UNIT-III

Content

- Electronic Mail Security
- IP Security

E-mail Security

PGP, S/MIME

Certificates and PKI

E-mail Security

- E-mail is one of the most widely used network services
  - One of the old killer applications of the Internet
- Normally message contents not secured
  - Can be read/modified either in transit or at destination by the attacker
- E-mail service is like postcard service
  - just pick it and read it

Email Security Enhancements

- confidentiality
  - protection from disclosure
- authentication
  - of sender of message
- message integrity
  - protection from modification
- non-repudiation of origin
  - protection from denial by sender

Pretty Good Privacy (PGP)

- widely used secure e-mail software
  - originally a file encryption/decryption facility
- developed by Phil Zimmermann
  - a security activist who has had legal problems due to PGP
- best available crypto algorithms are employed
- available on several platforms with source code
- originally free, now commercial versions exist
- not controlled by a standardization body
  - although there are RFCs

- PGP is a remarkable phenomenon. Largely the effort of a single person, Phil Zimmermann, PGP provides a confidentiality and authentication service that can be used for electronic mail and file storage applications.
- In essence, Zimmermann has done the following:

- Selected the best available cryptographic algorithms as building blocks.
- Integrated 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.
- Made the package and its documentation, including the source code, freely available via the Internet, bulletin boards, and commercial networks such as AOL (America On Line).
- Entered into an agreement with a company (Viacrypt, now Network Associates) 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 platforms, including Windows, UNIX, Macintosh, and many more. In addition, the commercial version satisfies users who want a product that comes with vendor support.

➢ It is based on algorithms that have survived extensive public review and are considered extremely secure. Specifically, the package includes RSA, DSS, and Diffie-Hellman for public-key encryption;CAST-128, IDEA, and 3DES for symmetric encryption; and SHA-1 for hash coding.

- 3. It has a wide range of applicability, from corporations that wish to select and enforce a standardized scheme for encrypting files and messages to individuals who wish to communicate securely with others worldwide over the Internet and other networks.

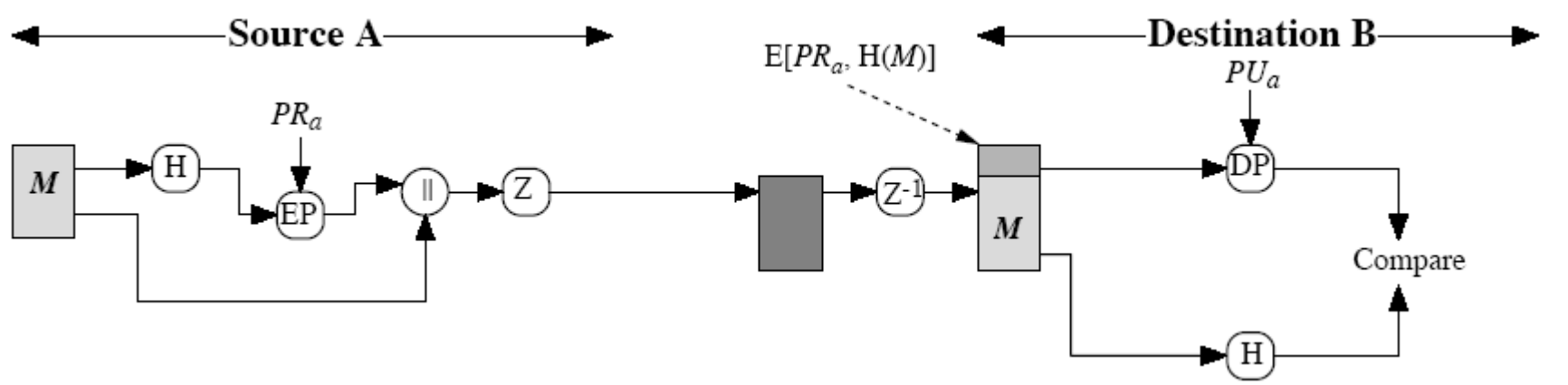
- It was not developed by, nor is it controlled by, any governmental or standards organization. For those with an instinctive distrust of “the establishment,” this makes PGP attractive.
- PGP is now on an Internet standards track (RFC 3156; MIME Security with Open PGP). Nevertheless, PGP still has an aura of an antiestablishment endeavor.

- We begin with an overall look at the operation of PGP. Next, we examine how cryptographic keys are created and stored. Then, we address the vital issue of public-key management.

PGP OPERATIONS

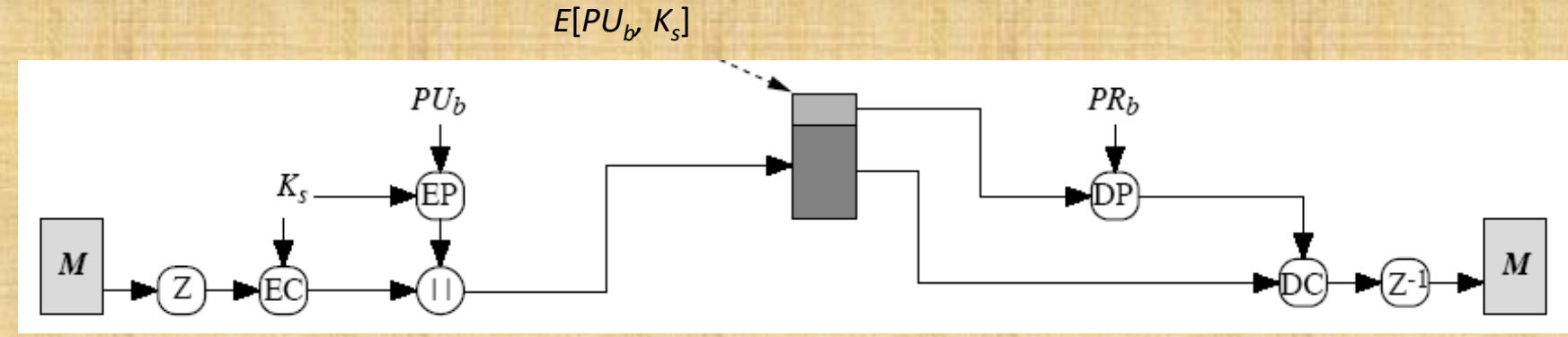
- Digital Signatures (and consequently message authentication and integrity)
  - RSA, DSS
- Message Encryption
  - CAST, IDEA, 3DES, AES (all at least 128 bits)
  - symmetric keys are used once and encrypted using RSA or ElGamal (based on discrete logs)
- Compression using ZIP
- Radix-64 conversion (to ASCII)
  - for e-mail compatibility

PGP Operation – Authentication



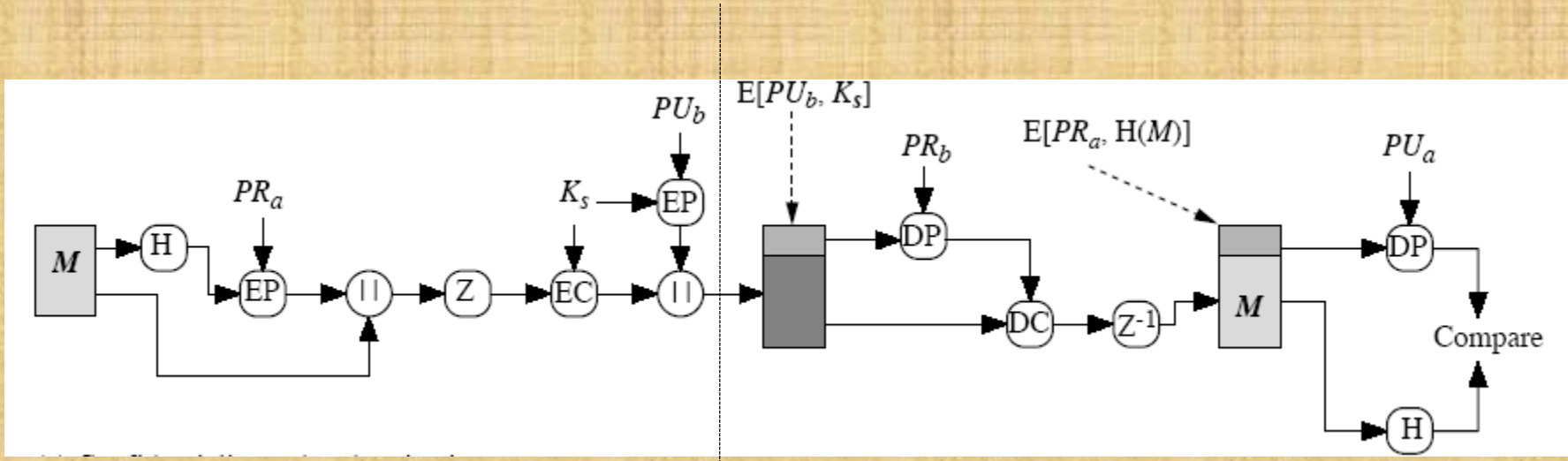
- Classical application of public key crypto
  - This figure is actually for RSA
  - for DSA refer to previous lectures
- Z is zip function
- radix-64 conversion is done after zip at sender, before Z^{-1} at receiver
  - may be done only for signature or for the whole message

PGP Operation – Confidentiality



- One-time session key, K_s
  - generated at random
  - encrypted using a public key cryptosystem, EP
    - RSA or ElGamal
- Message is compressed before encryption
  - This is the default case

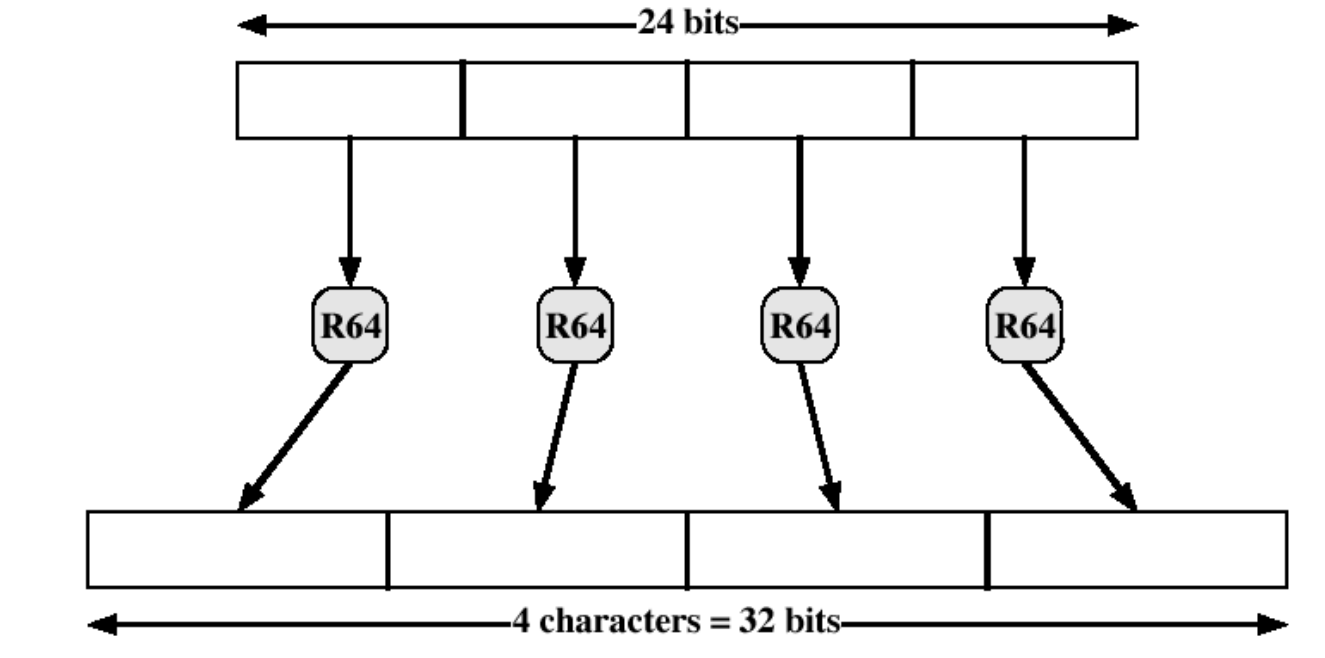
PGP Operation – Confidentiality and Authentication



- uses both services on same message
  - create signature and attach to message
  - compress and encrypt both message & signature
  - attach encrypted session key
  - radix-64 conversion is for everything at the end

PGP Operation – E-mail compatibility radix-64 conversion

- Encrypted text and signatures create binary output
- however email was designed only for text
  - hence PGP must encode raw binary data into printable ASCII characters
- uses radix-64 algorithm (See CS408 notes)
  - maps 3 bytes to 4 printable chars



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.

