AUTHENTICATION SERVICES KERBEROS

Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users. Kerberos relies exclusively on conventional encryption, making no use of public-key encryption.

The following are the requirements for Kerberos:

- **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] 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.

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. To counter this threat, servers must be able to confirm the identities of clients who request service. 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. In addition, the AS shares a unique secret key with each server. The simple authentication dialogue is as follows:

A more secure authentication dialogue

There are two major problems associated with the previous approach:

Plaintext transmission of the password.

Each time a user has to enter the password.

To solve these problems, we introduce a scheme for avoiding plaintext passwords, and anew server, known as ticket granting server (TGS). The hypothetical scenario is as follows:

Once per user logon session:

1. C >> AS: IDc||IDtgs
2. AS >> C: Ek_c ($Ticket_{tgs}$)

Once per type of service:

3. C >> TGS: IDc||IDv||$Ticket_{tgs}$
4. TGS >> C: $ticket_v$

Once per service session:

5. C >> V: IDc||$ticket_v$

$Ticket_{tgs}$= Ek_{tgs}(IDc||ADc||IDtgs||TS1||Lifetime1) $Ticket_v$= Ek_v(IDc||ADc||IDv||TS2||Lifetime2)

C: Client, AS: Authentication Server, V: Server, IDc : ID of the client, Pc:Password of the client, ADc: Address of client, IDv : ID of the server, Kv: secret key shared by AS and V, ||: concatenation, IDtgs: ID of the TGS server, TS1, TS2: time stamps, lifetime: lifetime of the ticket.

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 and password 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.

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.

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 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 nd 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

Kerbero V4 Authentication Dialogue Message Exchange

Two additional problems remain in the more secure authentication dialogue:

Lifetime associated with the ticket granting ticket. If the lifetime is very short, then the user will be repeatedly asked for a password. If the lifetime is long, then the opponent has the greater opportunity for replay.

Requirement for the servers to authenticate themselves to users. The actual Kerberos protocol version 4 is as follows:

- a basic third-party authentication scheme
- have an Authentication Server (AS)

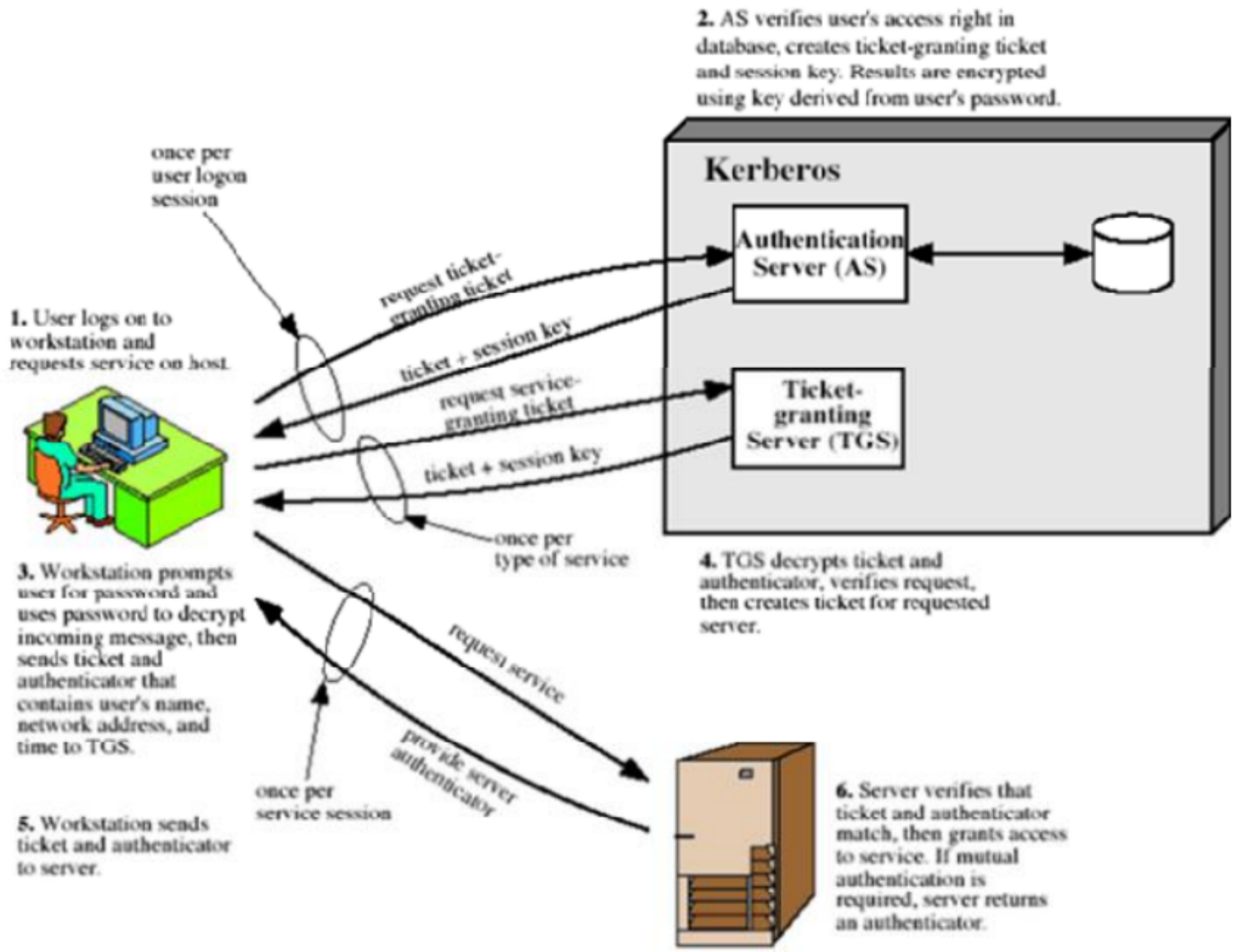
– users initially negotiate with AS to identify self

