

UNIT _ III: Digital Signatures, Authentication Protocols, Digital signature Standard, Authentication Applications, Kerberos, X.509 Directory Authentication Service, Cryptographic Libraries and APIs for Digital Signatures. Email Security: Pretty Good Privacy (PGP) and S/MIME

DIGITAL SIGNATURE

The most important development from the work on public-key cryptography is the digital signature. Message authentication protects two parties who exchange messages from any third party. However, it does not protect the two parties against each other. A digital signature is analogous to the handwritten signature, and provides a set of security capabilities that would be difficult to implement in any other way. It must have the following properties:

- It must verify the author and the date and time of the signature
- It must to authenticate the contents at the time of the signature
- It must be verifiable by third parties, to resolve disputes

Thus, the digital signature function includes the authentication function. A variety of approaches has been proposed for the digital signature function. These approaches fall into two categories: direct and arbitrated.

Direct Digital Signature

Direct Digital Signatures involve the direct application of public-key algorithms involving only the communicating parties. A digital signature may be formed by encrypting the entire message with the sender's private key, or by encrypting a hash code of the message with the sender's private key. Confidentiality can be provided by further encrypting the entire message plus signature using either public or private key schemes. It is important to perform the signature function first and then an outer confidentiality function, since in case of dispute, some third party must view the message and its signature. But these approaches are dependent on the security of the sender's private-key. Will have problems if it is lost/stolen and signatures forged. Need time-stamps and timely key revocation.

Arbitrated Digital Signature

The problems associated with direct digital signatures can be addressed by using an arbiter, in a variety of possible arrangements. The arbiter plays a sensitive and crucial role in this sort of scheme, and all parties must have a great deal of trust that the arbitration mechanism is working properly. These schemes can be implemented with either private or public-key algorithms, and the arbiter may or may not see the actual message contents. **Using Conventional encryption**

$$X \rightarrow A: M \parallel E(K_{xa}, [ID_x \parallel H(M)])$$

$$A \rightarrow Y: E(K_{ay}, [ID_x \parallel M \parallel E(K_{xa}, [ID_x \parallel H(M)])] \parallel T)$$

It is assumed that the sender X and the arbiter A share a secret key K_{xa} and that A and Y share secret key K_{ay} . X constructs a message M and computes its hash value $H(m)$. Then X transmits the message plus a signature to A. the signature consists of an identifier ID_x of X plus the hash value, all encrypted using K_{xa} .

A decrypts the signature and checks the hash value to validate the message. Then A transmits a message to Y, encrypted with K_{ay} . The message includes ID_x , the original message from X, the signature, and a timestamp.

Arbiter sees message

Problem : the arbiter could form an alliance with sender to deny a signed message, or with the receiver to forge the sender's signature.

Using Public Key Encryption

$$X \rightarrow A: ID_x \parallel E(PR_x, [ID_x \parallel E(PU_y, E(PR_x, M))])$$

$$A \rightarrow Y: E(PR_a, [ID_x \parallel E(PU_y, E(PR_x, M)) \parallel T])$$

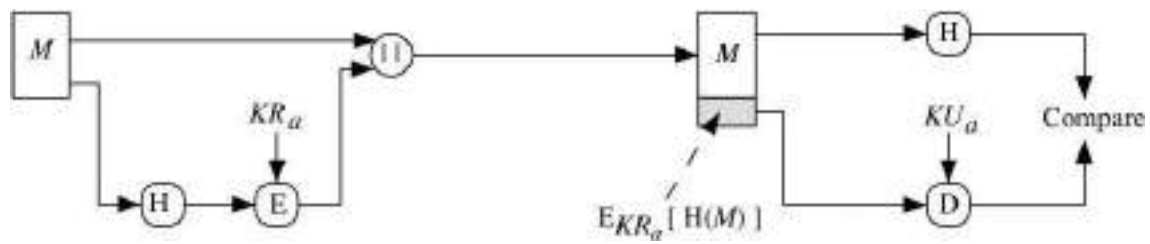
X double encrypts a message M first with X's private key, PR_x , and then with Y's public key, PU_y . This is a signed, secret version of the message. This signed message, together with X's identifier, is encrypted again with PR_x and, together with ID_x , is sent to A. The inner, double encrypted message is secure from the arbiter (and everyone else except Y)

A decrypts the outer encryption to assure that the message must have come from X (because only X has PR_x). Then A transmits a message to Y, encrypted with PR_a . The message includes ID_x , the double encrypted message, and a timestamp.

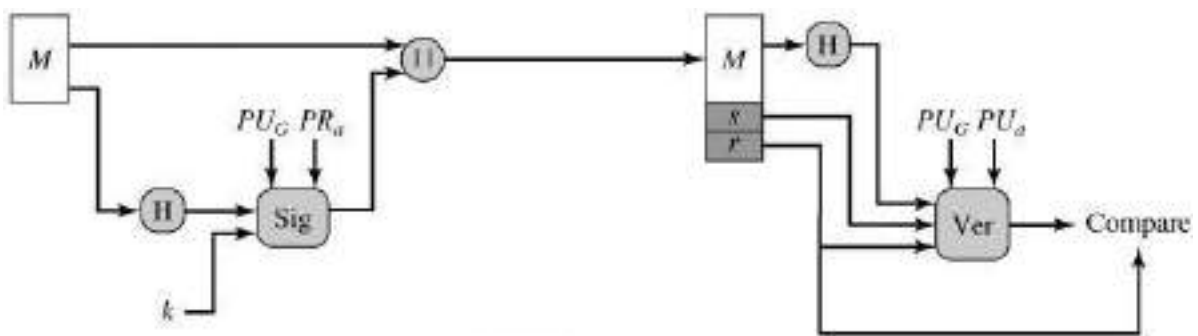
Arbiter does not see message

Digital Signature Standard (DSS)

The National Institute of Standards and Technology (NIST) has published Federal Information Processing Standard FIPS 186, known as the Digital Signature Standard (DSS). The DSS makes use of the Secure Hash Algorithm (SHA) and presents a new digital signature technique, the Digital Signature Algorithm (DSA). The DSS uses an algorithm that is designed to provide only the digital signature function and cannot be used for encryption or key exchange, unlike RSA. The RSA approach is shown below. The message to be signed is input to a hash function that produces a secure hash code of fixed length. This hash code is then encrypted using the sender's private key to form the signature. Both the message and the signature are then transmitted.



The recipient takes the message and produces a hash code. The recipient also decrypts the signature using the sender's public key. If the calculated hash code matches the decrypted signature, the signature is accepted as valid. Because only the sender knows the private key, only the sender could have produced a valid signature. The DSS approach also makes use of a hash function. The hash code is provided as input to a signature function along with a random number k generated for this particular signature. The signature function also depends on the sender's private key (PR_a) and a set of parameters known to a group of communicating principals. We can consider this set to constitute a global public key (PUG). The result is a signature consisting of two components, labeled s and r .