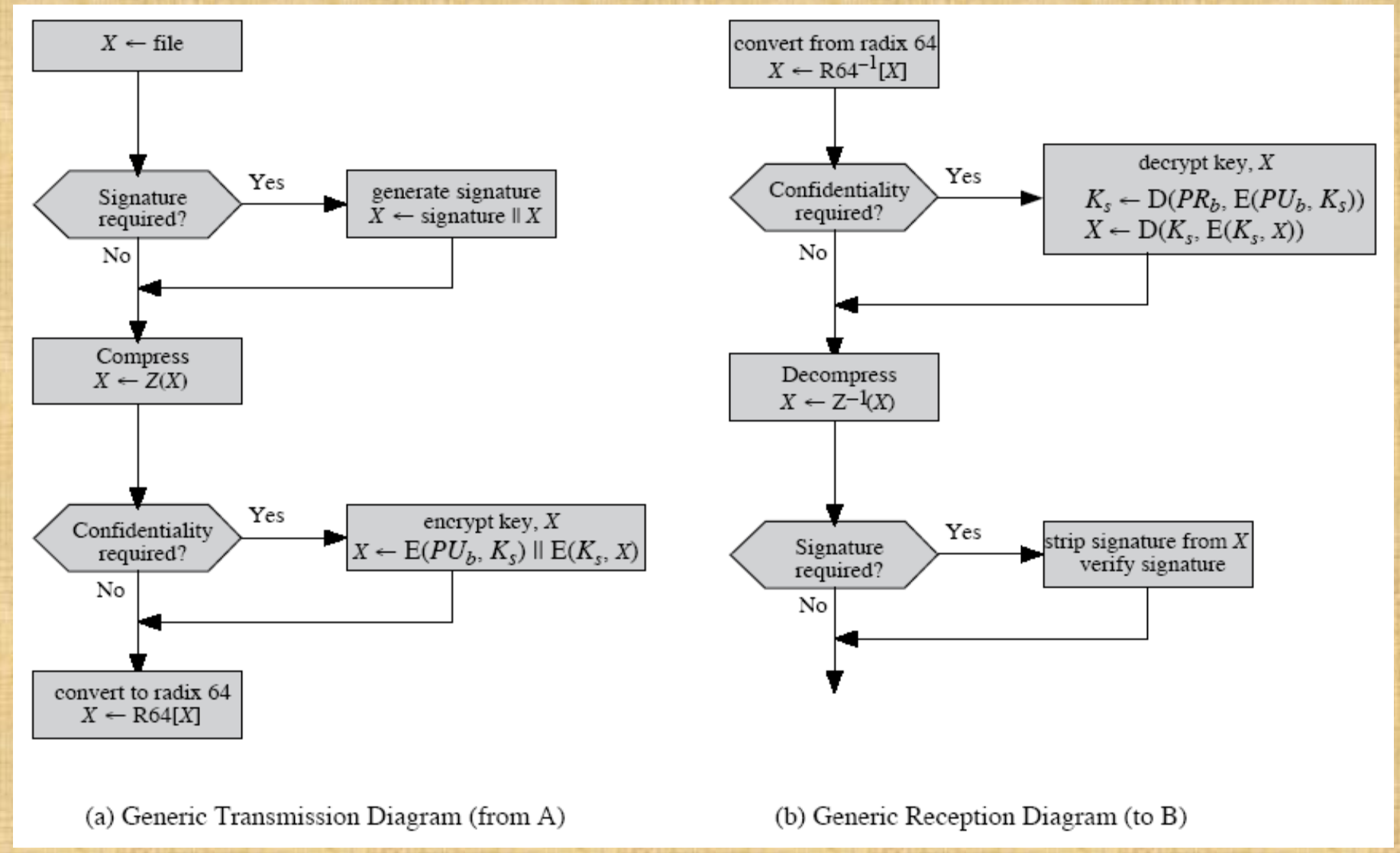
1. The signature is generated before compression for two reasons:

a. It is preferable to sign an uncompressed message so that one can store only the uncompressed message together with the signature for future verification. If one signed a compressed document, then it would be necessary either to store a compressed version of the message for later verification or to recompress the message when verification is required.

b. Even if one were willing to generate dynamically a recompressed message for verification, PGP's compression algorithm presents a difficulty. The algorithm is not deterministic; various implementations of the algorithm achieve different tradeoffs in running speed versus compression ratio and, as a result, produce different compressed forms. However, these different compression algorithms are interoperable because any version of the algorithm can correctly decompress the output of any other version. Applying the hash function and signature after compression would constrain all PGP implementations to the same version of the compression algorithm.

2. 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 – Summary



(a) Generic Transmission Diagram (from A)

(b) Generic Reception Diagram (to B)

Cryptographic Keys and Key Rings

- PGP makes use of four types of keys: one-time session symmetric keys, public keys, private keys, and passphrase-based symmetric keys (explained subsequently).
- Three separate requirements can be identified with respect to these keys.
- **A means of generating unpredictable session keys is needed.**

- We would like to allow a user to have multiple public-key/private-key pairs.
- One reason is that the user may wish to change his or her key pair from time to time.
- When this happens, any messages in the pipeline will be constructed with an obsolete key.
- Furthermore, recipients will know only the old public key until an update reaches them.
- In addition to the need to change keys over time, a user may wish to have multiple key pairs at a given time to interact with different groups of correspondents or simply to enhance security by limiting the amount of material encrypted with any one key.
- The upshot of all this is that there is not a one-to-one correspondence between users and their public keys.
- Thus, some means is needed for identifying particular keys.

- Each PGP entity must maintain a file of its own public/private key pairs as well as a file of public keys of correspondents.

SESSION KEY GENERATION

- Each session key is associated with a single message and is used only for the purpose of encrypting and decrypting that message.
- Recall that message encryption/decryption is done with a symmetric encryption algorithm.
- CAST-128 and IDEA use 128-bit keys; 3DES uses a 168-bit key.
- For the following discussion, we assume CAST-128.
- Random 128-bit numbers are generated using CAST-128 itself.
- The input to the random number generator consists of a 128-bit key and two 64-bit blocks that are treated as plaintext to be encrypted. Using cipher feedback mode, the CAST-128 encrypter produces two 64-bit cipher text blocks, which are concatenated to form the 128-bit session key.
- The algorithm that is used is based on the one specified in ANSI X12.17.

KEY IDENTIFIERS

- As we have discussed, an encrypted message is accompanied by an encrypted form of the session key that was used for message encryption.
- The session key itself is encrypted with the recipient's public key.
- Hence, only the recipient will be able to recover the session key and therefore recover the message.
- If each user employed a single public/private key pair, then the recipient would automatically know which key to use to decrypt the session key: the recipient's unique private key.
- We have stated a requirement that any given user may have **multiple public/private key pairs**.

How, then, does the recipient know which of its public keys was used to encrypt the session key?

- One simple solution would be to transmit the public key with the message.
- The recipient could then verify that this is indeed one of its public keys, and proceed.
- This scheme would work, but it is unnecessarily wasteful of space.
- An RSA public key may be hundreds of decimal digits in length.
- Another solution would be to associate an identifier with each public key that is unique at least within one user.
- That is, the combination of user ID and key ID would be sufficient to identify a key uniquely.
- Then only the much shorter key ID would need to be transmitted.
- This solution, however, raises a management and overhead problem: Key IDs must be assigned and stored so that both sender and recipient could map from key ID to public key.
- This seems unnecessarily burdensome.

- The solution adopted by PGP is to assign a key ID to each public key that is, with very high probability, unique within a user ID.
- The key ID associated with each public key consists of its least significant 64 bits.
- That is, the key ID of public key
- This is a sufficient length that the probability of duplicate key IDs is very small.
- A key ID is also required for the PGP digital signature. Because a sender may use one of a number of private keys to encrypt the message digest, the recipient must know which public key is intended for use.
- Accordingly, the digital signature component of a message includes the 64-bit key ID of the required public key.
- When the message is received, the recipient verifies that the key ID is for a public key that it knows for that sender and then proceeds to verify the signature.

PGP Key ID concept

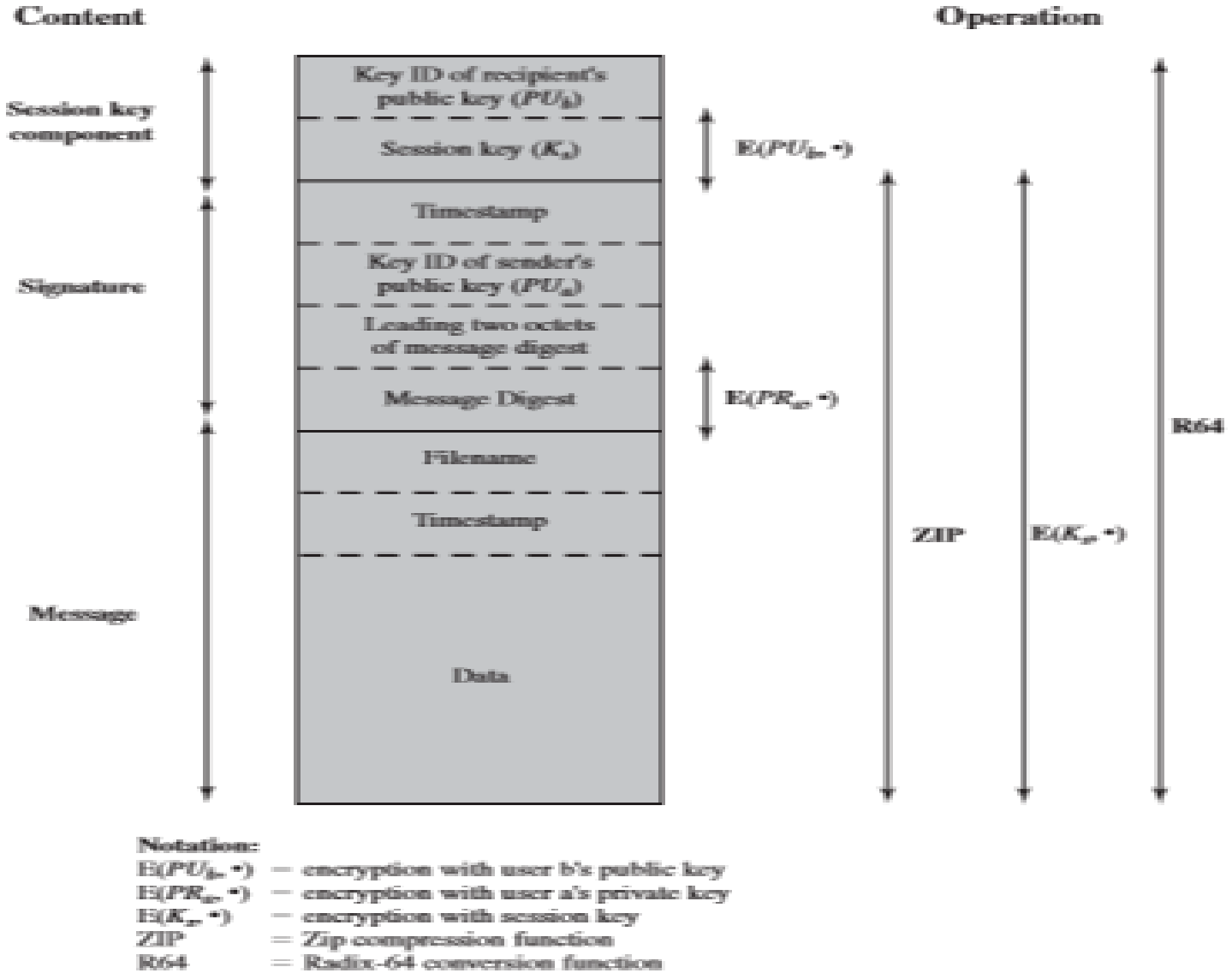


Figure 7.3 General Format PGP Message (from A to B)

- 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. The digest is calculated over the signature timestamp concatenated with the data portion of the message component. The inclusion of the signature timestamp in the digest insures against **replay types of attacks**.
- The exclusion of the filename and timestamp portions of the message component ensures that detached signatures are exactly the same as attached signatures prefixed to the message.