–AS provides a non-corruptible authentication credential (ticket granting ticket TGT)

- have a Ticket Granting server (TGS)

–users subsequently request access to other services from TGS on basis of users

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 asswo d, and p t ctin contents of m ssa e (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
ID_{tgs}	Confirms that this ticket is for the TGS

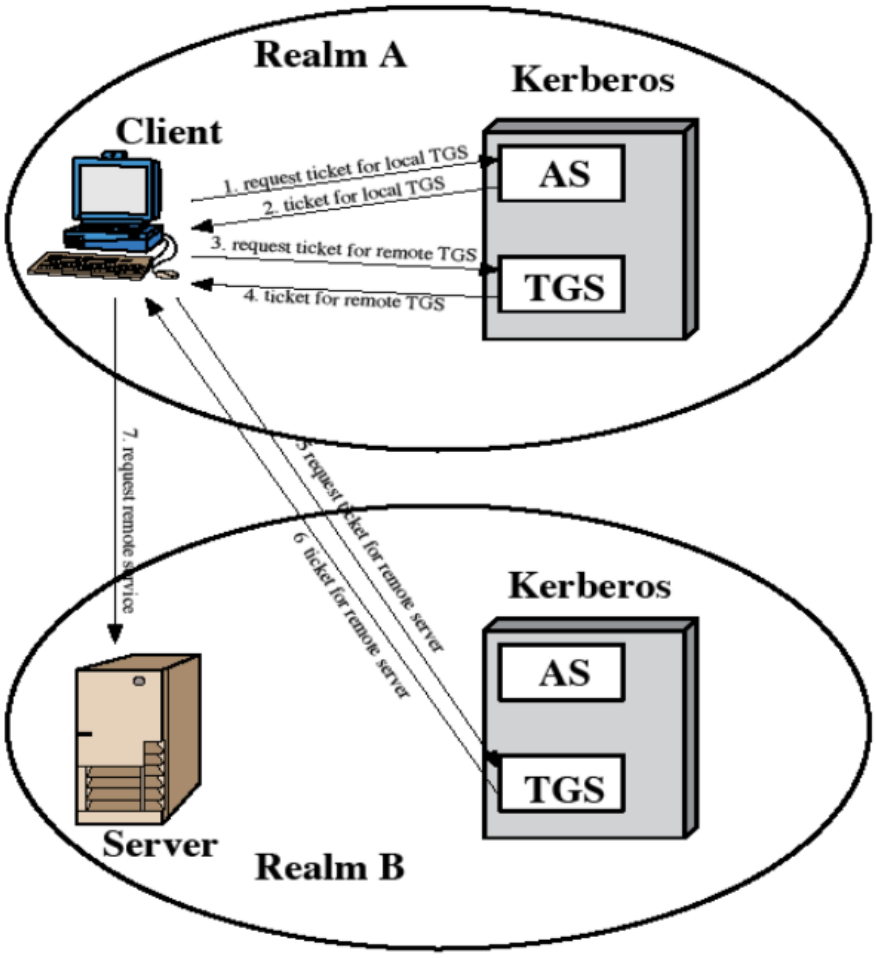

2. AS verifies user's access right in database, creates ticket-granting ticket and session key. Results are encrypted using key derived from user's password.
Kerberos
Authentication Server (AS)
Ticket-granting Server (TGS)
once per user logon session
request ticket-granting ticket
ticket + session key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

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.Such an environment is referred to as a **Kerberos realm**.

The concept of *realm* can be explained as follows.



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.

Kerberos version 5

Version 5 of Kerberos provides a number of improvements over version 4.

- developed in mid 1990's
- provides improvements over v4
  - addresses environmental shortcomings and technical deficiencies
- specified as Internet standard RFC 1510

Differences between version 4 and 5

Version 5 is intended to address the limitations of version 4 in two areas:

Environmental shortcomings

- o encryption system dependence
- o internet protocol dependence
- o message byte ordering
- o ticket lifetime
- o authentication forwarding
- o inter-realm authenticaiton

Technical deficiencies

- o double encryption o PCBC encryption o Session keys
- o Password attacks

The version 5 authentication dialogue

(a) Authentication Service Exchange: to obtain ticket-granting ticket
(1) C → AS: Options ‖ ID_c ‖ $Realm_c$ ‖ ID_{tgs} ‖ $Times$ ‖ $Nonce_1$
(2) AS → C: $Realm_c \| ID_C \| Ticket_{tgs} \| E_{K_c}\left[K_{c,tgs} \| Times \| Nonce_1 \| Realm_{tgs} \| ID_{tgs}\right]$
$Ticket_{tgs} = E_{K_{tgs}}\left[Flags \| K_{c,tgs} \| Realm_c \| ID_C \| AD_C \| Times\right]$
(b) Ticket-Granting Service Exchange: to obtain service-granting ticket
(3) C → TGS: Options ‖ ID_v ‖ $Times$ ‖ ‖ $Nonce_2$ ‖ $Ticket_{tgs}$ ‖ $Authenticator_c$
(4) TGS → C: $Realm_c \| ID_C \| Ticket_v \| E_{K_{c,tgs}}\left[K_{c,v} \| Times \| Nonce_2 \| Realm_v \| ID_V\right]$
$Ticket_{tgs} = E_{K_{tgs}}\left[Flags \| K_{c,tgs} \| Realm_c \| ID_C \| AD_C \| Times\right]$
$Ticket_v = E_{K_v}\left[Flags \| K_{c,v} \| Realm_c \| ID_C \| AD_C \| Times\right]$
$Authenticator_c = E_{K_{c,tgs}}\left[ID_C \| Realm_c \| TS_1\right]$
(c) Client/Server Authentication Exchange: to obtain service
(5) C → V: Options ‖ Ticket_v ‖ Authenticator_c
(6) V → C: $E_{K_{c,V}}$ [TS_2 ‖ Subkey ‖ Seq#]
$Ticket_v = E_{K_v}\left[Flags \| K_{c,v} \| Realm_c \| ID_C \| AD_C \| Times\right]$
$Authenticator_c = E_{K_{c,V}}\left[ID_C \| Realm_c \| TS_2 \| Subkey \| Seq\#\right]$

