users and stores these in a centralized database. Then the user can log onto the AS for identity verification. Once the AS has verified the user's identity, it can pass this information on to an application server, which will then accept service requests from the client.

The trick is how to do all this in a secure way. It simply won't do to have the client send the user's password to the AS over the network: An opponent could observe the password on the network and later reuse it. It also won't do for Kerberos to send a plain message to a server validating a client: An opponent could impersonate the AS and send a false validation.

The way around this problem is to use encryption and a set of messages that accomplish the task (Figure 22.1). In the case of Kerberos, the Data Encryption Standard (DES) is the encryption algorithm that is used.

The AS shares a unique secret key with each server. These keys have been distributed physically or in some other secure manner. This will enable the AS to send messages to application servers in a secure fashion. To begin, user X logs on to a workstation and requests access to server V. The client sends a message to the AS that includes the user's ID and a request for what is known as a ticket-granting ticket (TGT). The AS checks its database to find the password of this user. Then the AS responds with a TGT and a one-time encryption key, known as a session key,

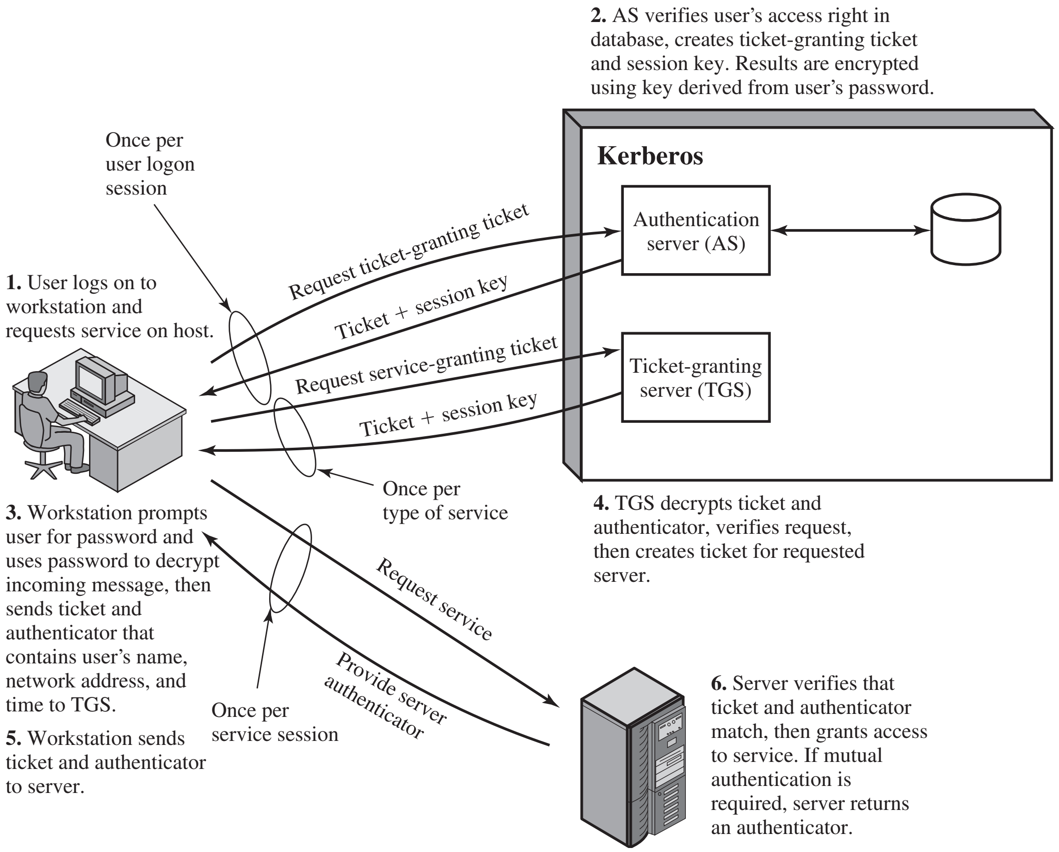
2. AS verifies user's access right in database, creates ticket-granting ticket and session key. Results are encrypted using key derived from user's password.
Kerberos
Authentication server (AS)
Ticket-granting server (TGS)
Once per user logon session
1. User logs on to workstation and requests service on host.
Request ticket-granting ticket
Ticket + session key
Request service-granting ticket
Ticket + session key
Once per type of service
4. TGS decrypts ticket and authenticator, verifies request, then creates ticket for requested server.
3. Workstation prompts user for password and uses password to decrypt incoming message, then sends ticket and authenticator that contains user's name, network address, and time to TGS.
5. Workstation sends ticket and authenticator to server.
Once per service session
Request service
Provide server authenticator
6. Server verifies that ticket and authenticator match, then grants access to service. If mutual authentication is required, server returns an authenticator.