- **Leading two octets of message digest:** Enables 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 identifies the private key that was used to encrypt the message digest.
- The **session key component** includes the session key and the identifier of the recipient's public key that was used by the sender to encrypt the session key.

PGP Key Rings

- each PGP user has a pair of keyrings to store public and private keys
  - public-key ring contains all the public-keys of other PGP users known to this user

Public Key Ring

Timestamp	Key ID*	Public Key	Owner Trust	User ID*	Key Legitimacy	Signature(s)	Signature Trust(s)
• • •	• • •	• • •	• • •	• • •	• • •	• • •	• • •
T_i	$PU_i \bmod 2^{64}$	PU_i	trust_flag_i	User i	trust_flag_i		
• • •	• • •	• • •	• • •	• • •	• • •	• • •	• • •

PGP Key Rings

- private-key ring contains the public/private key pair(s) for this user,
- private keys are encrypted using a key derived from a hashed passphrase

Private Key Ring

Timestamp	Key ID*	Public Key	Encrypted Private Key	User ID*
• • •	• • •	• • •	• • •	• • •
T_i	$PU_i \bmod 2^{64}$	PU_i	$E(H(P_i), PR_i)$	User i
• • •	• • •	• • •	• • •	• • •

Private-key ring

- **Timestamp:** The date/time when this key pair was generated.
- **Key ID:** The least significant 64 bits of the public key for this entry.
- **Public key:** The public-key portion of the pair.
- **Private key:** The private-key portion of the pair;this field is encrypted.
- **User ID:** Typically,this will be the user's e-mail address (e.g.,stallings@acm.org). However, the user may choose to associate a different name with each pair (e.g., Stallings,WStallings,WilliamStallings, etc.) or to reuse the same User ID more than once.

- The private-key ring can be indexed by either User ID or Key ID
- Although it is intended that the private-key ring be stored only on the machine of the user that created and owns the key pairs and that it be accessible only to that user, it makes sense to make the value of the private key as secure as possible.
- Accordingly the private key itself is not stored in the key ring.
- Rather this key is encrypted using CAST-128 (or IDEA or 3DES).The procedure is as follows:

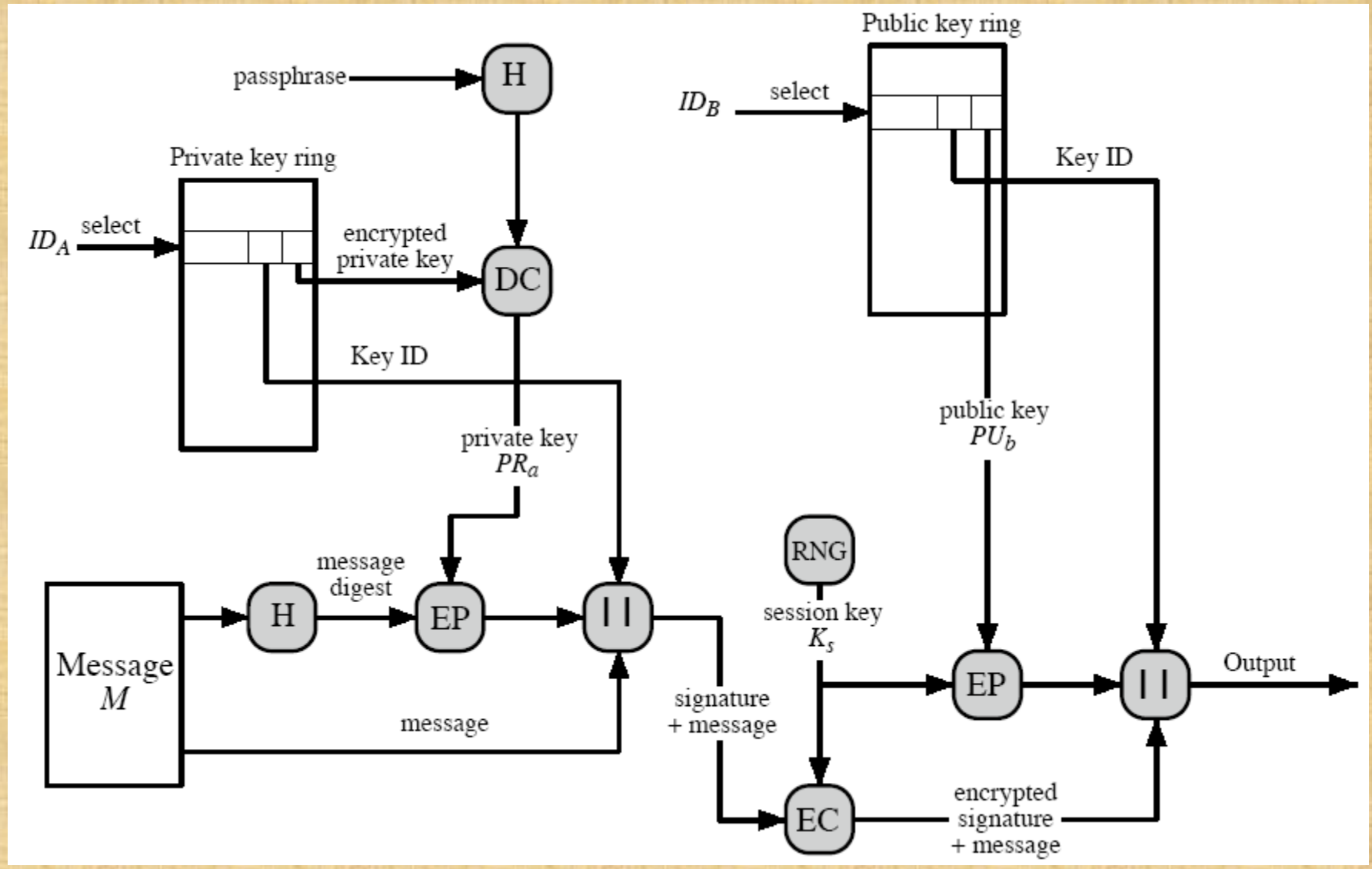
1. The user selects a passphrase to be used for encrypting private keys.
2. When the system generates a new public/private key pair using RSA it asks the user for the passphrase. Using SHA-1, a 160-bit hash code is generated from the passphrase and the passphrase is discarded.
3. The system encrypts the private key using CAST-128 with the 128 bits of the hash code as the key. The hash code is then discarded, and the encrypted private key is stored in the private-key ring.

- Subsequently, when a user accesses the private-key ring to retrieve a private key, he or she must supply the passphrase. PGP will retrieve the encrypted private key, generate the hash code of the passphrase, and decrypt the encrypted private key using CAST-128 with the hash code.

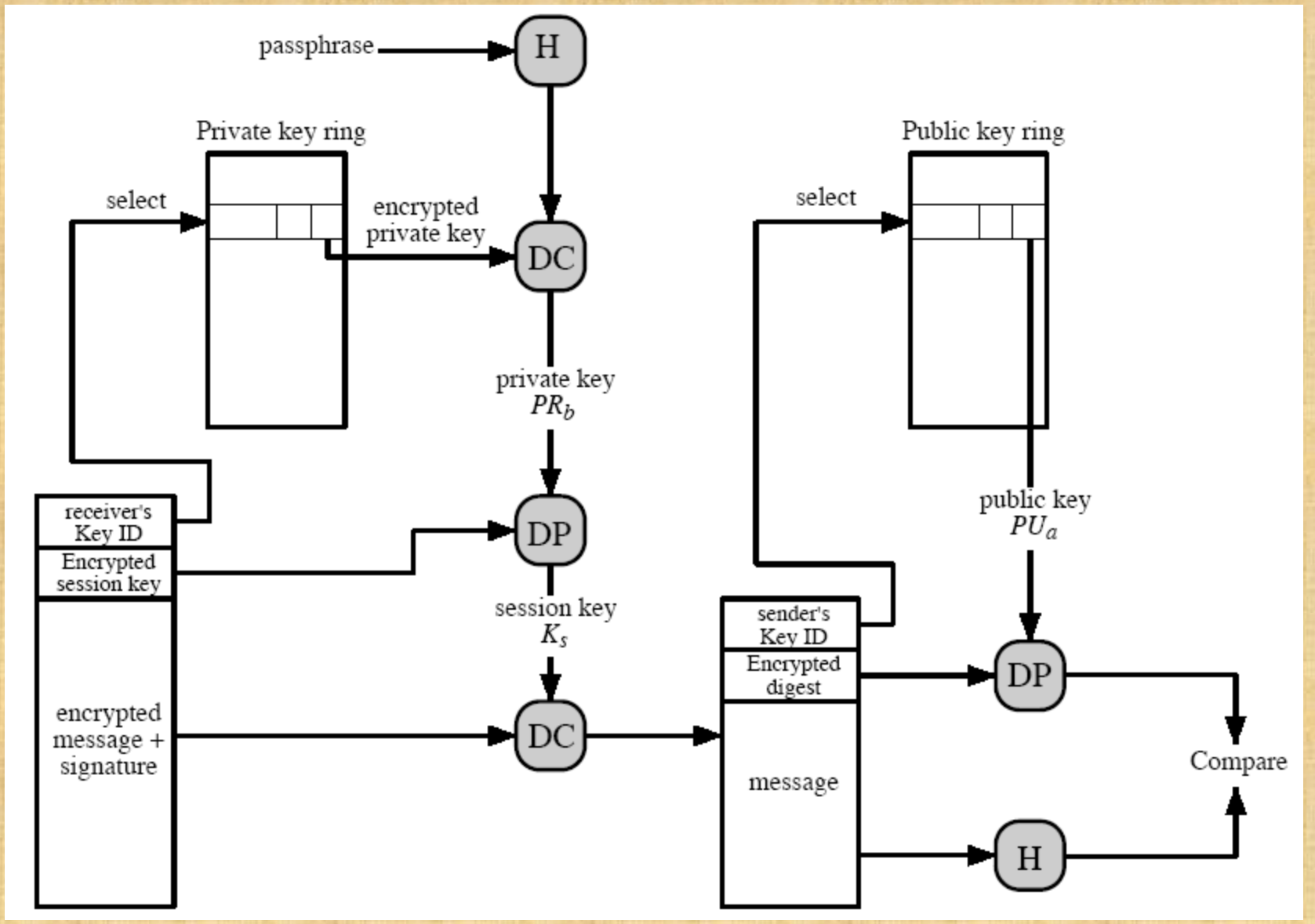
Public-key ring

- **Timestamp:** The date/time when this entry was generated.
- **Key ID:** The least significant 64 bits of the public key for this entry.
- **Public Key:** The public key for this entry.
- **User ID:** Identifies the owner of this key. Multiple user IDs may be associated with a single public key.