Overview:

- **issued by a Certification Authority (CA), containing:**

– version (1, 2, or 3)

– serial number (unique within CA) identifying certificate

– signature algorithm identifier

[redacted]

– issuer X.500 name (CA)

– period of validity (from - to dates)

– subject X.500 name (name of owner)

– subject public-key info (algorithm, parameters, key)

– issuer unique identifier (v2+)

– subject unique identifier (v2+)

– extension fields (v3)

– signature (of hash of all fields in certificate)

- **notation CA<<A>> denotes certificate for A signed by CA**

X.509 defines a framework for the provision of authentication services by the X.500 directory to its users. The directory may serve as a repository of public-key certificates. Each certificate contains the public key of a user and is signed with the private key of a trusted certification authority. In addition, X.509 defines alternative authentication protocols based on the use of public-key certificates.

X.509 is an important standard because the certificate structure and authentication protocols defined in X.509 are used in a variety of contexts. For example, the X.509 certificate format is used in S/MIME), IP Security and SSL/TLS and SET

X.509 is based on the use of public-key cryptography and digital signatures. The standard does not

dictate the use of a specific algorithm but recommends RSA. The digital signature scheme is assumed to require the use of a hash function.

Certificates

The heart of the X.509 scheme is the public-key certificate associated with each user. These user certificates are assumed to be created by some trusted certification authority (CA) and placed in the directory by the CA or by the user.

Version:

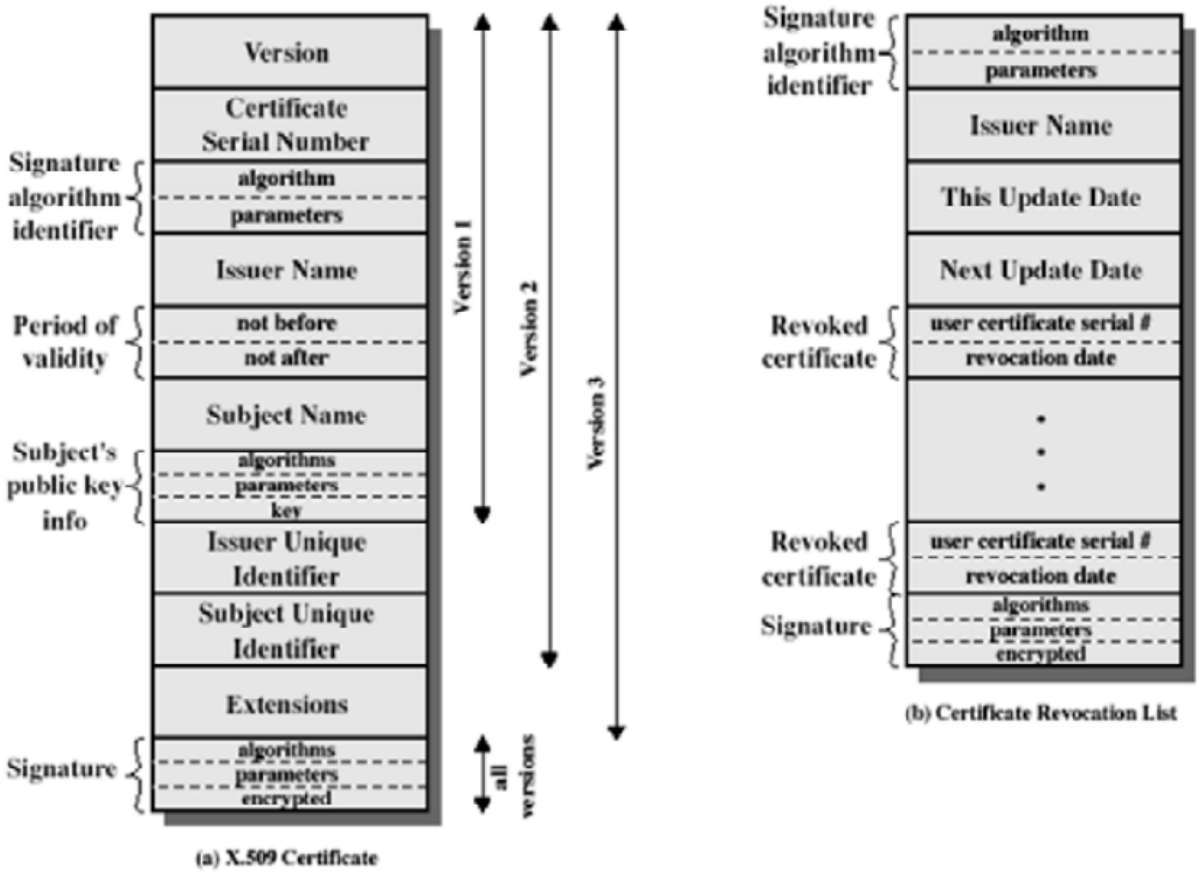
Differentiates among successive versions of the certificate format; the default is version 1. If the Issuer Unique Identifier or Subject Unique Identifier are present, the value must be version 2. If one or more extensions are present, the version must be version 3.

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 CA that created and signed this certificate.

Period of validity:

Consists of two dates: the first and last on which the certificate is valid.

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 code of the other fields, encrypted with the CA's private key. This field includes the signature algorithm identifie

The standard uses the following notation to define a certificate: CA<<A>> = CA {V, SN, AI, CA, T_A, A, Ap} where

Y <<X>> = the certificate of user X issued by certification authority Y Y {I} = the signing of I by Y. It consists of I with an encrypted hash code appended

The CA signs the certificate with its private key. If the corresponding public key is known to a user, then that user can verify that a certificate signed by the CA is valid.

Obtaining a User's Certificate

User certificates generated by a CA have the following characteristics:

Any user with access to the public key of the CA can verify the user public key that was certified.