Figure 22.1 **Overview of Kerberos**

both encrypted using the user's password as the encryption key. When this message arrives back 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 has been supplied, the ticket and session key are successfully recovered.

Notice what has happened. The AS has been able to verify the user's identity since this user knows the correct password, but it has been done in such a way that the password is never passed over the network. In addition, the AS has passed information to the client that will be used later on to apply to a server for service, and that information is secure since it is encrypted with the user's password.

The ticket constitutes a set of credentials that can be used by the client to apply for service. The ticket indicates that the AS has accepted this client and its user. The ticket contains the user's ID, the server's ID, a timestamp, a lifetime after which the ticket is invalid, and a copy of the same session key sent in the outer message to the client. The entire ticket is encrypted using a secret DES key shared by the AS and the server. Thus, no one can tamper with the ticket.

Now, Kerberos could have been set up so that the AS would send back a ticket granting access to a particular application server. This would require the client to request a new ticket from the AS for each service that the user wants to use during a logon session, which would in turn require that the AS query the user for his or her password for each service request or else to store the password in memory for the duration of the logon session. The first course is inconvenient for the user and the second course is a security risk. Therefore, the AS supplies a ticket good not for a specific application service but for a special ticket-granting service (TGS). The AS gives the client a ticket that can be used to get more tickets!

The idea is that this ticket can be used by the client 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 ticket and waits until the user has logged off the workstation. Then the opponent either gains access to that workstation or configures his workstation with the same network address as that of the victim. Then 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., 8 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.

Let's see how this works. The user has requested access to server V. The client has obtained a ticket-granting ticket and a temporary session key. The client then sends a message to the TGS requesting a ticket for user X that will grant service to server V. The message includes the ID of server V and the ticket-granting ticket. The TGS decrypts the incoming ticket (remember, the ticket is encrypted by a key known only to the AS and the TGS) and verifies the success of the decryption by the presence of its own 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.

At this point, the TGS is almost ready to grant a service-granting ticket to the client. But there is one more threat to overcome. The heart of the 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 a message to the TGS. This would give the opponent unlimited access to the resources and files available to the legitimate user.

To get around this problem, the AS has provided both the client and the TGS with a secret session key that they now share. The session key, recall, was in the message from the AS to the client, encrypted with the user's password. It was also buried in the ticket-granting ticket, encrypted with the key shared by the AS and TGS. In the message to the TGS requesting a service-granting ticket, the client includes an authenticator encrypted with the session key, which contains the ID and address of the user and a timestamp. Unlike the ticket, which is reusable, the authenticator is intended for use only once and has a very short lifetime. Now, TGS can decrypt the ticket with the key that it shares with the AS. This ticket indicates that user X has been provided with the session key. In effect, the ticket says, "Anyone who uses this session key must be X." 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 the time of this authenticator, I hereby use this session key." Note that the ticket doesn't 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. Later, if the client wants to apply to the TGS for a new service-granting ticket, it sends the reusable ticket-granting ticket plus a fresh authenticator.

The next two steps in the protocol repeat the last two. The TGS sends a service-granting ticket and a new session key to the client. The entire message is encrypted with the old session key, so that only the client can recover the message. The ticket is encrypted with a secret key shared only by the TGS and server V. The client now has a reusable service-granting ticket for V.

Each time user X wishes to use service V, the client can then send this ticket plus an authenticator to server V. The authenticator is encrypted with the new session key.

If mutual authentication is required, the server can reply with the value of the timestamp from the authenticator, incremented by 1, and encrypted in the session key. The client can decrypt this message to recover the incremented timestamp. Because the message was encrypted by the session key, the client is assured that it could have been created only by V. The contents of the message assures 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 session key for that purpose.

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 password 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.

Such an environment is referred to as a realm. Networks of clients and servers under different administrative organizations generally constitute different realms (Figure 22.2). 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