Key rings and message generation



Key rings and message reception



PGP Key Management - 1

- From PGP documentation:

  "This whole business of protecting public keys from tampering is the most difficult problem in practical public key applications"

- You have to make sure about the legitimacy of the public key of your party
  - exchange public-keys manually (using CDs, USB sticks, etc.)
  - verify fingerprint of a public key over the phone
  - trust another individual who signs public keys
    - public key signatures

PGP Key Management - 2

- Public keys could be signed by
  - Certification Authorities (CA)
    - trusted entities
    - the mechanism of S/MIME, not in PGP
  - in PGP each user is a CA
    - everybody can sign keys of users they know directly
    - other users' key signatures can also be used, if those users are trusted
- The only ultimately trusted entity is yourself
  - all other keys should either be directly signed by you or there should be a trusted path of key signatures
  - you reflect your own trust assessment in your public key ring (no system enforcement)
  - key ring includes trust indicators
  - "web of trust"

- **key legitimacy field** that indicates the extent to which PGP will trust that this is a valid public key for this user
- **Signature trust field** that indicates the degree to which this PGP user trusts the signer to certify public keys
- **Owner trust field** is included that indicates the degree to which this public key is trusted to sign other public-key certificates

States

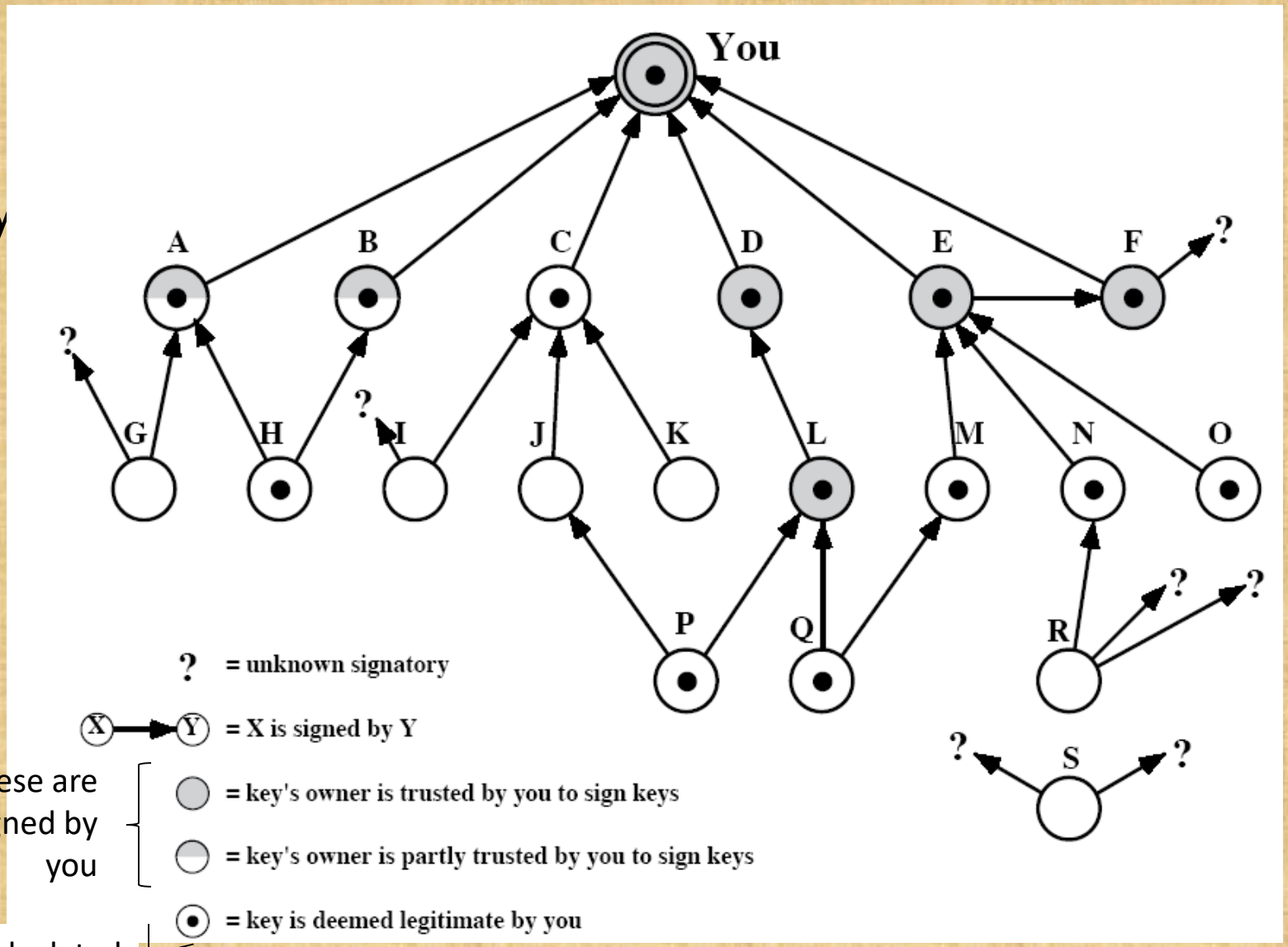
- ultimate trust
- Unknown
- untrusted
- marginally trusted
- completely trusted.

PGP Key Management - 3

- A trusted signature on a public key means that
  - the key really belongs to its owner
- But does not mean that key owner is trusted to sign other keys
  - key owner can sign other keys, but their trustworthiness is determined by the verifier (the owner of the pubkey ring)
- Making sure about the legitimacy of a key and trusting the key owner to find out other keys are two different concepts
- Keys and signatures on them are generally obtained from PGP public keyservers
  - there might be several signatures on a single key

PGP Key Management - 4

A public key ring owned by "you"



These are assigned by you

This is calculated

REVOKING PUBLIC KEYS

- A user may wish to revoke his or her current public key either because compromise is suspected or simply to avoid the use of the same key for an extended period.
- Note that a compromise would require that an opponent somehow had obtained a copy of your unencrypted private key or that the opponent had obtained both the private key from your private-key ring and your passphrase.
- The convention for revoking a public key is for the owner to issue a key revocation certificate, signed by the owner.This certificate has the same form as a normal signature certificate but includes an indicator that the purpose of this certificate is to revoke the use of this public key

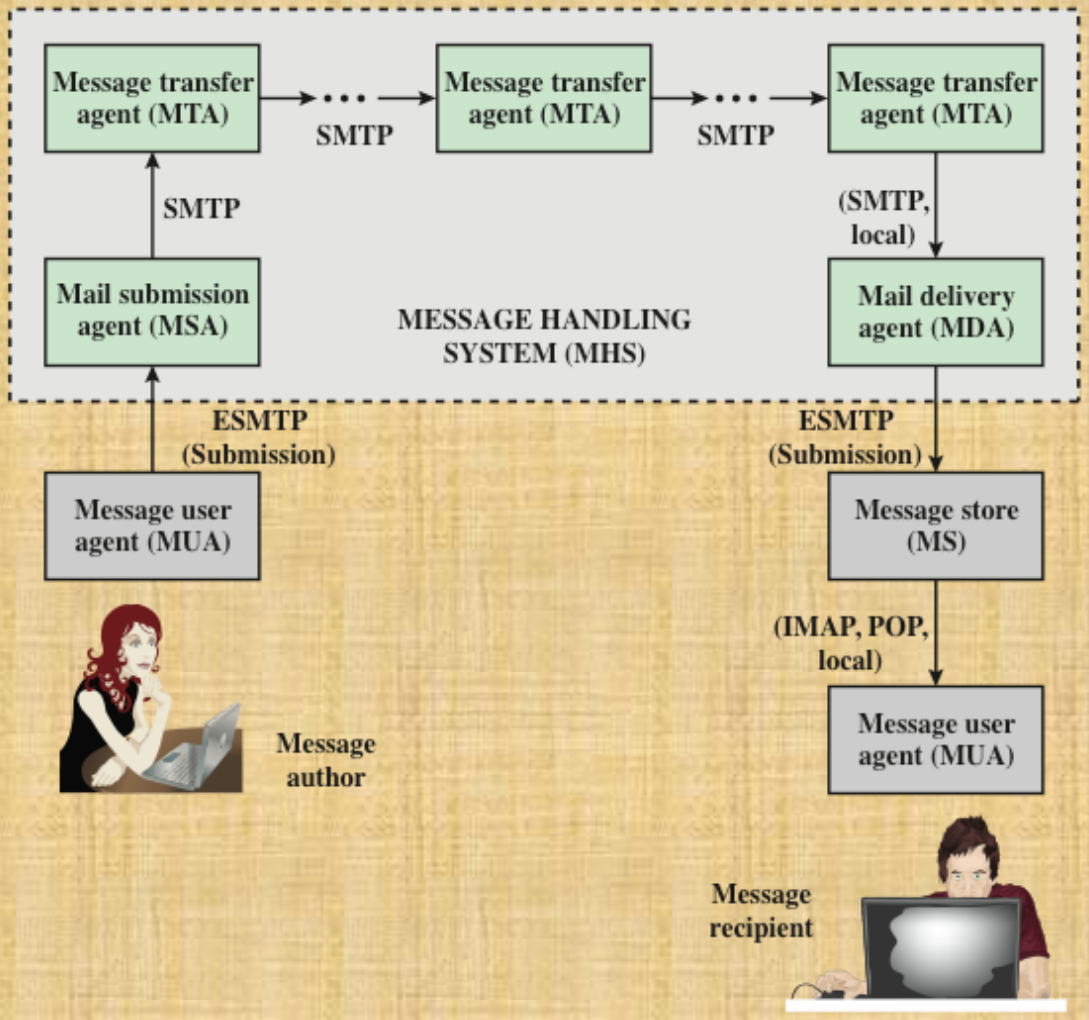
S/MIME

- Secure/Multipurpose Internet Mail Extensions
- A standard way for email encryption and signing
- IETF effort (RFCs 2632, 2633 – for version 3.0; RFCs 3850, 3851 for version 3.1; 5750, 5751 for version 3.2)
- Industry support
- Not a standalone software, a system that is to be supported by email clients
  - such as MS Outlook and Thunderbird
- S/MIME handles digital signatures
  - Also provides encryption

Quick E-mail History

- SMTP and RFC 822 (later RFC 5322)
  - SMTP is the email transfer protocol running over TCP
  - RFC 822/5322 defines the message format and headers
    - only ASCII messages (7-bit)
- MIME (Multipurpose Internet Mail Extensions)
  - content type
    - Almost any type of information can appear in an email message
  - transfer encoding
    - specifies how the message body is encoded into textual form (radix64 is common)
- S/MIME: Secure MIME
  - new content types, like signature, encrypted data

More on Internet Email Architecture



RFC 5322

- RFC 5322 defines a format for text messages that are sent using electronic mail.
- It has been the standard for Internet-based text mail messages and remains in common use.
- In the RFC 5322 context, messages are viewed as having an envelope and contents.
- The envelope contains whatever information is needed to accomplish transmission and delivery.
- The contents compose the object to be delivered to the recipient.
- The RFC 5322 standard applies only to the contents.
- However, the content standard includes a set of header fields that may be used by the mail system to create the envelope, and the standard is intended to facilitate the acquisition of such information by programs.

- 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.*
- *Put* differently, a message is ASCII text, and all lines up to the first blank line are assumed to be header lines used by the user agent part of the mail system.
- 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.*
- Another field that is commonly found in RFC 5322 headers is *Message-ID.*
- This field contains a unique identifier associated with this message.

```
Date: October 8, 2009 2:15:49 PM EDT
From: "William Stallings" <ws@shore.net>
Subject: The Syntax in RFC 5322
To: Smith@Other-host.com
Cc: Jones@Yet-Another-Host.com

Hello. This section begins the actual
message body, which is delimited from the
message heading by a blank line.
```

Limitations of the SMTP/5322 scheme

- SMTP cannot transmit executable files or other binary objects
- SMTP cannot transmit text data that includes national language characters
- SMTP servers may reject mail message over a certain size.
- SMTP gateways that translate between ASCII and the character code EBCDIC do not use a consistent set of mappings, resulting in translation problems.
- SMTP gateways to X.400 electronic mail networks cannot handle non textual data included in X.400 messages.