No party other than the certification authority can modify the certificate without this being detected.

ecause certificates are unforgeable, they can be placed in a directory without the need for the directory to make special efforts to protect them.

If all users subscribe to the same CA, then there is a common trust of that CA. All user certificates can be placed in the directory for access by all users.

If there is a large community of users, it may not be practical for all users to subscribe to the same CA. Because it is the CA that signs certificates, each participating user must have a copy of the CA's own public key to verify signatures. This public key must be provided to each user in an absolutely secure (with respect to integrity and authenticity) way so that the user has confidence in the associated certificates. Thus, with many users, it may be more practical for there to be a number of CAs, each of which securely provides its public key to some fraction of the users.

Now suppose that A has obtained a certificate from certification authority X_1 and B has obtained a certificate from CA X_2. If A does not securely know the public key of X_2, then B's certificate, issued by X_2, is useless to A.

A can read B's certificate, but A cannot verify the signature. However, if the two CAs have securely exchanged their own public keys, the following procedure will enable A to obtain B's public key:

1. A obtains, from the directory, the certificate of X_2 signed by X_1. Because A securely knows X_1's public key, A can obtain X_2's public key from its certificate and verify it by means of X_1's signature on the certificate.
2. A then goes back to the directory and obtains the certificate of B signed by X_2

Because A now has a trusted copy of X_2's public key, A can verify the signature and securely obtain B's public key.

A has used a chain of certificates to obtain B's public key. In the notation of X.509, this chain is expressed as

$X_1\langle\langle X_2\rangle\rangle$ $X_2\langle\langle B\rangle\rangle$

In the same fashion, B can obtain A's public key with the reverse chain: $X_2\langle\langle X_1\rangle\rangle$ $X_1\langle\langle A\rangle\rangle$

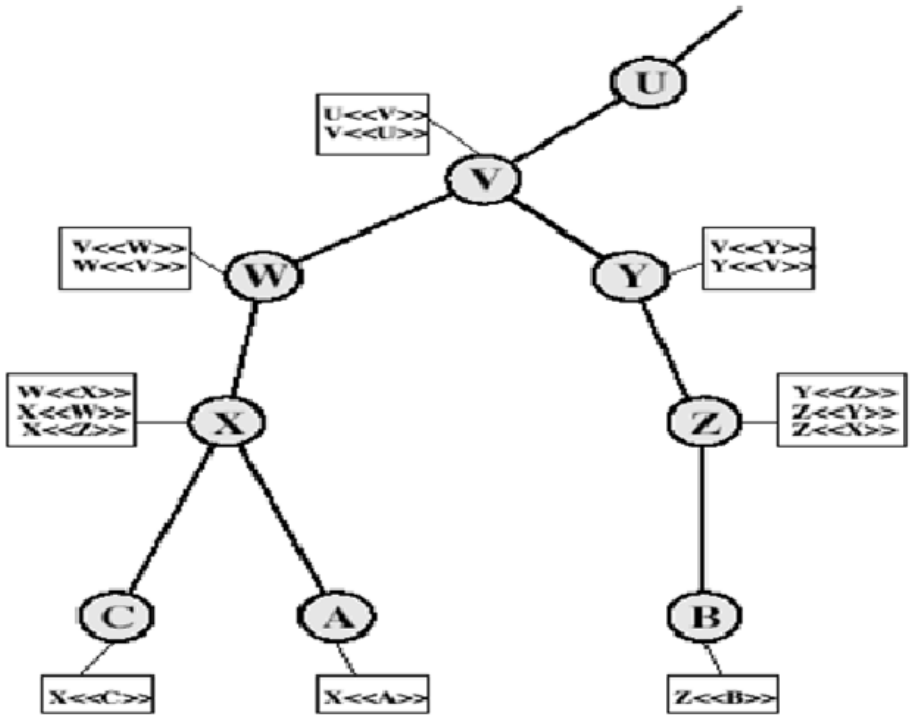
This scheme need not be limited to a chain of two certificates. An arbitrarily long path of CAs can be followed to produce a chain. A chain with N elements would be expressed as

$X_1\langle\langle X_2\rangle\rangle$ $X_2\langle\langle X_3\rangle\rangle \ldots X_N\langle\langle B\rangle\rangle$

Messages to B. If A wishes to receive encrypted messages back from B, or to sign messages sent to B, then B will require A's public key, which can be obtained from the following certification path:

Z<<Y>> Y <<V>> V <<W>> W <<X>>X <<A>>

B can obtain this set of certificates from the directory, or A can provide them as part of its initial message to B.



Certificate Revocation

- Certificates have a period of validity
- may need to revoke before expiry, for the following reasons eg:

1. user's private key is compromised
2. User is no longer certified by this CA

3. CA's certificate is compromised

- CA's maintain list of revoked certificates

1. the Certificate Revocation List (CRL)

- users should check certs with CA's CRL

Authentication Procedures

X.509 includes three alternative authentication procedures:

- **One-Way Authentication**
- **Two-Way Authentication**
- **Three-Way Authentication**
- all use public-key signatures

One-Way Authentication

- 1 message (A->B) used to establish
  - the identity of A and that message is from A
  - message was intended for B
  - integrity & originality of message
- message must include timestamp, nonce, B's identity and is signed by A

Two-Way Authentication

- 2 messages (A->B, B->A) which also establishes in addition:
  - the identity of B and that reply is from B
  - that reply is intended for A
  - integrity & originality of reply

  reply includes original nonce from A, also timestamp and nonce from B

Three-Way Authentication

- 3 messages (A->B, B->A, A->B) which enables above authentication without synchronized clocks
- has reply from a back to B containing signed copy of nonce from B
- means that timestamps need not be checked or relied upon

X.509 Version 3

The X.509 version 2 format does not convey all of the information that recent design and implementation experience has shown to be needed. [FORD95] lists the following requirements not satisfied by version 2:

1. The Subject field is inadequate to convey the identity of a key owner to a public- key user.
2. The Subject field is also inadequate for many applications, which typically recognize entities by an Internet e-mail address, a URL, or some other Internet- related identification.
3. There is a need to indicate security policy information. There is a need to limit the damage that can result from a faulty or malicious CA by setting constraints on the applicability of a particular certificate.
4. It is important to be able to identify different keys used by the same owner at different times.