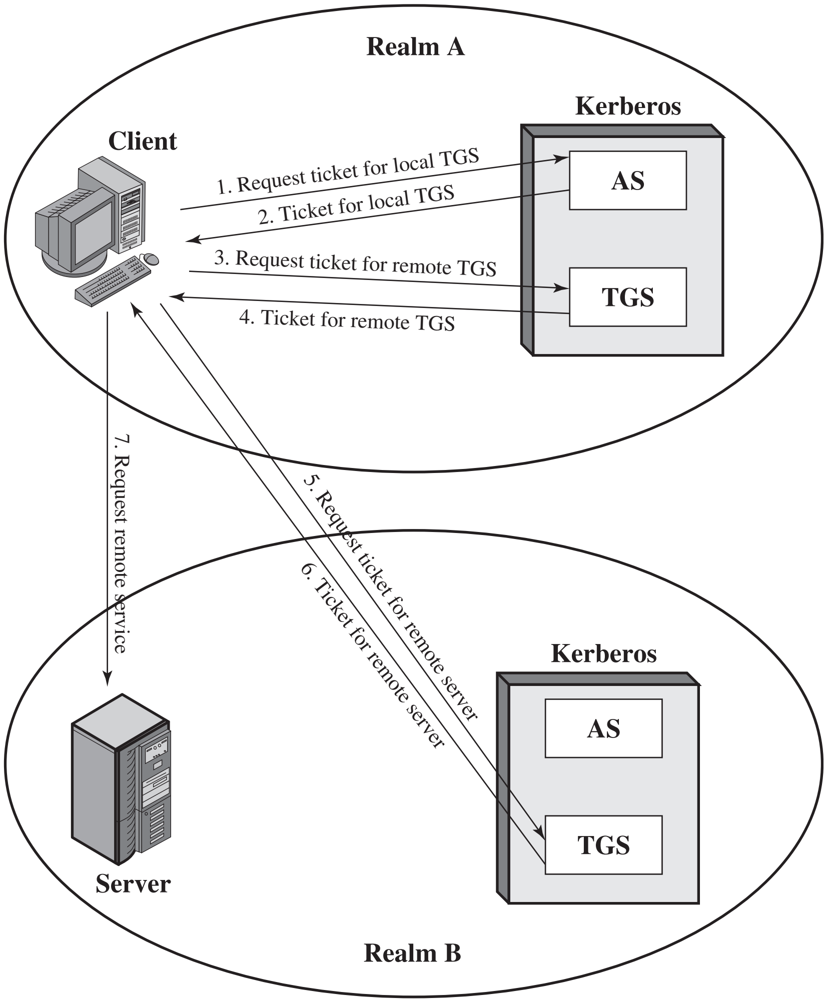
Realm A
Kerberos
Client
1. Request ticket for local TGS
AS
2. Ticket for local TGS
3. Request ticket for remote TGS
TGS
4. Ticket for remote TGS
7. Request remote service
5. Request ticket for remote server
6. Ticket for remote server
Kerberos
AS
TGS
Server
Realm B

Figure 22.2 **Request for Service in Another Realm**

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, 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 22.2): 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 ticket presented to the remote server 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 $N(N -)/2$ secure key exchanges so that each realm can interoperate with all other Kerberos realms.

Version 4 and Version 5

The most widely used version of Kerberos is version 4, which has been around for several years. More recently, a version 5 has been introduced. The most important improvements found in version 5 are the following. First, in version 5, an encrypted message is tagged with an encryption algorithm identifier. This enables users to configure Kerberos to use an algorithm other than DES. Recently, there has been some concern about the strength of DES, and version 5 gives the user the option of another algorithm.

Version 5 also supports a technique known as 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.

Finally, Version 5 supports a method for interrealm authentication that requires fewer secure key exchanges than in version 4.

Performance Issues

As client/server applications become more popular, larger and larger client/server installations are appearing. A case can be made that the larger the scale of the networking environment, the more important it is to have logon authentication. But the question arises: What impact does Kerberos have on performance in a large-scale environment?

Fortunately, the answer is that there is very little performance impact if the system is properly configured. Keep in mind that tickets are reusable. Therefore, the amount of traffic needed for the granting ticket requests is modest. With respect to the transfer of a ticket for logon authentication, the logon exchange must take place anyway, so again the extra overhead is modest.

A related issue is whether the Kerberos server application requires a dedicated platform or can share a computer with other applications. It probably is not wise to run the Kerberos server on the same machine as a resource-intensive application such as a database server. Moreover, the security of Kerberos is best assured by placing the Kerberos server on a separate, isolated machine.

Finally, in a large system, is it necessary to go to multiple realms in order to maintain performance? Probably not. Rather, the motivation for multiple realms is administrative. If you have geographically separate clusters of machines, each with its own network administrator, then one realm per administrator may be convenient. However, this is not always the case.