- Some SMTP implementations do not adhere completely to the SMTP standards defined in RFC 821.

Common problems include:

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

MIME is intended to resolve these problems in a manner that is compatible with existing RFC 5322 implementations.

MIME Elements

- Five new message header fields are defined, which may be included in an RFC 5322 header.
- These fields provide information about the body of the message.
- A number of content formats are defined, thus standardizing representations that support multimedia electronic mail.
- Transfer encodings are defined that enable the conversion of any content format into a form that is protected from alteration by the mail system.

More on MIME

- Multipurpose Internet Mail Extensions
  - Addresses the limitations of SMTP/5322 scheme
    - Most important one: binary data transfer
    - Attachments
- Adds some fields to the email messages: Important ones:
  - 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
  - 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

Message Header Fields

- **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

- For the **text type of body, no special software is required to get the full meaning** of the text aside from support of the indicated character set.
- *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 a 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.

MIME Content Types

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 ISDN mu-law encoding at a sample rate of 8 kHz.
Application	PostScript	Adobe Postscript format.
	octet-stream	General binary data consisting of 8-bit bytes.

(Table is on page 602 in textbook)

MIME CONTENT TYPES

```
From: Nathaniel Borenstein <nsb@bellcore.com>
To: Ned Freed <ned@innosoft.com>
Subject: Sample message
MIME-Version: 1.0
Content-type: multipart/mixed; boundary="simple
boundary"

This is the preamble. It is to be ignored, though it
is a handy place for mail composers to include an
explanatory note to non-MIME conformant readers.
—simple boundary

This is implicitly typed plain ASCII text. It does NOT
end with a linebreak.
—simple boundary
Content-type: text/plain; charset=us-ascii

This is explicitly typed plain ASCII text. It DOES end
with a linebreak.

—simple boundary—
This is the epilogue. It is also to be ignored.
```

MIME CONTENT TYPES

- There are four subtypes of the **multipart type**, all of which have the same overall syntax.
- The **Multipart/Mixed subtype is used when there are multiple independent** body parts that need to be bundled in a particular order.
- For the **multipart/parallel subtype, the order of the parts is not significant. If the recipient's** system is appropriate, the multiple parts can be presented in parallel.
  - For example, a picture or text part could be accompanied by a voice commentary that is played
  - while the picture or text is displayed.
- For the **multipart/alternative subtype, the various parts are different representations** of the same information.
  - The following is an example:

MIME CONTENT TYPES

```
From: Nathaniel Borenstein <nsb@bellcore.com>
To: Ned Freed <ned@innosoft.com>
Subject: Formatted text mail
MIME-Version: 1.0
Content-Type: multipart/alternative;
boundary=boundary42

    —boundary42

Content-Type: text/plain; charset=us-ascii

    ...plain text version of message goes here....

   —boundary42
   Content-Type: text/enriched

       .... RFC 1896 text/enriched version of same message
   goes here ...

   —boundary42—
```

MIME CONTENT TYPES

- The **multipart/digest subtype is used when each of the body parts is interpreted** as an RFC 5322 message with headers.
- This subtype enables the construction of a message whose parts are individual messages. For example, the moderator of a group might collect e-mail messages from participants, bundle these messages, and send them out in one encapsulating MIME message.

MIME CONTENT TYPES

- 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 5322 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.*

MIME CONTENT TYPES

- 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. As with the other message types, the message/external-body subtype has an outer header and an encapsulated message with its own header.
- The only necessary field in the outer header is the Content-Type field, which identifies this as a message/external-body subtype. The inner header is the message header for the encapsulated message.
- The Content-Type field in the outer header must include an access-type parameter, which indicates the method of access, such as FTP (file transfer protocol).
- 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 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

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.

(Table is on page 605 in textbook)

```
MIME-Version: 1.0
From: Nathaniel Borenstein <nsb@bellcore.com>
To: Ned Freed <ned@innosoft.com>
Subject: A multipart example
Content-Type: multipart/mixed;
      boundary=unique-boundary-1

This is the preamble area of a multipart message. Mail readers that understand multipart format should ignore
this preamble. If you are reading this text, you might want to consider changing to a mail reader that understands
how to properly display multipart messages.

--unique-boundary-1

      ...Some text appears here...
[Note that the preceding blank line means no header fields were given and this is text, with charset US ASCII.
It could have been done with explicit typing as in the next part.]

--unique-boundary-1
Content-type: text/plain; charset=US-ASCII

This could have been part of the previous part, but illustrates explicit versus implicit typing of body parts.

--unique-boundary-1
Content-Type: multipart/parallel;   boundary=unique-boundary-2

--unique-boundary-2
Content-Type: audio/basic
Content-Transfer-Encoding: base64

      ... base64-encoded 8000 Hz single-channel mu-law-format audio data goes here....

--unique-boundary-2
Content-Type: image/jpeg
Content-Transfer-Encoding: base64

      ... base64-encoded image data goes here....

--unique-boundary-2--

--unique-boundary-1
Content-type: text/enriched

This is <bold><italic>richtext.</italic></bold> <smaller>as defined in RFC 1896</smaller>

Isn't it <bigger><bigger>cool?</bigger></bigger>

--unique-boundary-1
Content-Type: message/rfc822

From: (mailbox in US-ASCII)
To: (address in US-ASCII)
Subject: (subject in US-ASCII)
Content-Type: Text/plain; charset=ISO-8859-1
Content-Transfer-Encoding: Quoted-printable

      ... Additional text in ISO-8859-1 goes here ...

--unique-boundary-1--
```

Figure 7.8 Example MIME Message Structure

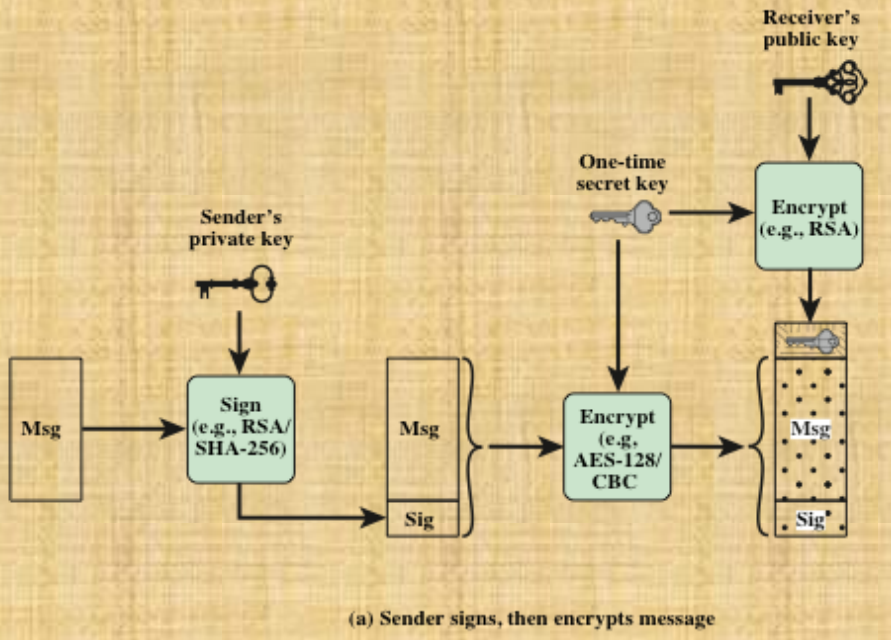
S/MIME Functions

- enveloped data
  - encrypted content and associated keys
- signed data
  - encoded message + encoded signed message digest
- clear-signed data
  - cleartext message + encoded signed message digest
- signed and enveloped data
  - Nested signed and encrypted entities

S/MIME Cryptographic Algorithms

- hash functions: switched to SHA256
- digital signatures: Mostly RSA is used
- session key encryption: Mostly RSA is used
- message encryption: Triple-DES, AES and others, but AES-128 is preferred
- Base64 (radix64) encoding is used for email compatibility (ASCII conversion)
- sender should know the capabilities of the receiving entity (public announcement or previously received messages from receiver)
  - otherwise sender takes a risk of sending unintelligible e-mail.

S/MIME Security Functionality: Simplified View



Scope of S/MIME Security

- S/MIME secures a MIME entity
  - a MIME entity is entire message except the headers
  - so the header is not secured
- First MIME message is prepared
- This message and other security related data (algorithm identifiers, certificates, etc.) are processed by S/MIME
- and packed as one of the S/MIME content type

S/MIME Content Types

Type	Subtype	smime Parameter	Description
Multipart	Signed		A clear-signed message in two parts: one is the message and the other is the signature.
Application	pkcs7-mime	signedData	A signed S/MIME entity.
	pkcs7-mime	envelopedData	An encrypted S/MIME entity.
	pkcs7-mime	degenerate signedData	An entity containing only public-key certificates.
	pkcs7-mime	CompressedData	A compressed S/MIME entity
	pkcs7-signature		The content type of the signature subpart of a multipart/signed message.

EnvelopedData

- For message encryption
- Similar to PGP
  - create a random session key, encrypt the message with that key and a conventional crypto, encrypt the session key with recipient's public key
- Unlike PGP, recipient's public key comes from an X.509 certificate
  - trust management is different

SignedData

- For signed message
  - both message and signature are encoded so that the recipient only sees some ASCII characters if he does not use an email client with S/MIME support
- Similar to PGP
  - first message is hashed, then the hash is encrypted using sender's private key
- Message, signature, identifiers of algorithms and the sender's certificate are packed together
  - again difference between S/MIME and PGP in trust management

Clear Signing

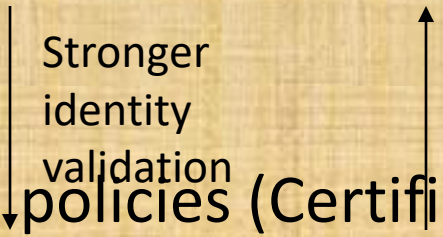
- Another mechanism for signature
  - but the message is not encoded, so an email client with no S/MIME support could also view the message
    - of course the signature will not be verified and will be seen as a meaningless attachment
- multipart/signed content type
  - 2 parts
    - Clear text message
    - Signature
  - Let's see an example

S/MIME Certificate Processing

- S/MIME uses X.509 v3 certificates
  - Certification Authorities (CAs) issue certificates
  - unlike PGP, a user cannot be a CA
- each client has a list of trusted CA certificates
  - actually that list comes with e-mail client software or OS
- and own public/private key pairs and certs
- It is very hard for an average user to maintain the list of trusted CAs
  - Generally OS and/or email client software default trusted CA certificate lists are directly used.
  - So trust management is not user-centric in practice

S/MIME Certificate Processing and CAs

- One should obtain a certificate from a CA in order to send signed messages
- Certificates classes (common practice by most CAs)
  - Class 1
  - Class 2
  - Class 3



- CA certification policies (Certificate Practice Statement)
  - ID-control practices
    - Class 1: only email address check
    - Class 2: class1 + against third party database / fax documents
    - Class 3: class1 + apply in person and submit picture IDs and/or paper documents