(b) DSS approach

At the receiving end, the hash code of the incoming message is generated. This plus the signature is input to a verification function. The verification function also depends on the global public key as well as the sender's public key (PU_a), which is paired with the sender's private key. The output of the verification function is a value that is equal to the signature component r if the signature is valid. The signature function is such that only the sender, with knowledge of the private key, could have produced the valid signature.

AUTHENTICATION APPLICATIONS

KERBEROS

Kerberos is an authentication service developed as part of Project Athena at MIT. It addresses the threats posed in an open distributed environment in which users at workstations wish to access services on servers distributed throughout the network. Some of these threats are:

- A user may gain access to a particular workstation and pretend to be another user

operating from that workstation.

- A user may alter the network address of a workstation so that the requests sent from the altered workstation appears to come from the impersonated workstation.
- A user may eavesdrop on exchanges and use a replay attack to gain entrance to a server or to disrupt operations.

Two versions of Kerberos are in current use: Version-4 and Version-5. The first published report on Kerberos 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.

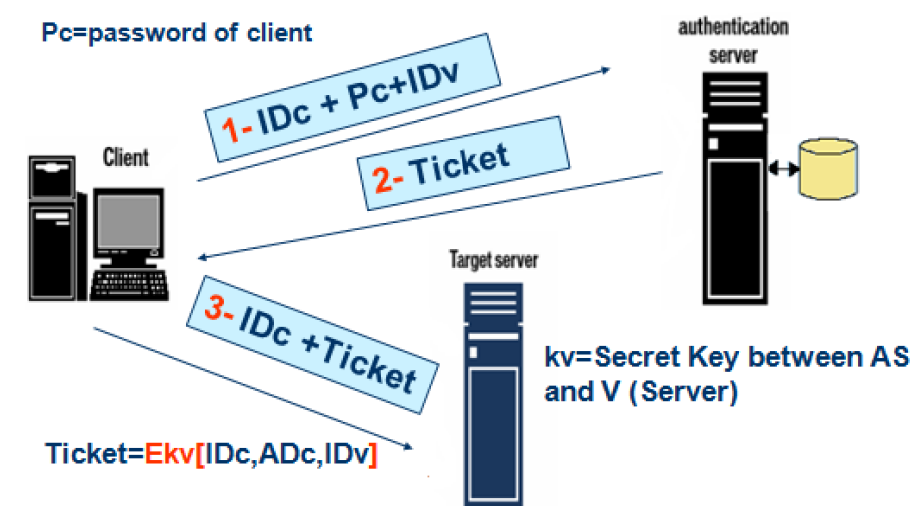
Transparent: Ideally, the user should not be aware that authentication is taking place, beyond the requirement to enter a password.

Scalable: The system should be capable of supporting large numbers of clients and servers. This suggests a modular, distributed architecture

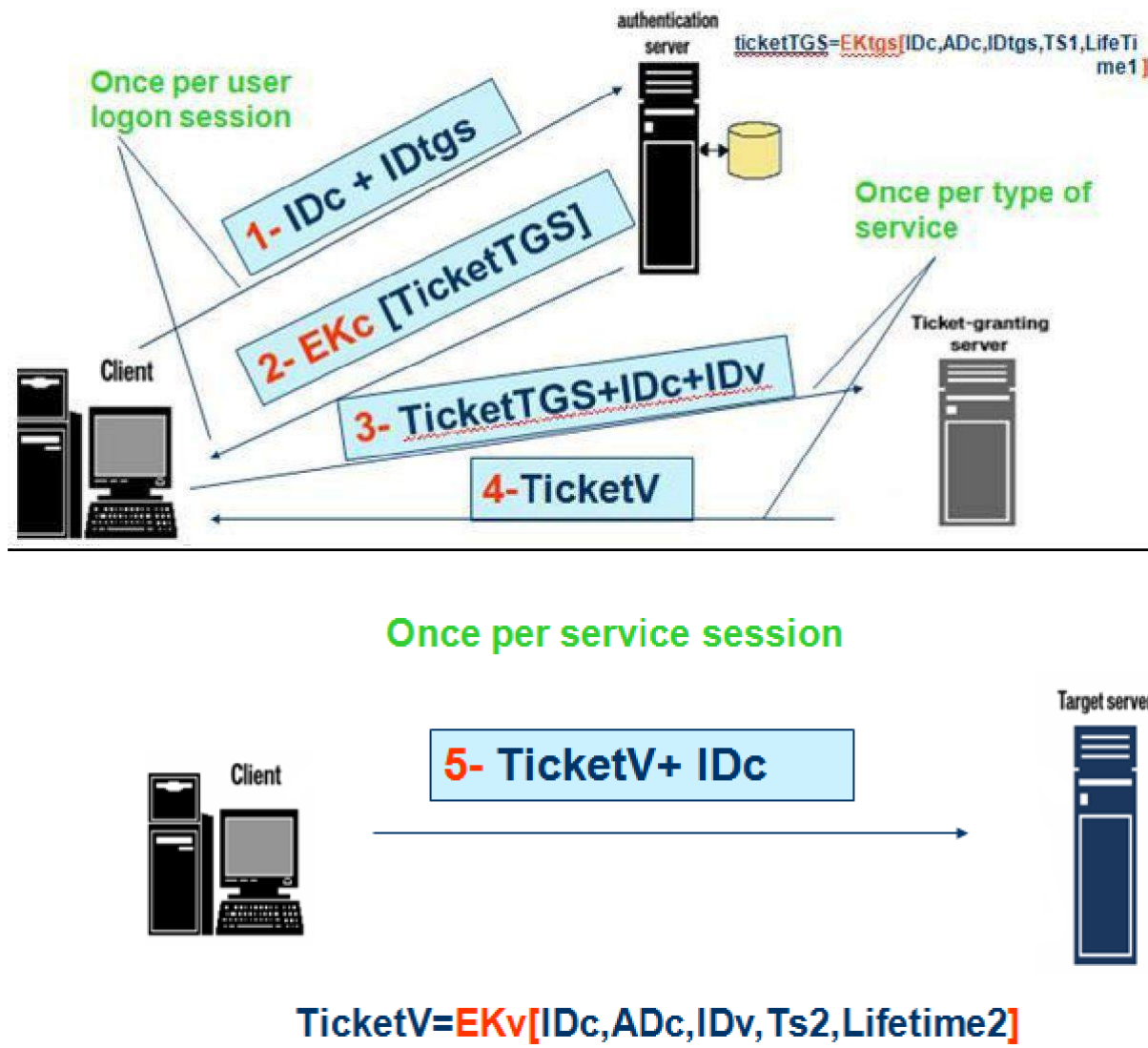
Two versions of Kerberos are in common use: Version 4 is most widely used version. Version 5 corrects some of the security deficiencies of Version 4. Version 5 has been issued as a draft Internet Standard (RFC 1510)

KERBEROS VERSION 4

1.) SIMPLE DIALOGUE:



MORE SECURE DIALOGUE



The Version 4 Authentication Dialogue The full Kerberos v4 authentication dialogue is shown here divided into 3 phases.