22.2 X.509

Public-key certificates are referred to briefly in Section 2.4. Recall that, in essence, a certificate consists of a public key plus a User ID of the key owner, with the whole block signed by a trusted third party. Typically, the third party is a **certificate authority** (CA) that is trusted by the user community, such as a government agency or a financial institution. A user can present his or her public key to the authority in a secure manner and obtain a certificate. The user can then publish the certificate. Anyone needing this user's public key can obtain the certificate and verify that it is valid by way of the attached trusted signature. Figure 2.8 illustrates the process. The key steps can be summarized as follows:

1. User software (client) creates a pair of keys: one public and one private.
2. Client prepares unsigned certificate that includes User ID and user's public key.
3. User provides the unsigned certificate to a CA in some secure manner. This might require a face-to-face meeting or the use of registered mail.
4. CA creates signature as follows:
   a. CA uses a hash function to calculate the hash code of the unsigned certificate. A hash function is one that maps a variable-length data block or message into a fixed-length value called a hash code. Examples of hash functions are MD5 and SHA.
   b. CA encrypts the hash code with the CA's private key.
5. CA attaches signature to unsigned certificate to create a signed certificate.
6. CA returns signed certificate to client.
7. Client may provide signed certificate to any other user.
8. Any user may verify that the certificate is valid as follows:
   a. User calculates hash code of certificate (not including signature).
   b. User decrypts signature using CA's public key.
   c. User compares the results of (a) and (b). If there is a match, the certificate is valid.

One scheme has become universally accepted for formatting public-key certificates: the X.509 standard. X.509 certificates are used in most network security applications, including IP security, secure sockets layer (SSL), secure electronic transactions (SET), and S/MIME.

An X.509 certificate includes the following elements (Figure 22.3a):

- **Version:** Differentiates among successive versions of the certificate format; the default is version 1. If the Initiator Unique Identifier or Subject Unique Identifier are present, the value must be version 2. If one or more extensions are present, the version must be version 3.
- **Serial number:** An integer value, unique within the issuing CA, that is unambiguously associated with this certificate.
- **Signature algorithm identifier:** The algorithm used to sign the certificate, together with any associated parameters. Because this information is repeated in the Signature field at the end of the certificate, this field has little, if any, utility.
- **Issuer name:** X.500 name of the certificate authority (CA) that created and signed this certificate.
- **Period of validity:** Consists of two dates: the first and last on which the certificate is valid.