The certificate extensions fall into three main categories: key and policy information, subject and issuer attributes, and certification path constraints.

Key and Policy Information

These extensions convey additional information about the subject and issuer keys, plus indicators of certificate policy.. For example, a policy might be applicable to the authentication of electronic data interchange (EDI) transactions for the trading of goods within a given price range.

This area includes the following:

Authority key identifier: Identifies the public key to be used to verify the signature on this certificate or CRL.

Subject key identifier: Identifies the public key being certified. Useful for subject key pair updating.

Key usage: Indicates a restriction imposed as to the purposes for which, and the policies under which, the certified public key may be used.

Private-key usage period: Indicates the period of use of the private key corresponding to the public key.. For example, with digital signature keys, the usage period for the signing private key is typically shorter than that for the verifying public key.

Certificate policies: Certificates may be used in environments where multiple policies apply.

Policy mappings: Used only in certificates for CAs issued by other CAs.

Certificate Subject and Issuer Attributes

These extensions support alternative names, in alternative formats, for a certificate subject or certificate issuer and can convey additional information about the certificate subject, to increase a certificate user's confidence that the certificate subject is a particular person or entity. For example, information such as postal address, position within a corporation, or picture image may be required. The extension fields in this area include the following:

Subject alternative name: Contains one or more alternative names, using any of a variety of forms

Subject directory attributes: Conveys any desired X.500 directory attribute values for the subject of this certificate.

Certification Path Constraints

These extensions allow constraint specifications to be included in certificates issued for CAs by other CAs.The extension fields in this area include the following:

Basic constraints: Indicates if the subject may act as a CA. If so, a certification path length constraint may be specified.

Name constraints: Indicates a name space within which all subject names in subsequent certificates in a certification path must be located.

Policy constraints: Specifies constraints that may require explicit certificate policy identification or inhibit policy mapping for the remainder of the certification path.

ELECTRONIC MAIL SECURITY PRETTY GOOD PRIVACY (PGP)

PGP provides the confidentiality and authentication service that can be used for electronic mail and file storage applications. The steps involved in PGP are

Select the best available cryptographic algorithms as building blocks.

Integrate these algorithms into a general purpose application that is independent of operating system and processor and that is based on a small set of easy-to-use commands.

Make the package and its documentation, including the source code, freely available via the internet, bulletin boards and commercial networks.

Enter into an agreement with a company to provide a fully compatible, low cost

commercial version of PGP.

PGP has grown explosively and is now widely used. A number of reasons can be cited for this growth.

- It is available free worldwide in versions that run on a variety of platform.
- It is based on algorithms that have survived extensive public review and are considered extremely secure.
- e.g., RSA, DSS and Diffie Hellman for public key encryption CAST-128, IDEA and 3DES for conventional encryption SHA-1 for hash coding.
- it has a wide range of applicability.
- It was not developed by, nor it is controlled by, any governmental or standards organization.

Operational description

The actual operation of PGP consists of five services: authentication, confidentiality, compression, e-mail compatibility and segmentation.

1. Authentication

The sequence for authentication is as follows:

The sender creates the message

SHA-1 is used to generate a 160-bit hash code of the message

The hash code is encrypted with RSA using the sender's private key and the result is prepended to the message

The receiver uses RSA with the sender's public key to decrypt and recover the hash code.

The receiver generates a new hash code for the message and compares it with the decrypted hash code. If the two match, the message is accepted as authentic.

2. Confidentiality

Confidentiality is provided by encrypting messages to be transmitted or to be stored locally as files. In both cases, the conventional encryption algorithm CAST-128 may be used. The 64-bit cipher feedback (CFB) mode is used.

In PGP, each conventional key is used only once. That is, a new key is generated as a random 128-bit number for each message. Thus although this is referred to as **a session key**, it is in reality a **one time key**. To protect the key, it is encrypted with the receiver's public key.

The sequence for confidentiality is as follows:

- The sender generates a message and a random 128-bit number to be used as a session key for this message only.
- The message is encrypted using CAST-128 with the session key.
- The session key is encrypted with RSA, using the receiver's public key and is prepended to the message.
- The receiver uses RSA with its private key to decrypt and recover the session key.
- The session key is used to decrypt the message.

Confidentiality and authentication

Here both services may be used for the same message. First, a signature is generated for the plaintext message and prepended to the message. Then the plaintext plus the signature is encrypted using CAST-128 and the session key is encrypted using RSA.