(1) $C \rightarrow AS \ ID_c \parallel ID_{TGS} \parallel TS_1$

(2) $AS \rightarrow C \ E(K_{c,TGS}, [K_{c,TGS} \parallel ID_{TGS} \parallel TS_2 \parallel Lifetime_2 \parallel Ticket_{TGS}])$

$Ticket_{TGS} = E(K_{TGS}, [K_{c,TGS} \parallel ID_C \parallel AD_C \parallel ID_{TGS} \parallel TS_2 \parallel Lifetime_2])$

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

(3) $C \rightarrow TGS \ ID_v \parallel Ticket_{TGS} \parallel Authenticator_c$

(4) $TGS \rightarrow C \ E(K_{c,TGS}, [K_{c,v} \parallel ID_v \parallel TS_4 \parallel Ticket_v])$

$Ticket_{TGS} = E(K_{TGS}, [K_{c,TGS} \parallel ID_C \parallel AD_C \parallel ID_{TGS} \parallel TS_2 \parallel Lifetime_2])$

$Ticket_v = E(K_v, [K_{c,v} \parallel ID_C \parallel AD_C \parallel ID_v \parallel TS_4 \parallel Lifetime_4])$

$Authenticator_c = E(K_{c,TGS}, [ID_C \parallel AD_C \parallel TS_3])$

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

(5) $C \rightarrow V \ Ticket_v \parallel Authenticator_c$

(6) $V \rightarrow C \ E(K_{c,v}, [TS_5 + 1])$ (for mutual authentication)

$Ticket_v = E(K_v, [K_{c,v} \parallel ID_C \parallel AD_C \parallel ID_v \parallel TS_4 \parallel Lifetime_4])$

$Authenticator_c = E(K_{c,v}, [ID_C \parallel AD_C \parallel TS_5])$

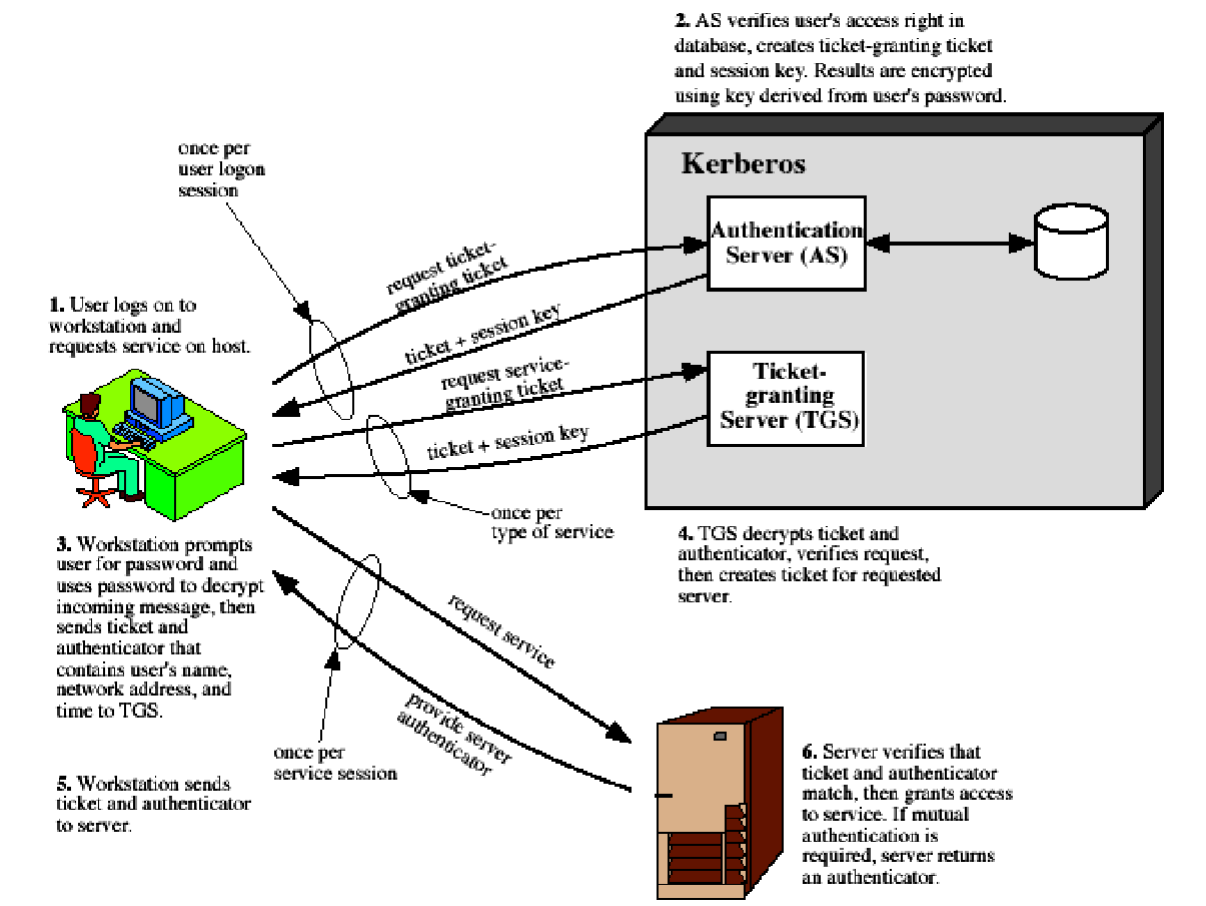
(c) Client/Server Authentication Exchange to obtain service

There is a 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. An efficient way of doing this is to use a session encryption key to secure information.

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. 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. C then sends the TGS a message that includes the ticket plus the ID of the requested service (message 3). The reply from the TGS, in message (4), follows the form of message (2). 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).