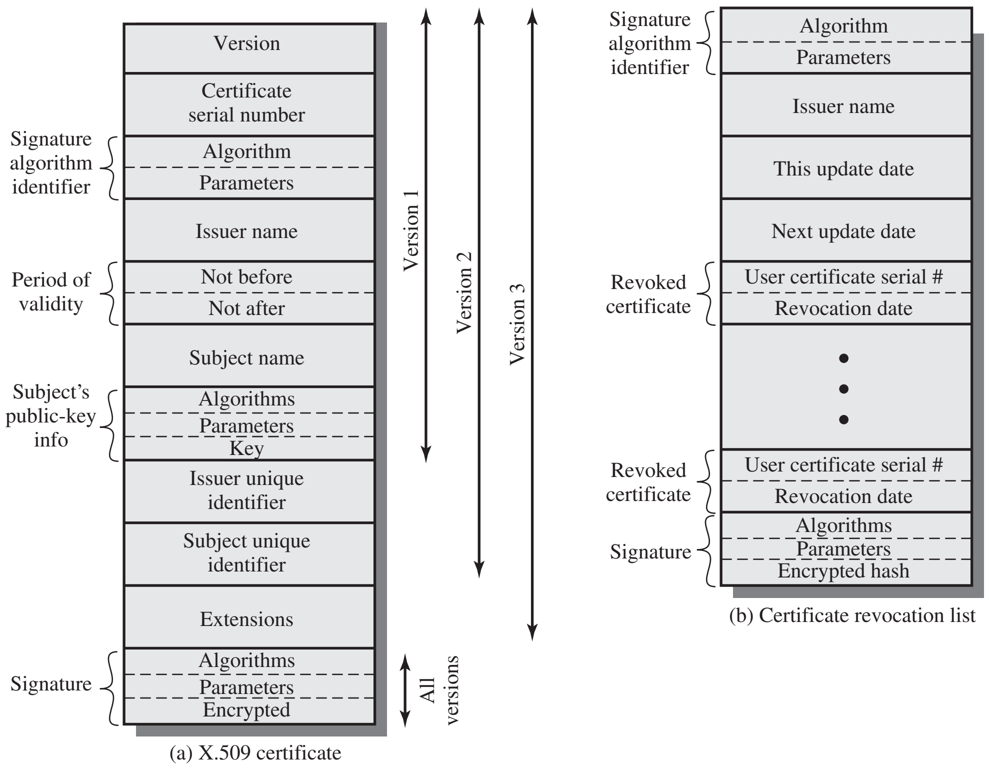

Figure 22.3 **X.509 Formats**

- **Subject name:** The name of the user to whom this certificate refers. That is, this certificate certifies the public key of the subject who holds the corresponding private key.
- **Subject's public-key information:** The public key of the subject, plus an identifier of the algorithm for which this key is to be used, together with any associated parameters.
- **Issuer unique identifier:** An optional bit string field used to identify uniquely the issuing CA in the event the X.500 name has been reused for different entities.
- **Subject unique identifier:** An optional bit string field used to identify uniquely the subject in the event the X.500 name has been reused for different entities.
- **Extensions:** A set of one or more extension fields. Extensions were added in version 3 and are discussed later in this section.
- **Signature:** Covers all of the other fields of the certificate; it contains the hash digest, or fingerprint, of the other fields, encrypted with the CA's private key. This field includes the signature algorithm identifier.

The unique identifier fields were added in version 2 to handle the possible reuse of subject and/or issuer names over time. The extensions field was added in X509.v3 to provide more flexibility and to convey information needed in special circumstances.

In addition, X.509 provides a format for use in revoking a key before it expires. This enables a user to cancel a key at any time. The user might do this if he or she thinks the key has been compromised or because of an upgrade in the user's software that requires generation of a new key.

Each certificate revocation list (CRL) posted to the directory is signed by the issuer and includes (Figure 22.3b) the issuer's name, the date the list was created, the date the next CRL is scheduled to be issued, and an entry for each revoked certificate. Each entry consists of the serial number of a certificate and revocation date for that certificate. Because serial numbers are unique within a CA, the serial number is sufficient to identify the certificate.

When a user receives a certificate in a message, the user must determine whether the certificate has been revoked. The user could check the directory each time a certificate is received. To avoid the delays (and possible costs) associated with directory searches, it is likely that the user would maintain a local cache of certificates and lists of revoked certificates.

22.3 PUBLIC-KEY INFRASTRUCTURE

RFC 2822 (*Internet Security Glossary*) defines public-key infrastructure (PKI) as the set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography. The principal objective for developing a PKI is to enable secure, convenient, and efficient acquisition of public keys. The Internet Engineering Task

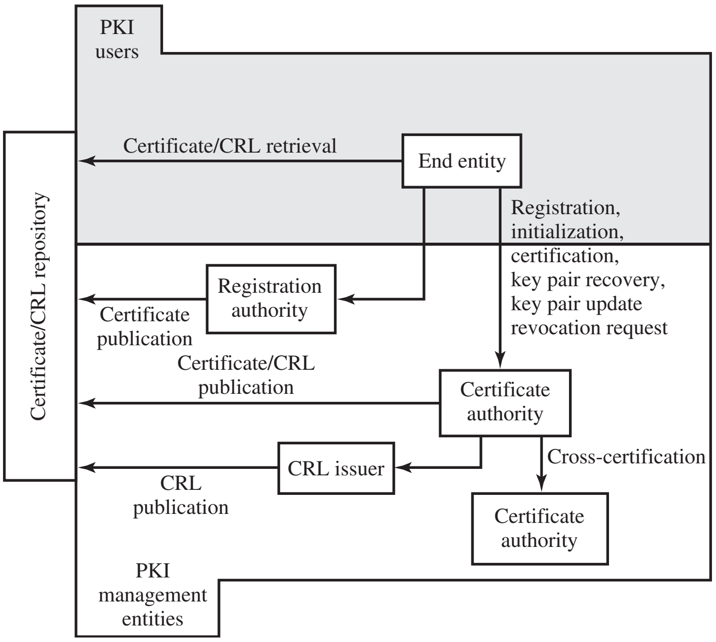
PKI users
Certificate/CRL retrieval
End entity
Registration, initialization, certification, key pair recovery, key pair update revocation request
Certificate/CRL repository
Registration authority
Certificate publication
Certificate/CRL publication
Certificate authority
CRL issuer
Cross-certification
CRL publication
Certificate authority
PKI management entities

Figure 22.4 **PKIX Architectural Model**

Force (IETF) Public Key Infrastructure X.509 (PKIX) working group has been the driving force behind setting up a formal (and generic) model based on X.509 that is suitable for deploying a certificate-based architecture on the Internet. This section describes the PKIX model.

Figure 22.4 shows the interrelationship among the key elements of the PKIX model. These elements are as follows:

- **End entity:** A generic term used to denote end users, devices (e.g., servers, routers), or any other entity that can be identified in the subject field of a public-key certificate. End entities typically consume and/or support PKI-related services.
- **Certification authority (CA):** The issuer of certificates and (usually) certificate revocation lists (CRLs). It may also support a variety of administrative functions, although these are often delegated to one or more registration authorities.
- **Registration authority (RA):** An optional component that can assume a number of administrative functions from the CA. The RA is often associated with the end entity registration process but can assist in a number of other areas as well.
- **CRL issuer:** An optional component that a CA can delegate to publish CRLs.
- **Repository:** A generic term used to denote any method for storing certificates and CRLs so that they can be retrieved by end entities.