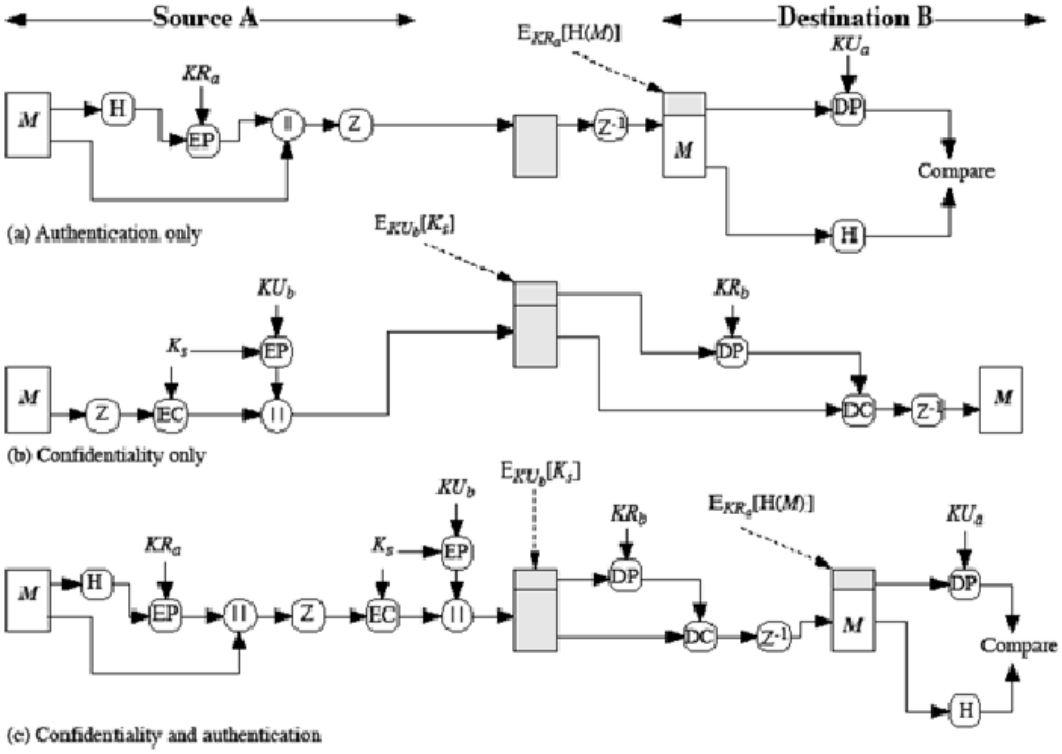


Figure 15.1 PGP Cryptographic Functions

3. The value of the key legitimacy field is calculated on the basis of the signature trust fields present in this entry. If at least one signature has a signature trust value of ultimate, then the key legitimacy value is set to complete.

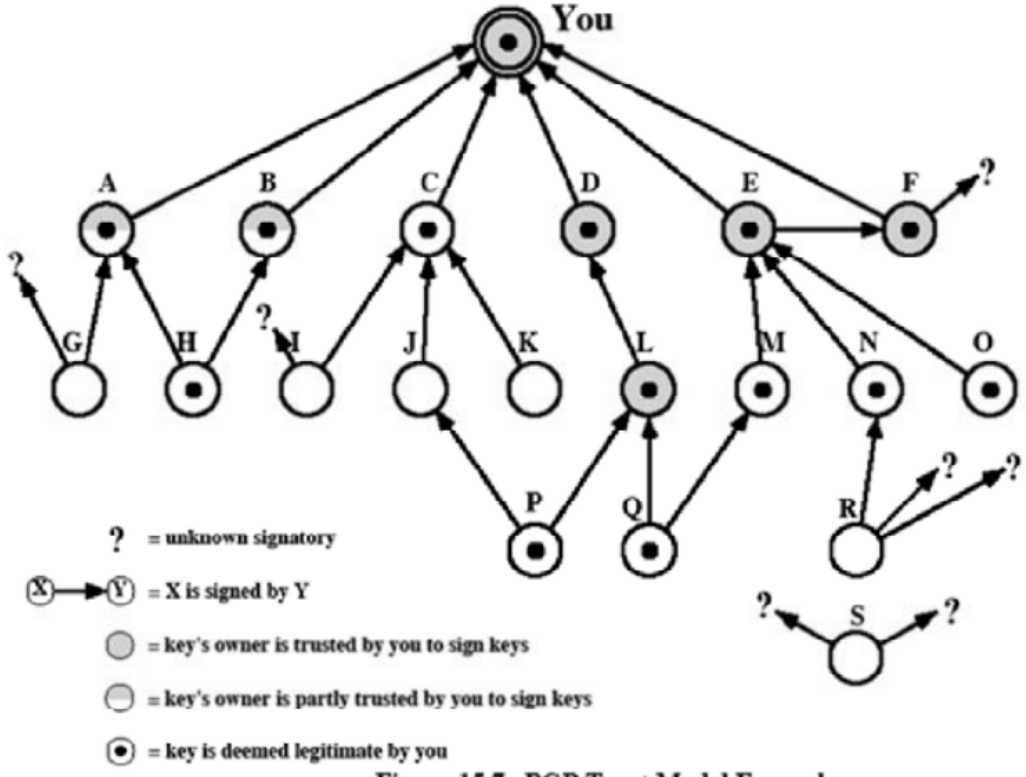


Figure 15.7 PGP Trust Model Example

The node labeled "You" refers to the entry in the public-key ring corresponding to this user. This key is legitimate and the OWNERTRUST value is ultimate trust. Each other node in the key ring has an OWNERTRUST value of undefined unless some other value is assigned by the user. In this example, this user has specified that it always trusts the following users to sign other keys: D, E, F, L. This user partially trusts users A and B to sign other keys.

So the shading, or lack thereof, of the nodes in Figure 15.7 indicates the level of trust assigned by this user. The tree structure indicates which keys have been signed by which

other users. If a key is signed by a user whose key is also in this key ring, the arrow joins the signed key to the signatory. If the key is signed by a user whose key is not present in this key ring, the arrow joins the signed key to a question mark, indicating that the signatory is unknown to this user.

Note that all keys whose owners are fully or partially trusted by this user have been signed by this user, with the exception of node L.

1. We assume that two partially trusted signatures are sufficient to certify a key. Hence, the key for user H is deemed legitimate by PGP because it is signed by A and B, both of whom are partially trusted.
2. A key may be determined to be legitimate because it is signed by one fully trusted or two partially trusted signatories, but its user may not be trusted to sign other keys. For example, N's key is legitimate because it is signed by E, whom this user trusts, but N is not trusted to sign other keys because this user has not assigned N that trust value. Therefore, although R's key is signed by N, PGP does not consider R's key legitimate. This situation makes perfect sense. If you wish to send a private message to some individual, it is not necessary that you trust that individual in any respect. It is only necessary that you are sure that you have the correct public key for that individual.
3. Figure 15.7 also shows an example of a detached "orphan" node S, with two unknown signatures. Such a key may have been acquired from a key server. PGP cannot assume that this key is legitimate simply because it came from a reputable server. The user must declare the key legitimate by signing it or by telling PGP that it is willing to trust fully one of the key's signatories.