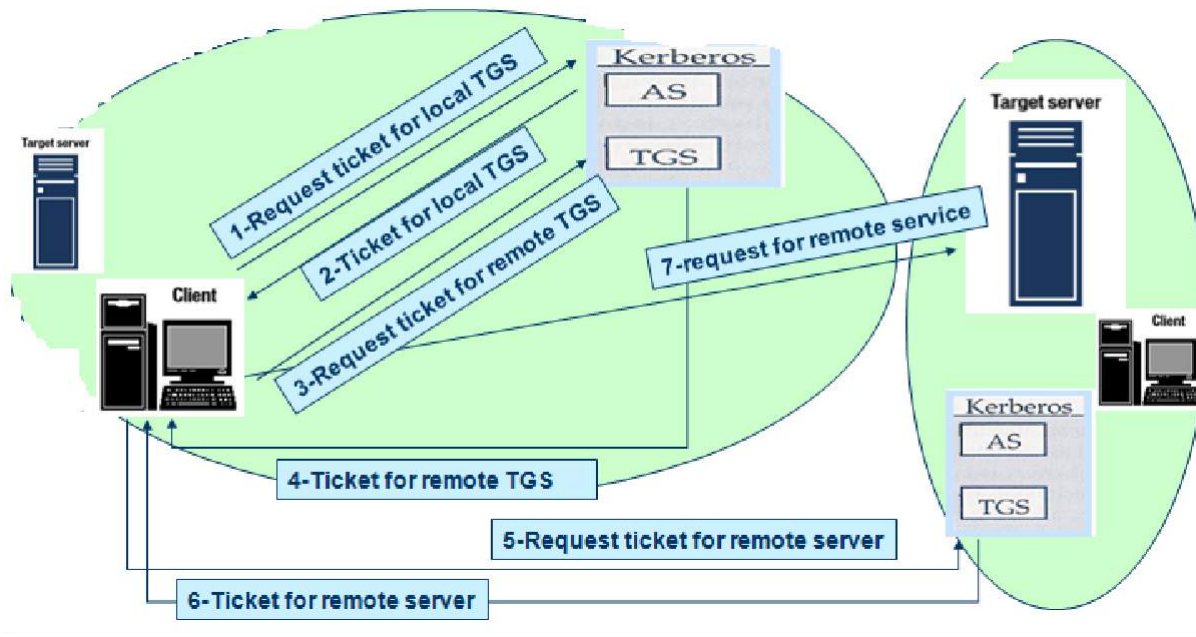
Overview of Kerberos



Kerberos Realms A full-service Kerberos environment consisting of a Kerberos server, a number of clients, and a number of application servers is referred to as a Kerberos realm. A Kerberos realm is a set of managed nodes that share the same Kerberos database, and are part of the same administrative domain. If have multiple realms, their Kerberos servers must share keys and trust each other.

The following figure shows the authentication messages where service is being requested from another domain. 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, as each pair of realms need to share a key.

Request for Service in another realm:



The limitations of Kerberos version-4 are categorised into two types:

Environmental shortcomings of Version 4:

- Encryption system dependence: DES
- Internet protocol dependence
- Ticket lifetime
- Authentication forwarding
- Inter-realm authentication

Technical deficiencies of Version 4:

- Double encryption
- Session Keys
- Password attack

KERBEROS VERSION 5

Kerberos Version 5 is specified in RFC 1510 and provides a number of improvements over version 4 in the areas of environmental shortcomings and technical deficiencies. It includes some new elements such as:

Realm: Indicates realm of the user

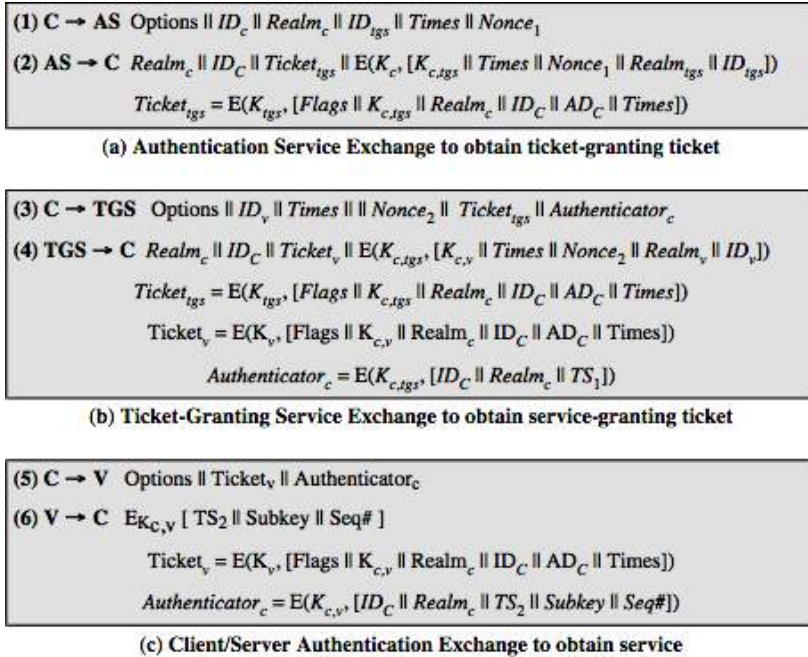
Options

Times

- From: the desired start time for the ticket
- Till: the requested expiration time
- Rtime: requested renew-till time

Nonce: A random value to assure the response is fresh

The basic Kerberos version 5 authentication dialogue is shown here First, consider the **authentication service exchange**.



Message (1) is a client request for a ticket-granting ticket. 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. Now compare the **ticket-granting service** exchange for versions 4 and 5. 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), returning a ticket plus information needed by the client, the latter encrypted with 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, such as a request for mutual authentication. If required, the server responds with message (6) that includes the timestamp from the authenticator. The flags field included in tickets in version 5 supports expanded functionality compared to that available in version 4.

Advantages of Kerberos:

Users passwords are never sent across the network, encrypted or in plain text

Secret keys are *only* passed across the network in encrypted form

Client and server systems mutually authenticate limits

It limits the duration of their users' authentication. Authentications

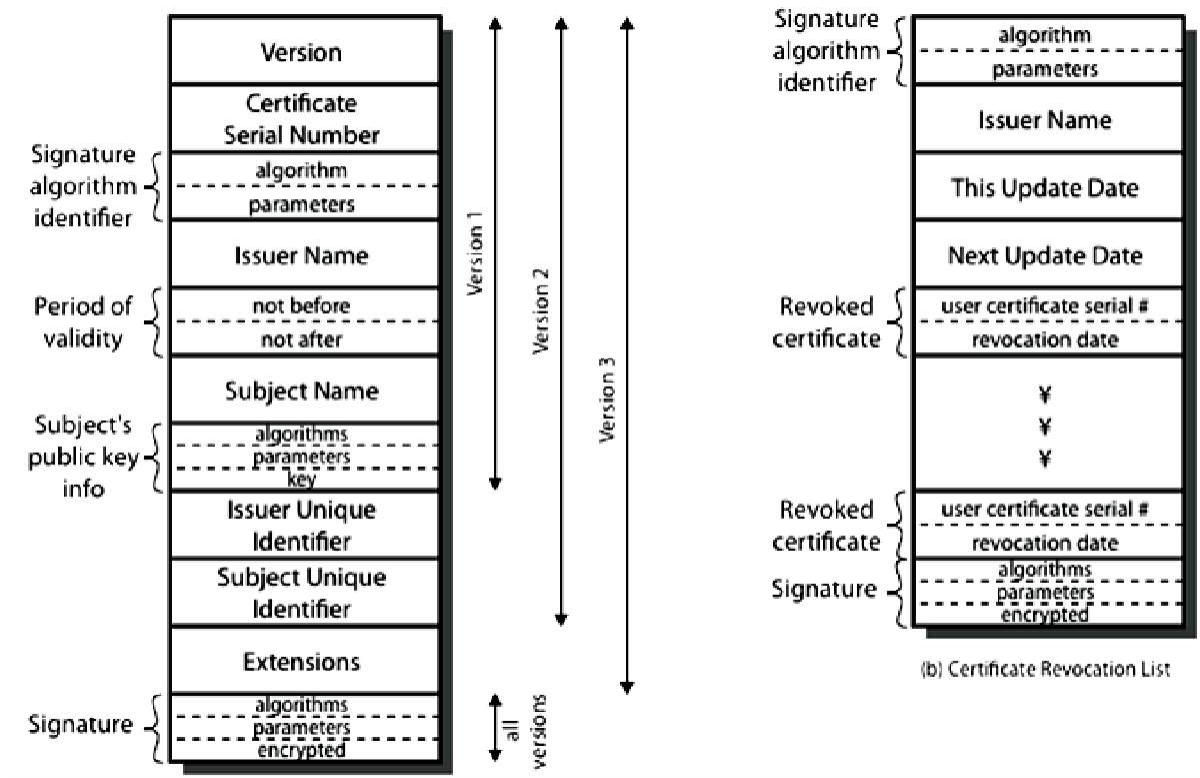
are **reusable** and **durable**

Kerberos has been scrutinized by many of the top programmers, cryptologists and security experts in the industry

X.509 AUTHENTICATION SERVICE

ITU-T recommendation X.509 is part of the X.500 series of recommendations that define a directory service. The directory is, in effect, a server or distributed set of servers that maintains a database of information about users. The information includes a mapping from user name to network address, as well as other attributes and information about the users. X.509 is based on the use of public-key cryptography and digital signatures. 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. The directory server itself is not responsible for the creation of public keys or for the certification function; it merely provides an easily accessible location for users to obtain certificates. The general format of a certificate is shown above, which includes the following elements:

- version 1, 2, or 3
- serial number (unique within CA) identifying certificate
- signature algorithm identifier
- 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)

The standard uses the following notation to define a certificate:

$$CA\langle\langle A \rangle\rangle = CA \{V, SN, AI, CA, TA, A, Ap\}$$

Where $Y\langle\langle X \rangle\rangle$ = 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

User certificates generated by a CA have the following characteristics:

- Any user with CA's public key can verify the user public key that was certified
- No party other than the CA can modify the certificate without being detected because they cannot be forged, certificates can be placed in a public directory

Scenario: Obtaining a User Certificate If both users share a common CA then they are assumed to know its public key. Otherwise, CA's must form a hierarchy and use certificates linking members of hierarchy to validate other CA's. Each CA has certificates for clients (forward) and parent (backward). Each client trusts parents' certificates. It enables verification of any certificate from one CA by users of all other CA's in hierarchy. A has obtained a certificate from the CA X1. B has obtained a certificate from the CA X2. A can read the B's certificate but cannot verify it. In order to

solve the problem, the Solution:

- $X1 \ll X2 \gg X2 \ll B \gg$.
- A obtain the certificate of X2 signed by X1 from directory.
- obtain X2's public key.
- A goes back to directory and obtain the certificate of B signed by X2.
- obtain B's public key securely.

The directory entry for each CA includes two types of certificates:

- Forward certificates: Certificates of X generated by other CAs
- Reverse certificates: Certificates generated by X that are the certificates of other CAs

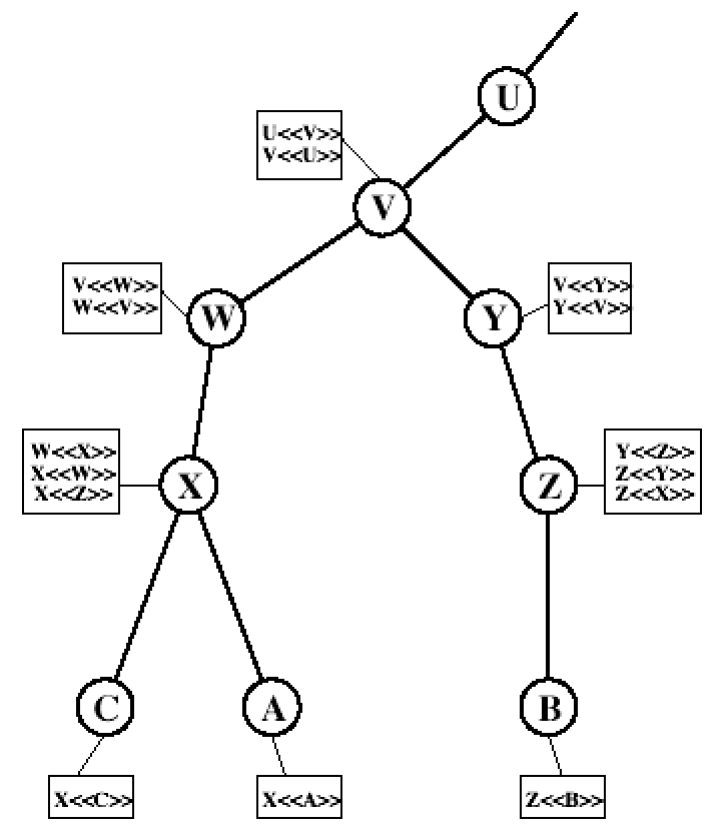
X.509 CA Hierarchy

A acquires B certificate using chain:

$$X \ll W \gg W \ll V \gg V \ll Y \gg Y \ll Z \gg Z \ll B \gg$$

B acquires A certificate using chain:

$$Z \ll Y \gg Y \ll V \gg V \ll W \gg W \ll X \gg X \ll A \gg$$



Revocation of Certificates Typically, a new certificate is issued just before the expiration of the old one. In addition, it may be desirable on occasion to revoke a certificate before it expires, for one of the following reasons:

- The user's private key is assumed to be compromised.
The user is no longer certified by this CA.
- The CA's certificate is assumed to be compromised.

Each CA must maintain a list consisting of all revoked but not expired certificates issued by that CA, including both those issued to users and to other CAs. These lists should also be posted on the directory. Each **certificate revocation list (CRL) posted** to the directory is signed by the issuer and includes 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.

AUTHENTICATION PROCEDURES

X.509 also includes three alternative authentication procedures that are intended for use across a variety of applications. All these procedures make use of public-key signatures. It is assumed that the two parties know each other's public key, either by obtaining each other's certificates from the directory or because the certificate is included in the initial message from each side. 1. One-Way Authentication: One way authentication involves a single transfer of information from one user (A) to another (B), and establishes the details shown above. Note that only the identity of the initiating entity is verified in this process, not that of the responding entity. At a minimum, the message includes a timestamp, a nonce, and the identity of B and is signed with A's private key. The message may also include information to be conveyed, such as a session key for B.