PKIX Management Functions

PKIX identifies a number of management functions that potentially need to be supported by management protocols. These are indicated in Figure 22.4 and include the following:

- **Registration:** This is the process whereby a user first makes itself known to a CA (directly, or through an RA), prior to that CA issuing a certificate or certificates for that user. Registration begins the process of enrolling in a PKI. Registration usually involves some offline or online procedure for mutual authentication. Typically, the end entity is issued one or more shared secret keys used for subsequent authentication.
- **Initialization:** Before a client system can operate securely, it is necessary to install key materials that have the appropriate relationship with keys stored elsewhere in the infrastructure. For example, the client needs to be securely initialized with the public key and other assured information of the trusted CA(s), to be used in validating certificate paths.
- **Certification:** This is the process in which a CA issues a certificate for a user's public key and returns that certificate to the user's client system and/or posts that certificate in a repository.
- **Key pair recovery:** Key pairs can be used to support digital signature creation and verification, encryption and decryption, or both. When a key pair is used for encryption/decryption, it is important to provide a mechanism to recover the necessary decryption keys when normal access to the keying material is no longer possible; otherwise it will not be possible to recover the encrypted data. Loss of access to the decryption key can result from forgotten passwords/PINs, corrupted disk drives, damage to hardware tokens, and so on. Key pair recovery allows end entities to restore their encryption/decryption key pair from an authorized key backup facility (typically, the CA that issued the end entity's certificate).
- **Key pair update:** All key pairs need to be updated regularly (i.e., replaced with a new key pair) and new certificates issued. Update is required when the certificate lifetime expires and as a result of certificate revocation.
- **Revocation request:** An authorized person advises a CA of an abnormal situation requiring certificate revocation. Reasons for revocation include private key compromise, change in affiliation, and name change.
- **Cross certification:** Two CAs exchange information used in establishing a cross-certificate. A cross-certificate is a certificate issued by one CA to another CA that contains a CA signature key used for issuing certificates.

PKIX Management Protocols

The PKIX working group has defines two alternative management protocols between PKIX entities that support the management functions listed in the preceding subsection. RFC 2510 defines the certificate management protocols (CMP). Within CMP, each of the management functions is explicitly identified by specific protocol exchanges. CMP is designed to be a flexible protocol able to accommodate a variety of technical, operational, and business models.

RFC 2797 defines certificate management messages over CMS (CMC), where CMS refers to RFC 2630, cryptographic message syntax. CMC is built on earlier work and is intended to leverage existing implementations. Although all of the PKIX functions are supported, the functions do not all map into specific protocol exchanges.

22.4 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. [PELT07] lists the following as the principal elements of an identity management system:

- **Authentication:** Confirmation that a user corresponds to the user name provided.
- **Authorization:** Granting access to specific services and/or resources based on the authentication.
- **Accounting:** A process for logging access and authorization.
- **Provisioning:** The enrollment of users in the system.
- **Workflow automation:** Movement of data in a business process.
- **Delegated administration:** The use of role-based access control to grant permissions.
- **Password synchronization:** Creating a process for single sign-on (SSO) or reduced sign-on (RSO). Single sign-on enables a user to access all network resources after a single authentication. RSO may involve multiple sign-ons but requires less user effort than if each resource and service maintained its own authentication facility.
- **Self-service password reset:** Enables the user to modify his or her password.
- **Federation:** A process where authentication and permission will be passed on from one system to another, usually across multiple enterprises, reducing the number of authentications needed by the user.