S/MIME

S/MIME (Secure/Multipurpose Internet Mail Extension) is a security enhancement to the MIME Internet e-mail format standard, based on technology from RSA Data Security. S/MIME is defined in a number of documents, most importantly RFCs 3369, 3370, 3850 and 3851.

Multipurpose Internet Mail Extensions

MIME is an extension to the RFC 822 framework that is intended to address some of the problems and limitations of the use of SMTP (Simple Mail Transfer Protocol) or some other mail transfer protocol and RFC 822 for electronic mail. Following are the limitations of SMTP/822 scheme:

1. SMTP cannot transmit executable files or other binary objects.
2. SMTP cannot transmit text data that includes national language characters because these are represented by 8-bit codes with values of 128 decimal or higher, and SMTP is limited to 7-bit ASCII.
3. SMTP servers may reject mail message over a certain size.
4. SMTP gateways that translate between ASCII and the character code EBCDIC do not use a consistent set of mappings, resulting in translation problems.
5. SMTP gateways to X.400 electronic mail networks cannot handle nontextual data included in X.400 messages.
6. Some SMTP implementations do not adhere completely to the SMTP standards defined in RFC 821. Common problems include:

- o Deletion, addition, or reordering of carriage return and linefeed
- o Truncating or wrapping lines longer than 76 characters
- o Removal of trailing white space (tab and space characters)
- o Padding of lines in a message to the same length
- o Conversion of tab characters into multiple space characters

MIME is intended to resolve these problems in a manner that is compatible with existing RFC 822 implementations. The specification is provided in RFCs 2045 through 2049.

Overview

The MIME specification includes the following elements:

1. **Five new message header** fields are defined, which may be included in an RFC 822 header. These fields provide information about the body of the message.
2. **A number of content formats** are defined, thus standardizing representations that support multimedia electronic mail.
3. **Transfer encodings** are defined that enable the conversion of any content format into a form that is protected from alteration by the mail system.

In this subsection, we introduce the five message header fields. The next two subsections deal with content formats and transfer encodings.

The five header fields defined in MIME are as follows:

MIME-Version: Must have the parameter value 1.0. This field indicates that the message conforms to RFCs 2045 and 2046.

Content-Type: Describes the data contained in the body with sufficient detail

Content-Transfer-Encoding: Indicates the type of transformation that has been used to represent the body of the message in a way that is acceptable for mail transport.

Content-ID: Used to identify MIME entities uniquely in multiple contexts.

Content-Description: A text description of the object with the body; this is useful when the object is not readable (e.g., audio data).

MIME Content Types

The bulk of the MIME specification is concerned with the definition of a variety of content types. This reflects the need to provide standardized ways of dealing with a wide variety of information representations in a multimedia environment.

Table 15.3 lists the content types specified in RFC 2046. There are seven different major types of content and a total of 15 subtypes

Type	**Subtype**	**Description**
Text	Plain	Unformatted text; may be ASCII or ISO 8859.
	Enriched	Provides greater format flexibility.
Multipart	Mixed	The different parts are independent but are to be transmitted together. They should be presented to the receiver in the order that they appear in the mail message.
	Parallel	Differs from Mixed only in that no order is defined for delivering the parts to the receiver.
	Alternative	The different parts are alternative versions of the same information. They are ordered in increasing faithfulness to the original, and the recipient's mail system should display the
	Digest	Similar to Mixed, but the default type/subtype of each part is message/rfc822.
Message	rfc822	The body is itself an encapsulated message that conforms to RFC 822.
	Partial	Used to allow fragmentation of large mail items, in a way that is transparent to the recipient.
	External-body	Contains a pointer to an object that exists elsewhere.
Image	jpeg	The image is in JPEG format, JFIF encoding.

MIME Transfer Encodings

The other major component of the MIME specification, in addition to content type specification, is a definition of transfer encodings for message bodies. The objective is to provide reliable delivery across the largest range of environments.

The MIME standard defines two methods of encoding data. The Content-Transfer- Encoding field can actually take on six values, as listed in Table 15.4. For SMTP transfer, it is safe to use the 7bit form. The 8bit and binary forms may be usable in other mail transport contexts. Another Content-Transfer-Encoding value is x-token, which indicates that some other encoding scheme is used, for which a name is to be supplied. The two actual encoding schemes defined are quoted-printable and base64.

MIME Transfer Encodings	
7bit	The data are all represented by short lines of ASCII characters.
8bit	The lines are short, but there may be non-ASCII characters (octets with the high-order bit set)
binary	Not only may non-ASCII characters be present but the lines are not necessarily short enough for SMTP transport
quoted-printable	Encodes the data in such a way that if the data being encoded are mostly ASCII text, the encoded form of the data remains largely recognizable by .
base64	Encodes data by mapping 6-bit blocks of input to 8-bit blocks of output, all of which are printable ASCII characters.
x-token	A named nonstandard encoding.

The quoted-printable transfer encoding is useful when the data consists largely of octets that correspond to printable ASCII characters. In essence, it represents nonsafe characters by the hexadecimal representation of their code and introduces reversible (soft) line breaks to limit message lines to 76 characters.

The base64 transfer encoding, also known as radix-64 encoding, is a common one for encoding arbitrary binary data in such a way as to be invulnerable to the processing by mail transport programs.

Canonical Form

An important concept in MIME and S/MIME is that of canonical form. Canonical form is a format, appropriate to the content type that is standardized for use between systems. This is in contrast to native form, which is a format that may be peculiar to a particular system.

S/MIME Functionality

In terms of general functionality, S/MIME is very similar to PGP. Both offer the ability to sign and/or encrypt messages. In this subsection, we briefly summarize S/MIME capability. We then look in more detail at this capability by examining message formats and message preparation.