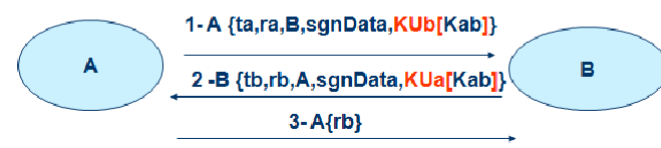
Two-Way Authentication: Two-way authentication thus permits both parties in a communication to verify the identity of the other, thus additionally establishing the above details. The reply message includes the nonce from A, to validate the reply. It also includes a timestamp and nonce generated by B, and possible additional information for A.

- 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



Three-Way Authentication: Three-Way Authentication includes a final message from A to B, which contains a signed copy of the nonce, so that timestamps need not be checked, for use when synchronized clocks are not available.

- 3 messages (A->B, B->A, A->B) which enables above authentication without synchronized clocks



PRETTY GOOD PRIVACY

In virtually all distributed environments, electronic mail is the most heavily used network-based application. But current email services are roughly like "postcards", anyone who wants could pick it up and have a look as it's in transit or sitting in the recipients mailbox. PGP provides a confidentiality and authentication service that can be used for electronic mail and file storage applications. With the explosively growing reliance on electronic mail for every conceivable purpose, there grows a demand for authentication and confidentiality services. The Pretty Good Privacy (PGP) secure email program, is a remarkable phenomenon, has grown explosively and is now widely used. Largely the effort of a single person, Phil Zimmermann, who selected the best available crypto algorithms to use & integrated them into a single program, PGP provides a confidentiality and authentication service that can be used for electronic mail and file storage applications. It is independent of government organizations and runs on a wide range of systems, in both free & commercial versions. There are **five** important services in PGP:

- Authentication (Sign/Verify)
- Confidentiality (Encryption/Decryption)
- Compression
- Email compatibility

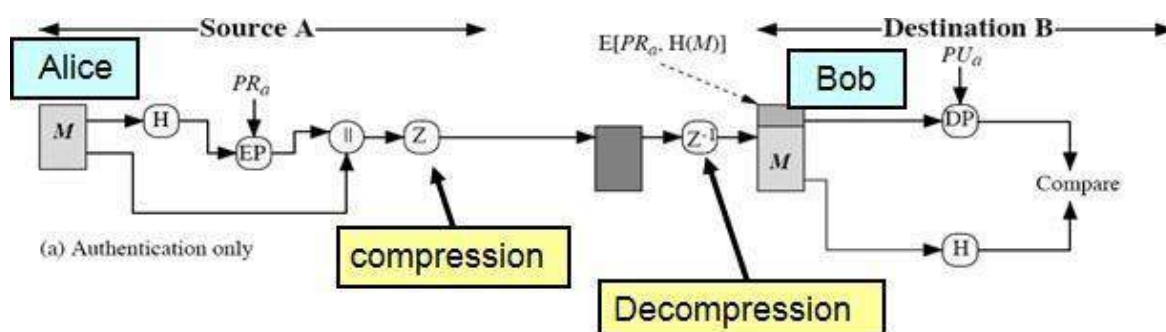
- Segmentation and Reassembly

The last three are **transparent** to the user

PGP Notations:

K_s	=session key used in symmetric encryption scheme
PR_a	=private key of user A, used in public-key encryption scheme
PU_a	=public key of user A, used in public-key encryption scheme
EP	= public-key encryption
DP	= public-key decryption
EC	= symmetric encryption
DC	= symmetric decryption
H	= hash function
	= concatenation
Z	= compression using ZIP algorithm
R64	= conversion to radix 64 ASCII format

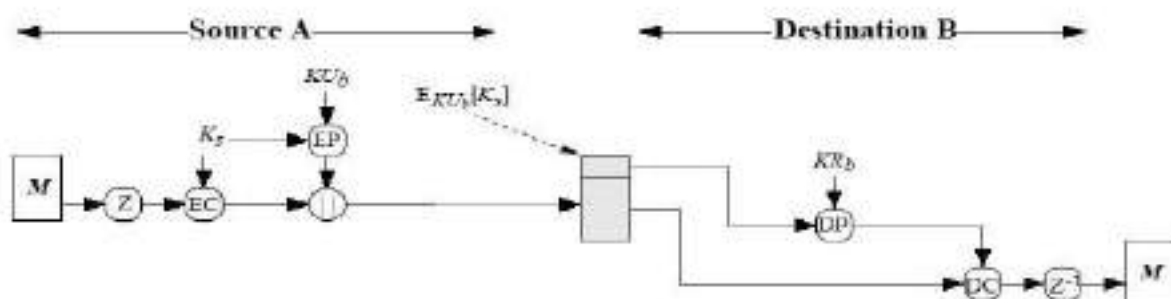
PGP Operation- Authentication



1. sender creates message
2. use SHA-1 to generate 160-bit hash of message
3. signed hash with RSA using sender's private key, and is attached to message
4. receiver uses RSA with sender's public key to decrypt and recover hash code
5. receiver verifies received message using hash of it and compares with decrypted hashcode

PGP Operation- Confidentiality

PGP Operation- Confidentiality



Sender:

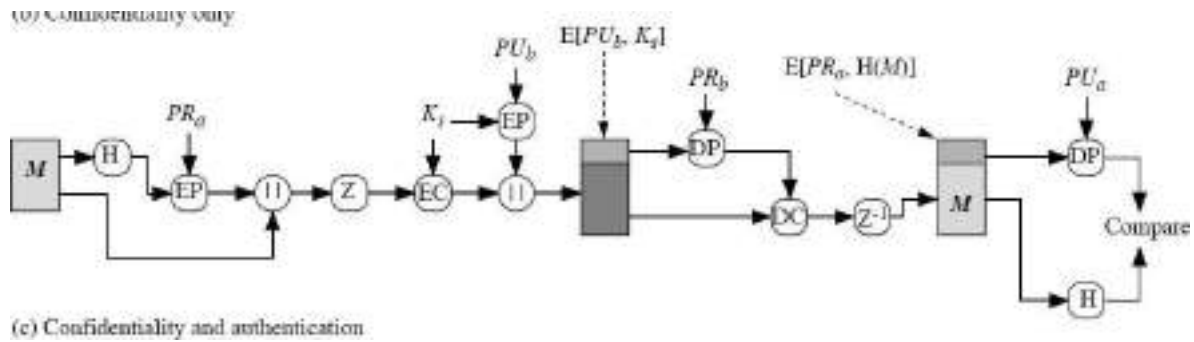
1. Generates message and a random number (session key) only for this message
2. Encrypts message with the session key using AES, 3DES, IDEA or CAST-128
3. Encrypts session key itself with recipient's public key using RSA
4. Attaches it to message

Receiver:

1. Recovers session key by decrypting using his private key
2. Decrypts message using the session key

Confidentiality service provides no assurance to the receiver as to the identity of sender (i.e. no authentication). Only provides confidentiality for sender that only the recipient can read the message (and no one else)

PGP Operation – Confidentiality & Authentication



can use both services on same message to create signature & attach to message
 encrypt both message & signature
 attach RSA/ElGamal encrypted session key
 is called **authenticated confidentiality**

PGP Operation – Compression

As a default, PGP compresses the message after applying the signature but before encryption. This has the benefit of saving space both for e-mail transmission and for file storage. The placement of the compression algorithm, indicated by Z for compression and Z-1 for decompression is critical. The compression algorithm used is ZIP.