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

Figure 22.5 [LINN06] 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

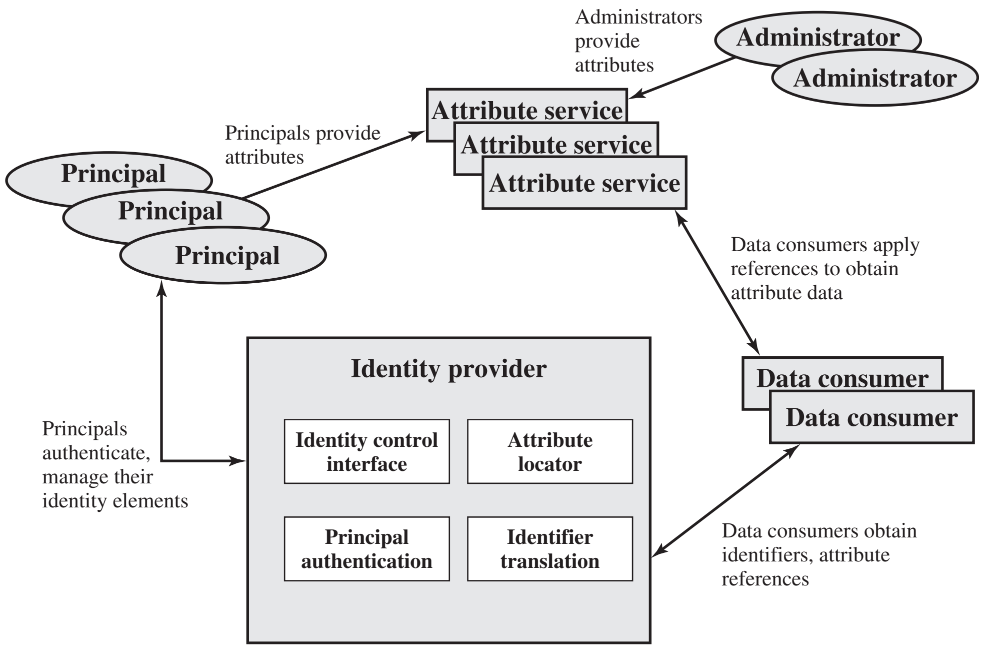
Administrators provide attributes
Administrator
Administrator
Attribute service
Attribute service
Attribute service
Principals provide attributes
Principal
Principal
Principal
Data consumers apply references to obtain attribute data
Identity provider
Identity control interface
Attribute locator
Principal authentication
Identifier translation
Data consumer
Data consumer
Principals authenticate, manage their identity elements
Data consumers obtain identifiers, attribute references

Figure 22.5 **Generic Identity Management Architecture**

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 authorization and privacy policies. Users may create some of the attributes to be associated with their digital identity, such as 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, 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.

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.

Standards Federated identity management makes use of a number of standards for that provide the building blocks for secure identity information exchange across different domains or heterogeneous systems:

- **The Extensible Markup Language (XML):** A markup language 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.

Examples To get some feel for the functionality of identity federation, we look at three scenarios, taken from [COMP06]. In the first scenario (Figure 22.6a), 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 the cooperatively exchanges user identifiers. Health.com maintains user identities 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 22.6b shows another type of browser-based scheme. PartsSupplier.com is a regular supplier of parts to Workplace.com. In this case, a role-based access control