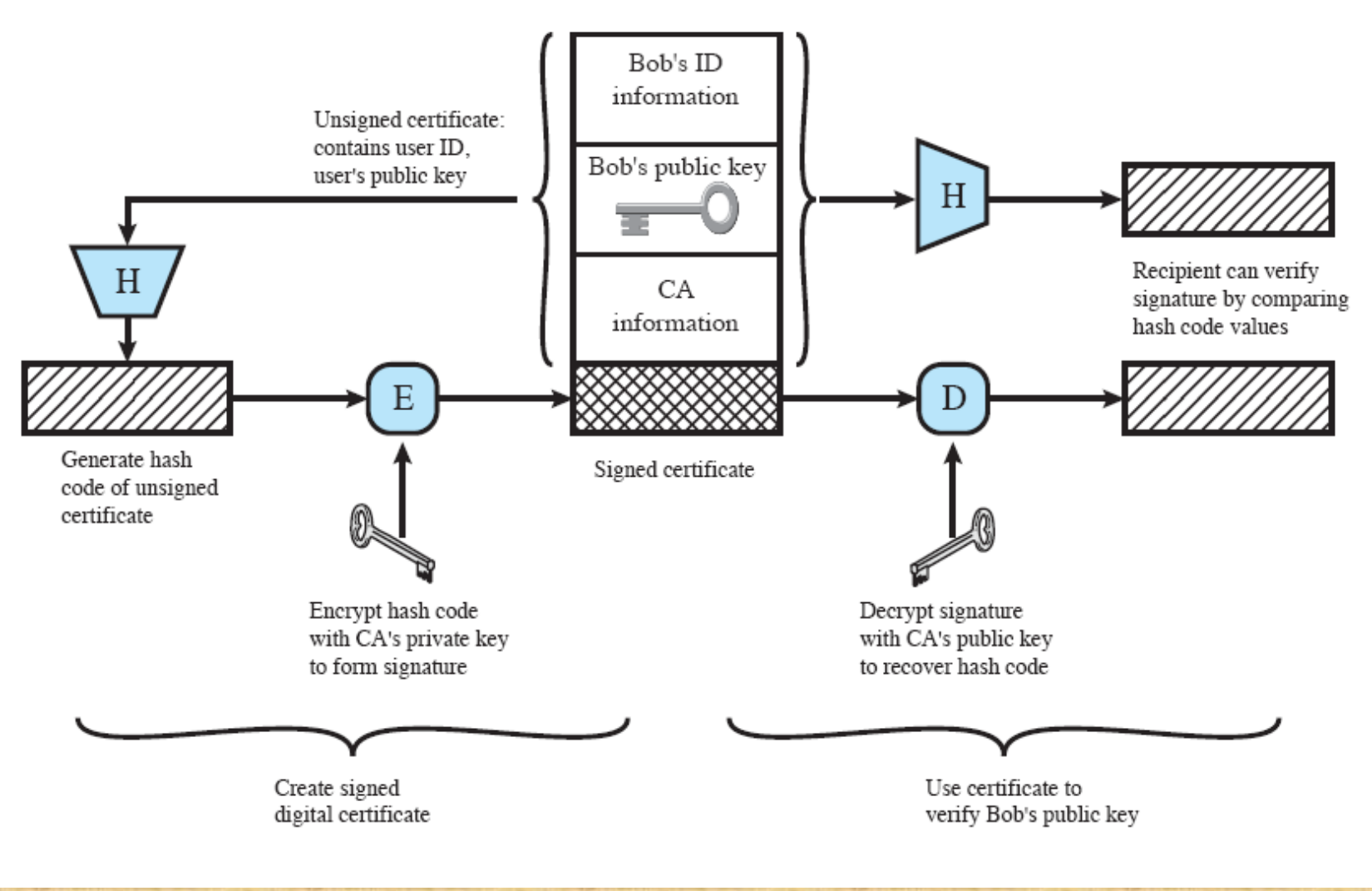
X.509 Certificates and PKIs

- SSL and S/MIME uses X.509 certificates
  - now we will see the details of them
  - later we will continue with PKIs (Public Key Infrastructures)

Certificates

- Yet another public-key distribution method
  - first (conceptually) offered by Kohnfelder (1978)
- Binding between the public-key and its owner
- Issued (digitally signed) by the Certificate Authority (CA)

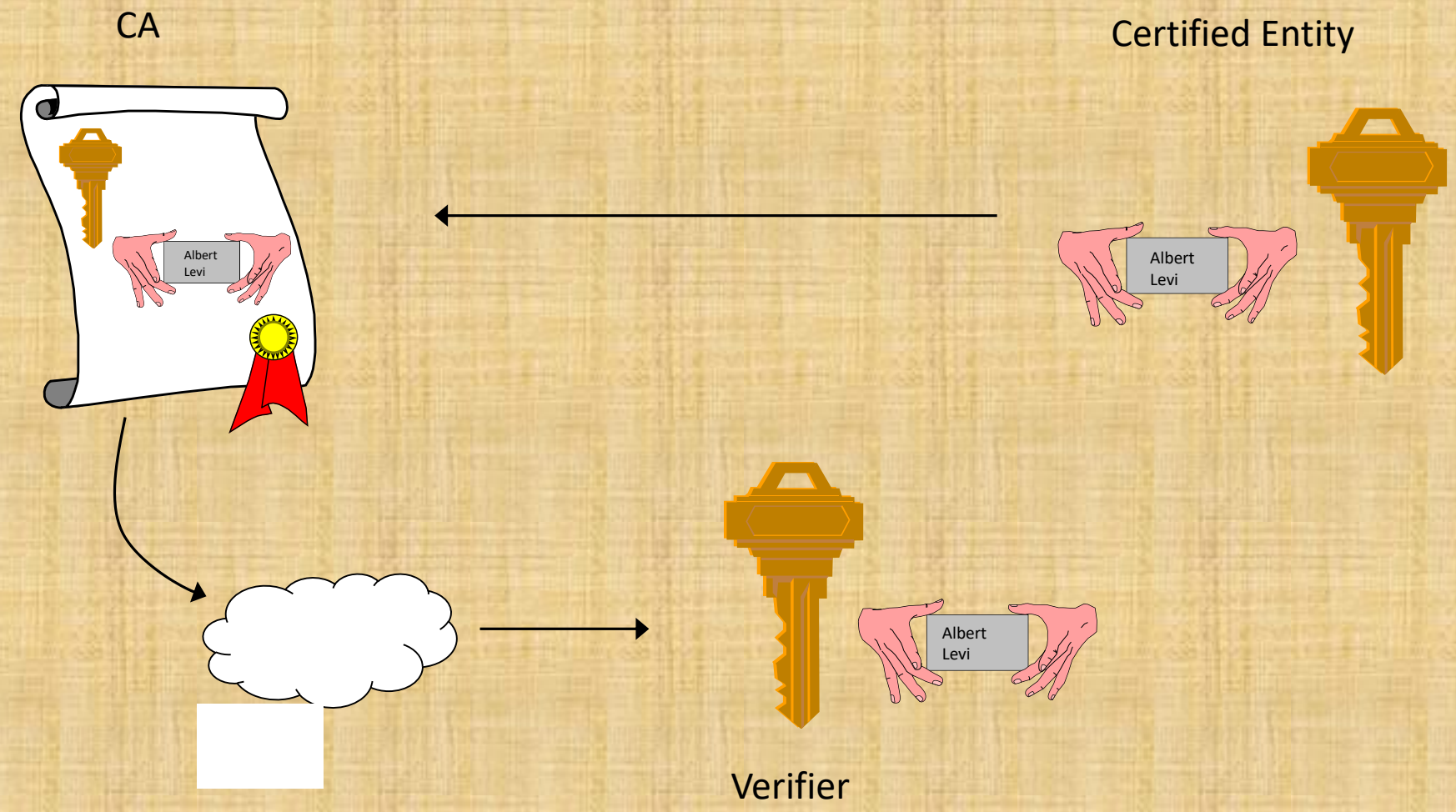
Certificates



Certificates

- Certificates are verified by the verifiers to find out correct public key of the target entity
- Certificate verification is the verification of the signature on certificate
- In order to verify a certificate, the verifier
  - must know the public key of the CA
  - must trust the CA

Certificates



Issues Related Certificates

- CA certification policies (Certificate Practice Statement)
  - how reliable is the CA?
  - certification policies describe the methodology of certificate issuance
  - ID-control practices
    - loose control: only email address
    - tight control: apply in person and submit picture IDs and/or hard documentation

Issues Related Certificates

- TRUST
  - verifiers must trust CAs
  - CAs need not trust the certified entities
  - certified entities need not trust its CA
- What is “trust” in certification systems?
  - Answer to the question: “How correct is the certificate information?”
  - related to certification policies

Issues Related Certificates

- Certificate types
  - ID certificates
    - discussed here
  - authorization certificates
    - no identity
    - binding between public key and authorization info
- Certificate storage and distribution
  - along with a signed message
  - distributed/centralized databases

Issues Related Certificates

- Certificate Revocation
  - certificates have lifetimes, but they may be revoked before the expiration time
  - Reasons:
    - certificate holder key compromise/lost
    - CA key compromise
    - end of contract (e.g. certificates for employees)
  - Certificate Revocation Lists (CRLs) hold the list of certificates that are not expired but revoked
    - each CA periodically issues such a list with digital signature on it

Real World Analogies

- Is a certificate an “electronic identity”?
- Concerns
  - a certificate is a binding between an identity and a key, not a binding between an identity and a real person
  - anyone can submit someone else’s certificate
  - one must submit its certificate to identify itself, but submission is not sufficient, the key must be used in a protocol

Real World Analogies

- Result: Certificates are not picture IDs
- So, what is the real world analogy for certificates?
  - Endorsed document/card that serves as a binding between the identity and signature

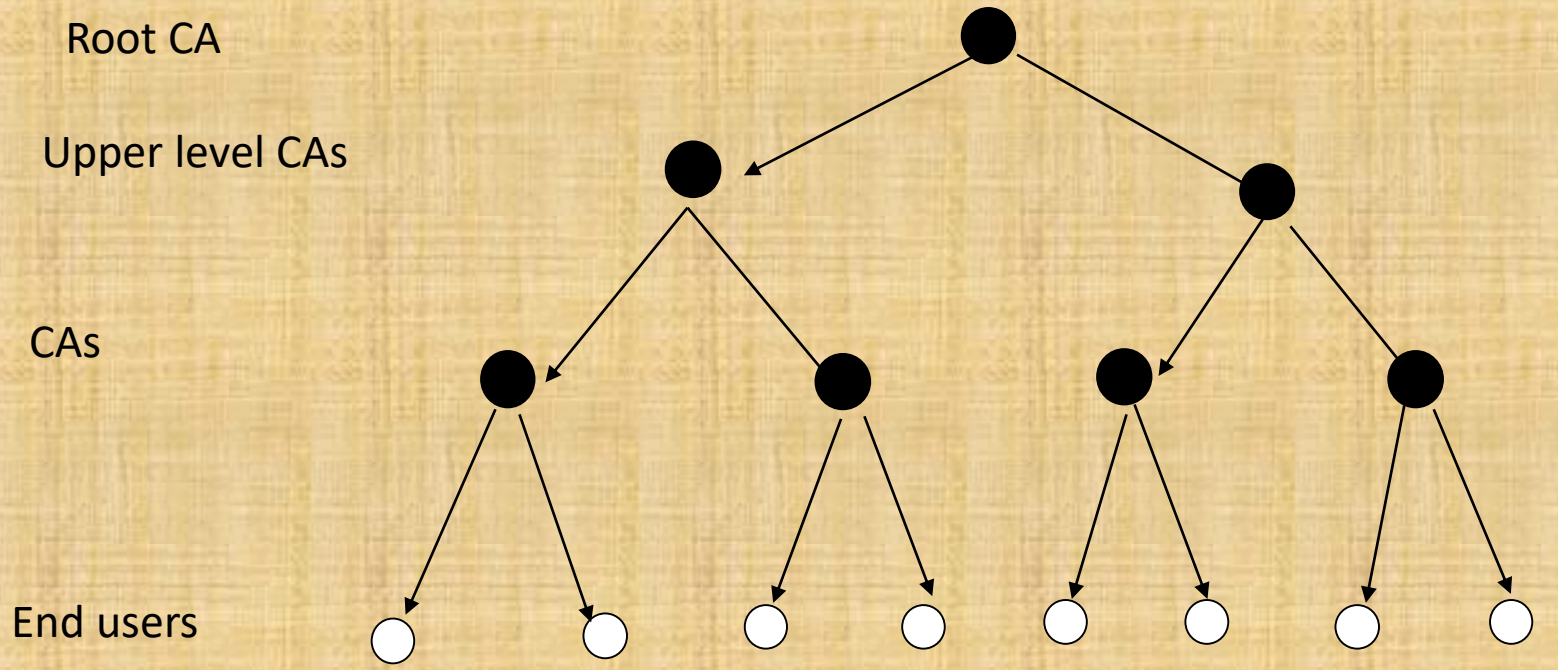
Public Key Infrastructure (PKI)