The signature is generated before compression for two reasons:

1. so that one can store only the uncompressed message together with signature for later verification
2. Applying the hash function and signature after compression would constrain all PGP implementations to the same version of the compression algorithm as the PGP compression algorithm is not deterministic

Message encryption is applied after compression to strengthen cryptographic security. Because the compressed message has less redundancy than the original plaintext, cryptanalysis is more difficult.

PGP Operation – Email Compatibility

When PGP is used, at least part of the block to be transmitted is encrypted, and thus consists of a stream of arbitrary 8-bit octets. However many electronic mail systems only permit the use of ASCII text. To accommodate this restriction, PGP provides the service

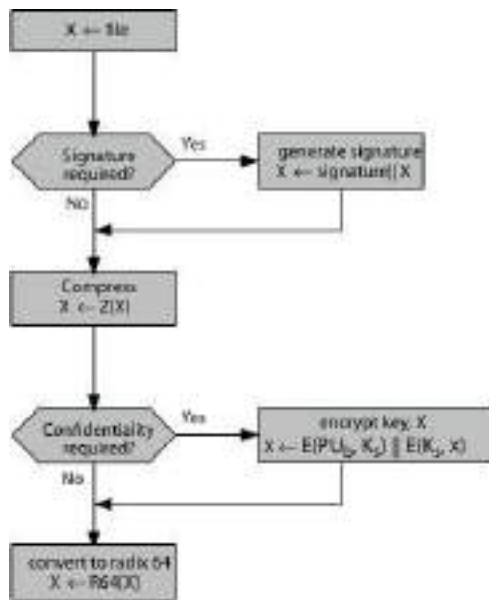
of converting the raw 8-bit binary stream to a stream of printable ASCII characters. It uses radix-64 conversion, in which each group of three octets of binary data is mapped into four ASCII characters. This format also appends a CRC to detect transmission errors. The use of radix 64 expands a message by 33%, but still an overall compression of about one-third can be achieved.

PGP Operation - Segmentation/Reassembly

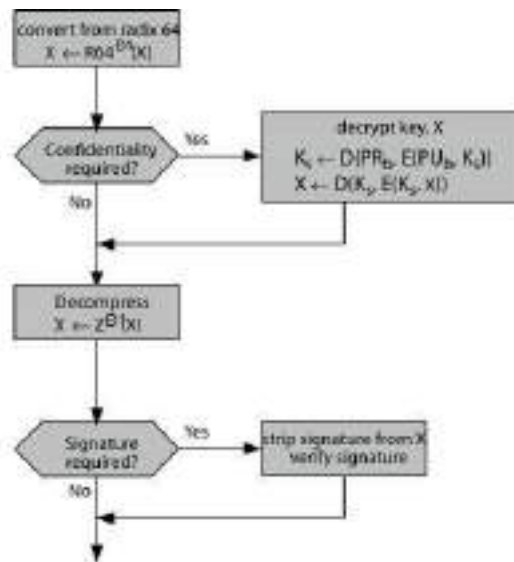
E-mail facilities often are restricted to a maximum message length. For example, many of the facilities accessible through the Internet impose a maximum length of 50,000 octets. Any message longer than that must be broken up into smaller segments, each of which is mailed separately. To accommodate this restriction, PGP automatically subdivides a message that is too large into segments that are small enough to send via e-mail. The segmentation is done after all of the other processing, including the radix-64 conversion. Thus, the session key component and signature component appear only once, at the beginning of the first segment. Reassembly at the receiving end is required before verifying signature or decryption.

PGP Operations – Summary

Function	Algorithms Used	Description
Digital signature	DSS/SHA or RSA/SHA	A hash code of a message is created using SHA-1. This message digest is encrypted using DSS or RSA with the sender's private key, and included with the message.
Message encryption	CAST or IDEA or Three-key Triple DES with Diffie-Hellman or RSA	A message is encrypted using CAST-128 or IDEA or 3DES with a one-time session key generated by the sender. The session key is encrypted using Diffie-Hellman or RSA with the recipient's public key, and included with the message.
Compression	ZIP	A message may be compressed, for storage or transmission, using ZIP.
Email compatibility	Radix 64 conversion	To provide transparency for email applications, an encrypted message may be converted to an ASCII string using radix 64 conversion.
Segmentation	—	To accommodate maximum message size limitations, PGP performs segmentation and reassembly.



(a) Generic Transmission Diagram (from A)



(b) Generic Reception Diagram (to B)

PGP Message Format

A message consists of three components: the message component, a signature (optional), and a session key component (optional). The *message component* includes the actual data to be stored or transmitted, as well as a filename and a timestamp that specifies the time of creation. The *signature component* includes the following:

Timestamp: The time at which the signature was made.

Message digest: The 160-bit SHA-1 digest, encrypted with the sender's private signature key.

Leading two octets of message digest: To enable the recipient to determine if the correct public key was used to decrypt the message digest for authentication, by comparing this plaintext copy of the first two octets with the first two octets of the decrypted digest. These octets also serve as a 16-bit frame check sequence for the message.

Key ID of sender's public key: Identifies the public key that should be used to decrypt the message digest, and, hence, identifies the private key that was used to encrypt the message digest.

The sending PGP entity performs the following steps:

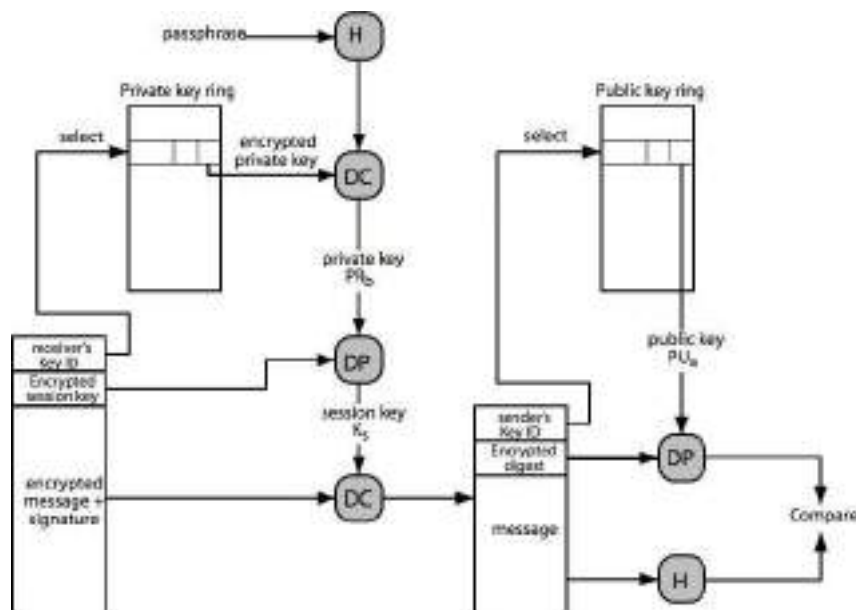
Signing the message

- PGP retrieves the sender's private key from the private-key ring using your_userid as an index. If your_userid was not provided in the command, the first private key on the ring is retrieved.
- PGP prompts the user for the passphrase to recover the unencrypted private key.
- The signature component of the message is constructed