Functions

S/MIME provides the following functions:

Enveloped data: This consists of encrypted content of any type and encrypted- content encryption keys for one or more recipients.

Signed data: A digital signature is formed by taking the message digest of the content to be signed and then encrypting that with the private key of the signer. The content plus signature are then encoded using base64 encoding. A signed data message can only be viewed by a recipient with S/MIME capability.

Clear-signed data: As with signed data, a digital signature of the content is formed. However, in this case, only the digital signature is encoded using base64. As a result, recipients without S/MIME capability can view the message content, although they cannot verify the signature.

Signed and enveloped data: Signed-only and encrypted-only entities may be nested, so that encrypted data may be signed and signed data or clear-signed data may be encrypted.

Overview of IPsec Services and Functions

IPsec is not a single protocol, but rather a set of services and protocols that provide a complete security solution for an IP network. These services and protocols combine to provide various types of protection. Since IPsec works at the IP layer, it can provide these protections for any higher-layer TCP/IP application or protocol without the need for additional security methods, which is a major strength. Some of the kinds of protection services offered by IPsec include the following:

- Encryption of user data for privacy
- Authentication of the integrity of a message to ensure that it is not changed en route
- Protection against certain types of security attacks, such as replay attacks
- The ability for devices to negotiate the security algorithms and keys required to meet their security needs
- Two security modes, tunnel and transport, to meet different network needs

KEY CONCEPT *IPsec* is a contraction of *IP Security*, and it consists of a set of services and protocols that provide security to IP networks. It is defined by a sequence of several Internet standards.

IPsec Standards

Since IPsec is actually a collection of techniques and protocols, it is not defined in a single Internet standard. Instead, a collection of RFCs defines the architecture, services, and specific protocols used in IPsec. Some of the most important of these are shown in Table 29-1, all of which were published in November 1998.

IPsec Core Protocols

To support these activities, a number of different components make up the total package known as IPsec, as shown in Figure 29-1. The two main pieces are a pair of technologies sometimes called the *core protocols* of IPsec, which actually do the work of encoding information to ensure security:

IPsec Authentication Header (AH) This protocol provides authentication services for IPsec. It allows the recipient of a message to verify that the supposed originator of a message was actually fact the one that sent it. It also allows the recipient to verify that intermediate devices en route haven't changed any of the data in the datagram. It also provides protection against so-called *replay attacks*, whereby a message is captured by an unauthorized user and resent.

Encapsulating Security Payload (ESP) AH ensures the integrity of the data in datagram, but not its privacy. When the information in a datagram is "for your eyes only," it can be further protected using ESP, which encrypts the payload of the IP datagram.

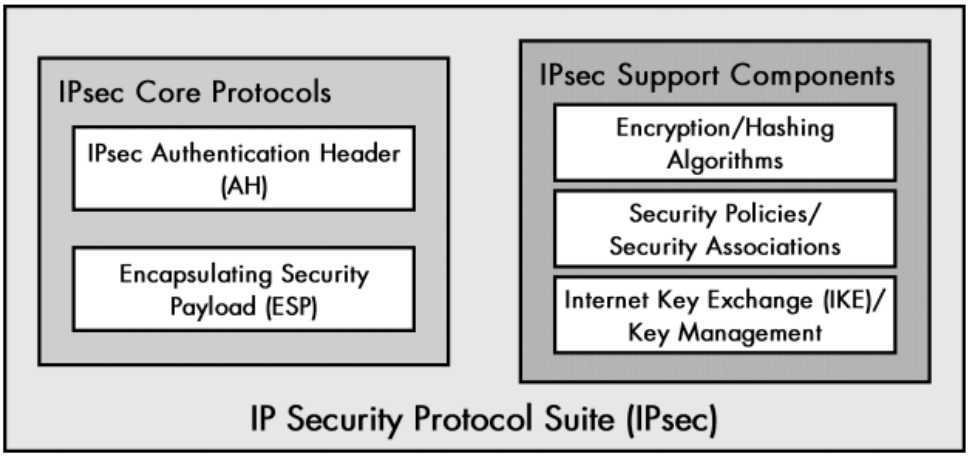


Figure 29-1: Overview of IPsec protocols and components *IPsec consists of two core protocols, AH and ESP, and three supporting components.*

IPsec Support Components

AH and ESP are commonly called *protocols*, though this is another case where the use of this term is debatable. They are not really distinct protocols but are implemented as headers that are inserted into IP datagrams, as you will see. They thus do the "grunt work" of IPsec, and can be used together to provide both authentication and privacy. However, they cannot operate on their own. To function properly, they need the support of several other protocols and services (see Figure 29-1). The most important of these include the following:

Encryption/Hashing Algorithms AH and ESP are generic and do not specify the exact mechanism used for encryption. This gives them the flexibility to work with a variety of such algorithms and to negotiate which one to use as needed. Two common ones used with IPsec are *Message Digest 5 (MD5)* and *Secure Hash Algorithm 1 (SHA-1)*. These are also called *hashing* algorithms because they work by computing a formula called a *hash* based on input data and a key.

Security Policies, Security Associations, and Management Methods Since IPsec provides flexibility in letting different devices decide how they want to implement security, they require some means to keep track of the security relationships between themselves. This is done in IPsec using constructs called *security policies* and *security associations*, and by providing ways to exchange security association information.

Key Exchange Framework and Mechanism For two devices to exchange encrypted information, they need to be able to share keys for unlocking the encryption. They also need a way to exchange security association information. In IPsec, a protocol called the *Internet Key Exchange (IKE)* provides these capabilities.

KEY CONCEPT IPsec consists of a number of different components that work together to provide security services. The two main ones are protocols called the *Authentication Header (AH)* and *Encapsulating Security Payload (ESP)*, which provide authenticity and privacy to IP data in the form of special headers added to IP datagrams.

Well, that's at least a start at providing a framework for understanding what IPsec is all about and how the pieces fit together. You'll examine these components and protocols in more detail as you proceed through this chapter.