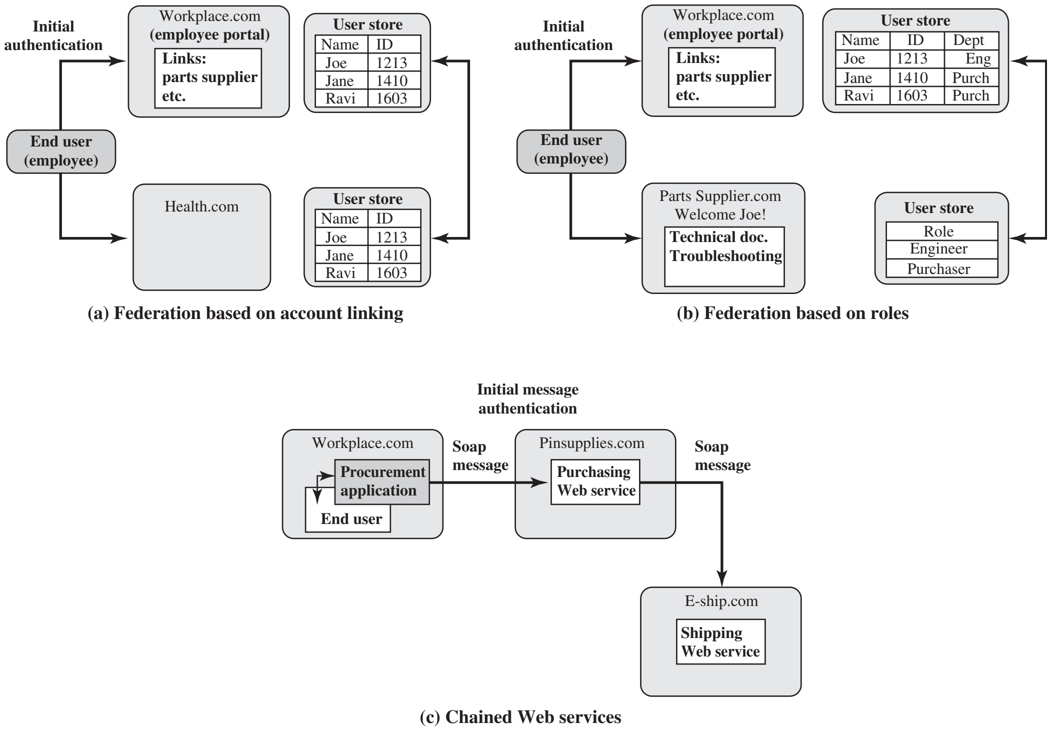

Figure 22.6 Federated Identity Scenarios

(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 PartSupplier.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 PartSupplier.com without having to authenticate to PartSupplier.com. For this scenario, PartSupplier.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 22.6c can be referred to as document based rather than browser based. In this example, Workplace.com has a purchasing agreement with PinSupplies.com and PinSupplies.com has a business relationship with E-Ship.com. In this example, 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 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.

22.5 RECOMMENDED READING AND WEB SITES

Most of the topics in this chapter are covered in greater detail in [STAL06a]. A painless way to get a grasp of Kerberos concepts is found in [BRYA88]. One of the best treatments of Kerberos is [KOHL94]. [PERL99] reviews various trust models that can be used in a PKI. [GUTM02] highlights difficulties in PKI use and recommends approaches for an effective PKI. [SHIM05] provides a brief overview of federated identity management and examines one approach to standardization. [BHAT07] describes an integrated approach to federated identity management couple with management of access control privileges.

BHAT07 Bhatti, R.; Bertino, E.; and Ghafoor, A. "An Integrated Approach to Federated Identity and Privilege Management in Open Systems." *Communications of the ACM*, February 2007.

BRYA88 Bryant, W. *Designing an Authentication System: A Dialogue in Four Scenes.* Project Athena document, February 1988. Available at http://web.mit.edu/kerberos/www/dialogue.html

GUTM02 Gutmann, P. "PKI: It's Not Dead, Just Resting." *Computer*, August 2002.

KOHL94 Kohl, J.; Neuman, B.; and Ts'o, T. "The Evolution of the Kerberos Authentication Service." In Brazier, F., and Johansen, D. *Distributed Open Systems.* Los Alamitos, CA: IEEE Computer Society Press, 1994. Available at http://web.mit.edu/kerberos/www/papers.html

PERL99 Perlman, R. "An Overview of PKI Trust Models." *IEEE Network*, November/December 1999.

SHIM05 Shim, S.; Bhalla, G.; and Pendyala, V. "Federated Identity Mangement." *Computer*, December 2005.

STAL06a Stallings, W. *Cryptography and Network Security: Principles and Practice, Fourth Edition.* Upper Saddle River, NJ: Prentice Hall, 2003.

Recommended Web sites:

- **MIT Kerberos Site:** Information about Kerberos, including the FAQ, papers and documents, and pointers to commercial product sites
- **USC/ISI Kerberos Page:** Another good source of Kerberos material
- **Kerberos Working Group:** IETF group developing standards based on Kerberos
- **Public-Key Infrastructure Working Group:** IETF group developing standards based on X.509v3
- **NIST PKI Program:** Good source of information

22.6 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

certificate authority (CA)	identity management	Public-Key Infrastructure (PKI)
federated identity management	Kerberos	X.509
	Kerberos realm	

Review Questions

22.1 What are the principal elements of a Kerberos system?

22.2 What is Kerberos realm?

22.3 What are the differences between versions 4 and 5 of Kerberos?

22.4 What is X.509?

22.5 What is the role of a CA in X.509?

22.6 What is a public key infrastructure?

22.7 List the key elements of the PKIX model.

Problems

22.1 CBC (cipher block chaining) has the property that if an error occurs in transmission of ciphertext block C_l, then this error propagates to the recovered plaintext blocks P_l and P_{l+1}. Version 4 of Kerberos uses an extension to CBC, called the propagating CBC (PCBC) mode. This mode has the property that an error in one ciphertext block is propagated to all subsequent decrypted blocks of the message, rendering each block useless. Thus, data encryption and integrity are combined in one operation. For PCBC, the input to the encryption algorithm is the XOR of the current plaintext block, the preceding cipher text block, and the preceding plaintext block:

$$\mathrm{C}_n = \mathrm{E}(K, [C_{n-1} \oplus P_{n-1} \oplus P_n])$$

On decryption, each ciphertext block is passed through the decryption algorithm. Then the output is XORed with the preceding ciphertext block and the preceding plaintext block.

a. Draw a diagram similar to those used in Chapter 21 to illustrate PCBC.

b. Use a Boolean equation to demonstrate that PCBC works.

c. Show that a random error in one block of ciphertext is propagated to all subsequent blocks of plaintext.

22.2 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.