Encrypting the message

- PGP generates a session key and encrypts the message.
- PGP retrieves the recipient's public key from the public-key ring using her_userid as an index.
- The session key component of the message is constructed.

Message Reception



The receiving PGP entity performs the following steps:

Decrypting the message

- PGP retrieves the receiver's private key from the private-key ring, using the Key ID field in the session key component of the message as an index.
- PGP prompts the user for the passphrase to recover the unencrypted private key.
- PGP then recovers the session key and decrypts the message.

Authenticating the message

- a. PGP retrieves the sender's public key from the public-key ring, using the Key ID field in the signature key component of the message as an index.
- b. PGP recovers the transmitted message digest.
- c. PGP computes the message digest for the received message and compares it to the transmitted message digest to authenticate.

S/MIME

S/MIME (Secure/Multipurpose Internet Mail Extension) is a security enhancement to the MIME Internet e-mail format standard, which in turn provided support for varying content types and multipart messages over the text only support in the original Internet RFC822 email standard. MIME allows encoding of binary data to textual form for transport over traditional RFC822 email systems. S/MIME is defined in a number of documents, most importantly RFCs 3369, 3370, 3850 and 3851 and S/MIME support is now included in many modern mail agents.

RFC 822

RFC 822 defines a format for text messages that are sent using electronic mail and it has been the standard for Internet-based text mail message. The overall structure of a message that conforms to RFC 822 is very simple. A message consists of some number of header lines (the header) followed by unrestricted text (the body). The header is separated from the body by a blank line. A header line usually consists of a keyword, followed by a colon, followed by the keyword's arguments; the format allows a long line to be broken up into several lines. The most frequently used keywords are *From*, *To*, *Subject*, and *Date*.

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. **Problems with RFC 822 and SMTP**

- Executable files or other binary objects must be converted into ASCII. Various schemes exist (e.g., Unix UUencode), but a standard is needed

- Text data that includes special characters (e.g., Hungarian text) cannot be transmitted as SMTP is limited to 7-bit ASCII
- Some servers reject mail messages over a certain size
- Some common problems exist with the SMTP implementations which do not adhere completely to the SMTP standards defined in RFC 821. They are:

delete, add, or reorder CR and LF characters truncate or wrap lines longer than 76 characters remove trailing white space (tabs and spaces) pad lines in a message to the same length convert tab characters into multiple spaces

MIME is intended to resolve these problems in a manner that is compatible with existing RFC 822 implementations and the specification is provided in RFC's 2045 through 2049.

The MIME specification includes the following elements:

1. Five new message header fields are defined, which 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 protect the content from alteration by the mail system.

MIME - New header fields 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 that the receiving user agent can pick an appropriate agent or mechanism to represent the data to the user or otherwise deal with the data in an appropriate manner.
- *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. There are seven different major types of content and a total of 15 subtypes. In general, a content type declares the general type of data, and the subtype specifies a particular format for that type of data. For the text type of body, the primary subtype is plain text, which is simply a string of ASCII characters or ISO 8859 characters. The enriched subtype allows greater formatting flexibility. The multipart type indicates that the body contains multiple, independent parts. The Content-Type header field includes a parameter called boundary that defines the delimiter between body parts. This boundary should not appear in any parts of the message. Each boundary starts on a new line and consists of two hyphens followed by the boundary value. The final boundary, which indicates the end of the last part, also has a suffix of two hyphens. Within each part, there may be an optional ordinary MIME header. There are four subtypes of the multipart type, all of which have the same overall syntax.

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 "best" version to the user.
	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.
	gif	The image is in GIF format.
Video	mpeg	MPEG format.
Audio	Basic	Single-channel 8-bit 150K mu-law encoding at a sample rate of 8 kHz.
Application	PostScript	Adobe Postscript.
	octet-stream	General binary data consisting of 8-bit bytes.

The message type provides a number of important capabilities in MIME. The message/rfc822 subtype indicates that the body is an entire message, including header

and body. Despite the name of this subtype, the encapsulated message may be not only a simple RFC 822 message, but also any MIME message. The message/partial subtype enables fragmentation of a large message into a number of parts, which must be reassembled at the destination. For this subtype, three parameters are specified in the Content-Type: Message/Partial field: an id common to all fragments of the same message, a sequence number unique to each fragment, and the total number of fragments. The message/external-body subtype indicates that the actual data to be conveyed in this message are not contained in the body. Instead, the body contains the information needed to access the data. The application type refers to other kinds of data, typically either uninterpreted binary data or information to be processed by a mail-based application.

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.

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 humans.
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 MIME standard defines two methods of encoding data. The Content-Transfer-Encoding field can actually take on six values. Three of these values (7bit, 8bit, and binary) indicate that no encoding has been done but provide some information about the nature of the data. 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. Two schemes are defined to provide a choice between a transfer technique that is essentially human

readable and one that is safe for all types of data in a way that is reasonably compact. The quoted-printable transfer encoding is useful when the data consists largely of octets that correspond to printable ASCII characters. In essence, it represents unsafe 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.

Native Form	The body to be transmitted is created in the system's native format. The native character set is used and, where appropriate, local end-of-line conventions are used as well. The body may be a UNIX-style text file, or a Sun raster image, or a VMS indexed file, or audio data in a system-dependent format stored only in memory, or anything else that corresponds to the local model for the representation of some form of information. Fundamentally, the data is created in the "native" form that corresponds to the type specified by the media type.
Canonical Form	The entire body, including "out-of-band" information such as record lengths and possibly file attribute information, is converted to a universal canonical form. The specific media type of the body as well as its associated attributes dictate the nature of the canonical form that is used. Conversion to the proper canonical form may involve character set conversion, transformation of audio data, compression, or various other operations specific to the various media types. If character set conversion is involved, however, care must be taken to understand the semantics of the media type, which may have strong implications for any character set conversion (e.g. with regard to syntactically meaningful characters in a text subtype other than "plain").

S/MIME Functionality

S/MIME has a very similar functionality to PGP. Both offer the ability to sign and/or encrypt messages.

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.