- PKI is a complete system and well-defined mechanisms for certificates
  - certificate issuance
  - certificate revocation
  - certificate storage
  - certificate distribution

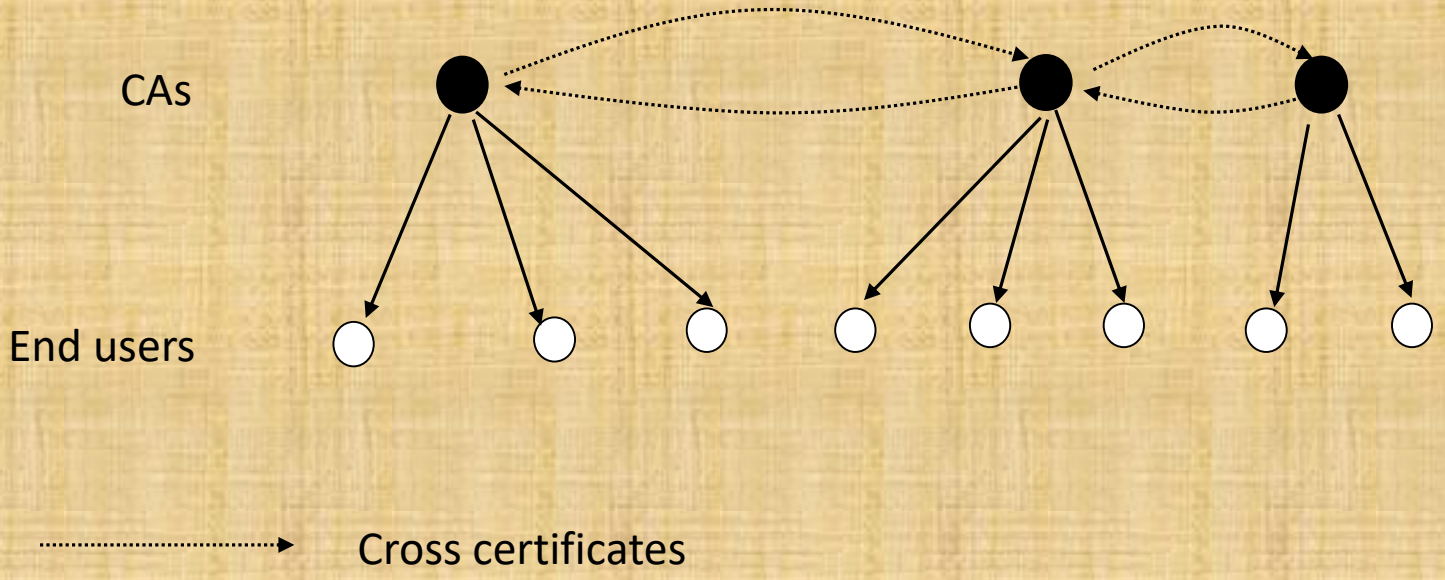
PKI

- Business Practice: Issue certificates and make money
  - several CAs
- Several CAs are also necessary due to political, geographical and trust reasons
- 3 interconnection models
  - hierarchical
  - cross certificates
  - hybrid

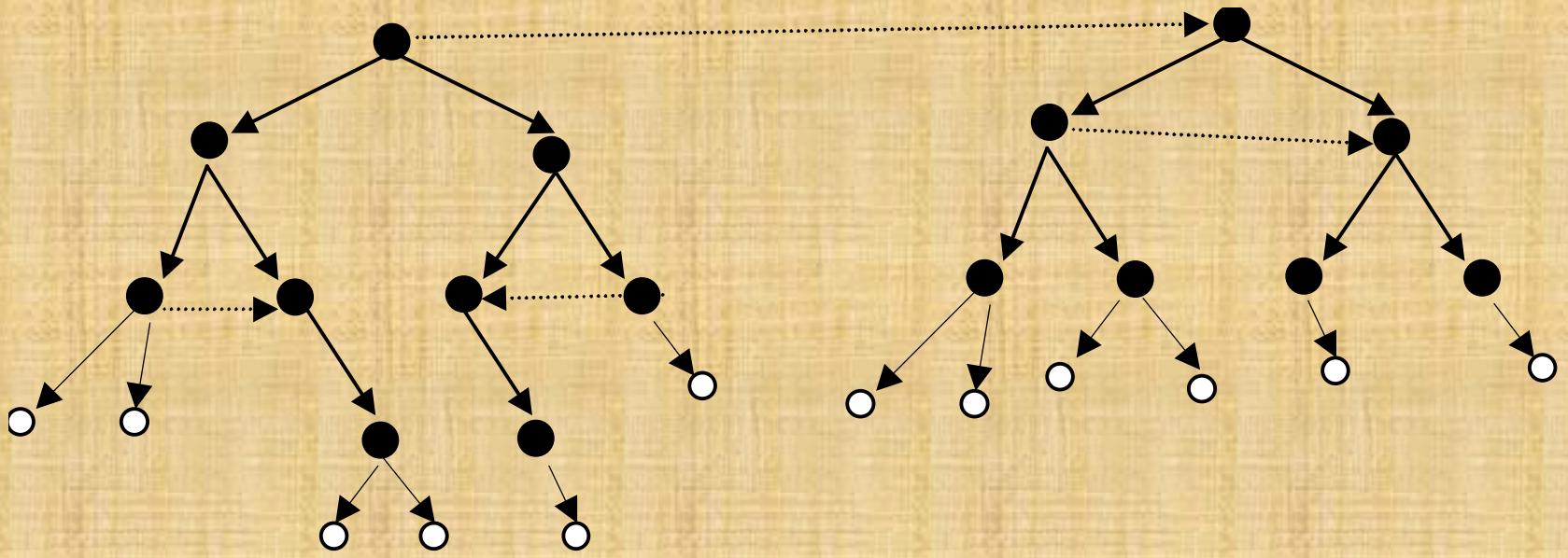
Hierarchical PKI Example



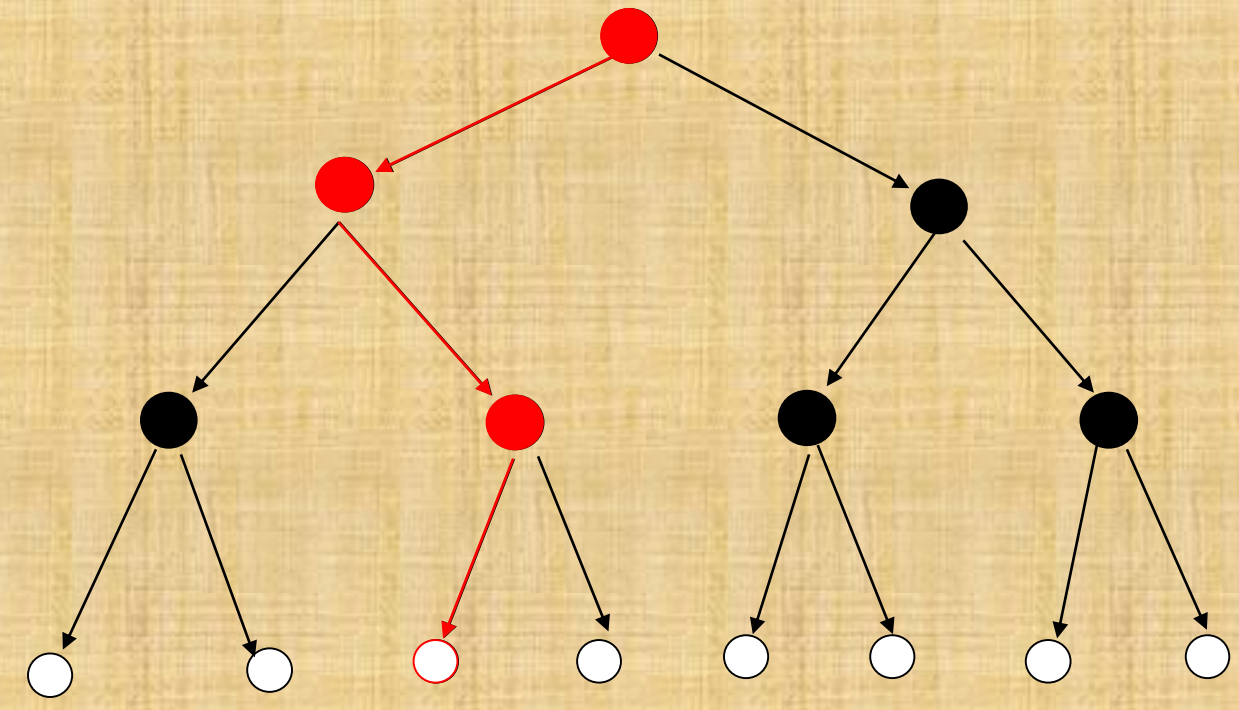
Cross Certificate Based PKI Example



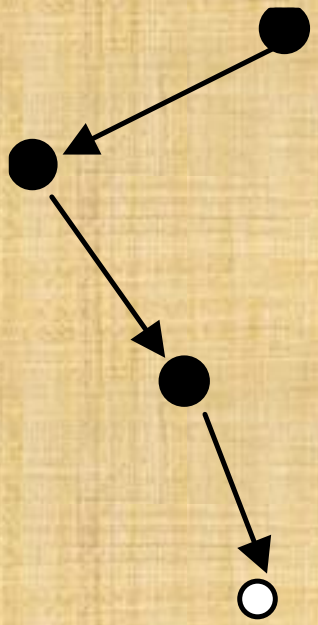
Hybrid PKI example



Certificate Paths

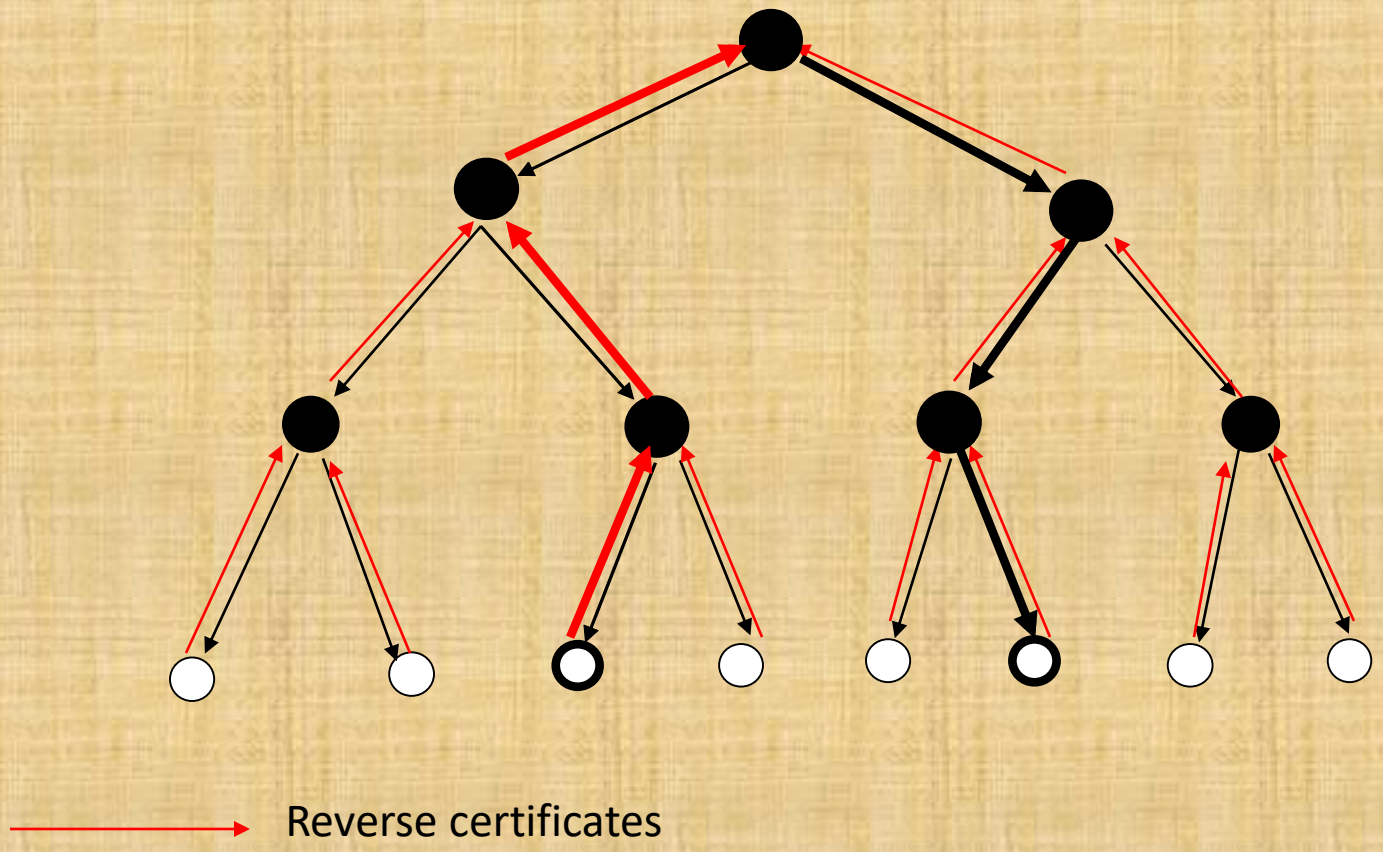


Certificate Paths



- Verifier must know public key of the first CA
- Other public keys are found out one by one
- All CAs on the path must be trusted by the verifier

Certificate Paths with Reverse Certificates

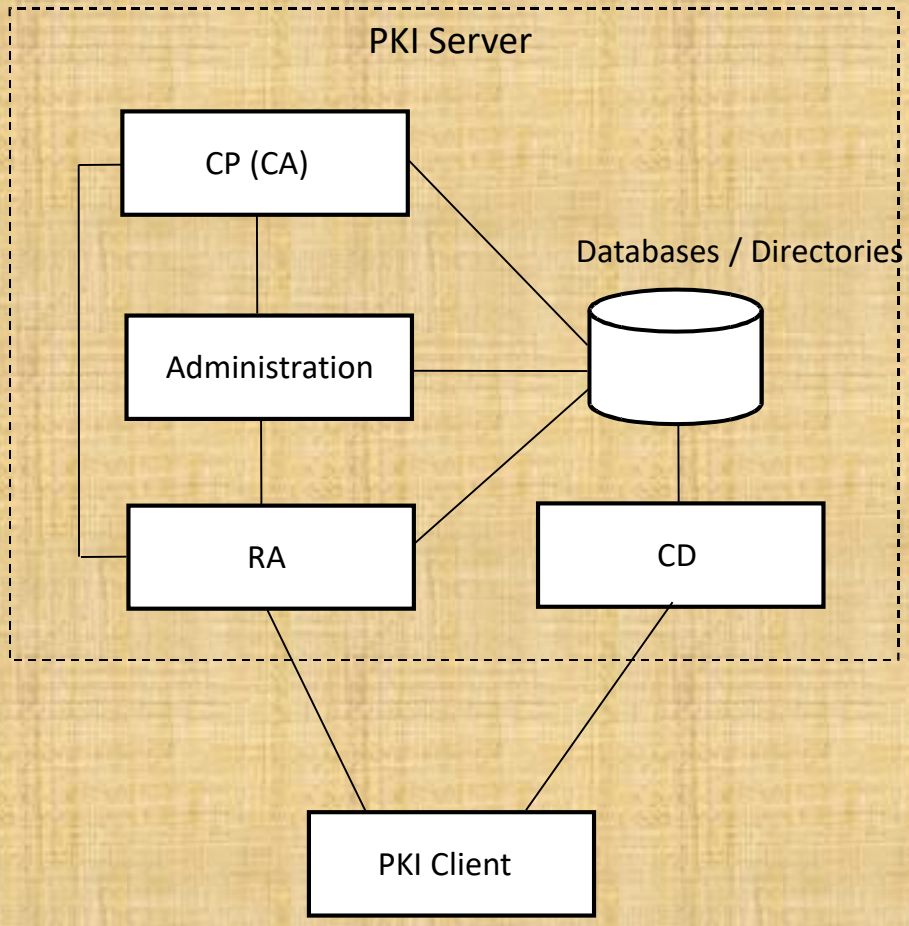


Organization-wide PKI

- Local PKI for organizations
  - may have global connections, but the registration facilities remain local
  - generally to solve local problems
    - local secure access to resources

Organization-wide PKI

Certificate Processor/Authority



Registration Authority

Certificate Distribution

Architecture of a typical organization-wide PKI

Hosted vs. Standalone PKI

- Hosted (outsourced) PKI
  - PKI vendor acts as CA
  - PKI owner is the RA
- Standalone PKI
  - PKI owner is both RA and CA

Hosted vs. Standalone PKI

Advantages of hosted PKI over standalone PKI

Standalone PKI	Hosted PKI
Organization has to have a secure server for certificate issuance and processing.	Organization does not need to run a secure server for certificate processing.
Organization must issue cross certificates or has to have some other arrangements for universal connection of its PKI. Otherwise, the PKI remains local.	PKI provider (host) already has such arrangements. Organization does not have to worry about worldwide visibility of its PKI.
More administrative work for organization.	Less administrative work for organization.

Disadvantages of hosted PKI over standalone PKI

Standalone PKI	Hosted PKI
No continuous dependency on the PKI vendor. Organization does not have to pay periodic fees.	Continuous dependency on the PKI vendor (host). The organization must pay regular fees to the host based on the certificate volume.
Security of the PKI is in the organization's hands.	Although the organization is responsible for the security of its PKI, they are dependent on the host's security.
Organization does not have to trust the PKI vendor as different than its other software vendors.	Ultimate trust to host is indispensable.
The only user of the private key is the organization itself.	Private key is being used by the host for certificate issuance.

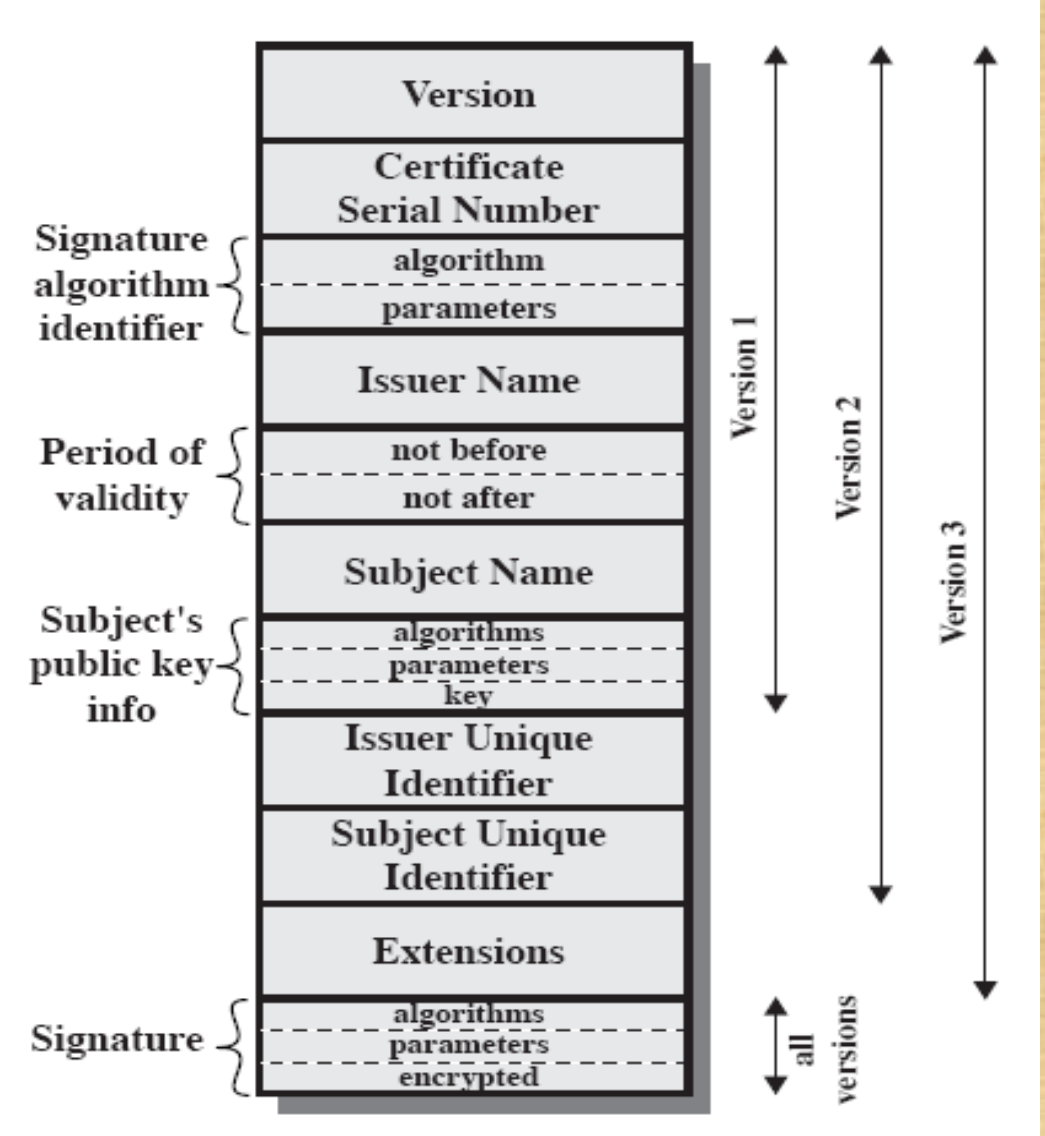
X.509

- ITU-T standard (recommendation)
  - ISO 9495-2 is the equivalent ISO standard
- part of X.500 family for “directory services”
  - distributed set of servers that store user information
    - an utopia that has never been carried out
  - X.509 defines the authentication services and the pubic-key certificate structure (certificates are to be stored in the directory)
  - so that the directory would contain public keys of the users

X.509

- Defines identity certificates
  - attribute (authorization) certificates are added in 4th edition (2000)
- Defines certificate structure, not PKI
- Supports both hierarchical model and cross certificates
- End users cannot be CAs

X.509 Certificate Format



X.509v3 Extensions

- Not enough flexibility in X.509 v1 and v2
  - mostly due to “directory” specific fields
  - real-world security needs are different
    - email/URL names should be included in a certificate
    - key identification was missing (so should be included)
    - policy details should indicate under which conditions a certificate can be used (was not the case in v1 and v2)
    - avoidance of blind trust was not possible in v1 and v2
- Rather than explicitly naming new fields a general extension method is defined
  - An extension consists of an extension identifier, value and criticality indicator

X.509v3 Extensions

- Key and policy information
  - subject & issuer key identifiers
  - indicators of certificate policies supported by the cert
  - key usage (list of purposes like signature, encryption, etc)
- Alternative names, in alternative formats for certificate subject and issuer
- Certificate path constraints
  - For CA certs and to restrict certificate issuance based on
    - path length (restricting number of subordinate CAs)
    - policy identifiers
    - names
- Verifier could exercise its own restrictions during verification as well
  - No blind trust to CAs