E-mail Security:
PGP and S/MIME

Outline

- PGP
  - services
  - message format
  - key management
  - trust management
- S/MIME
  - services
  - message formats
  - key management

What is PGP?

- PGP - Pretty Good Privacy
- general purpose application to protect (encrypt and/or sign) files
- can be used to protect e-mail messages
- can be used by corporations as well as individuals
- based on strong cryptographic algorithms (IDEA, RSA, SHA-1)
- available free of charge at http://www.pgpi.org
- first version developed by Phil Zimmermann
- PGP is now on an Internet standards track (RFC 3156)

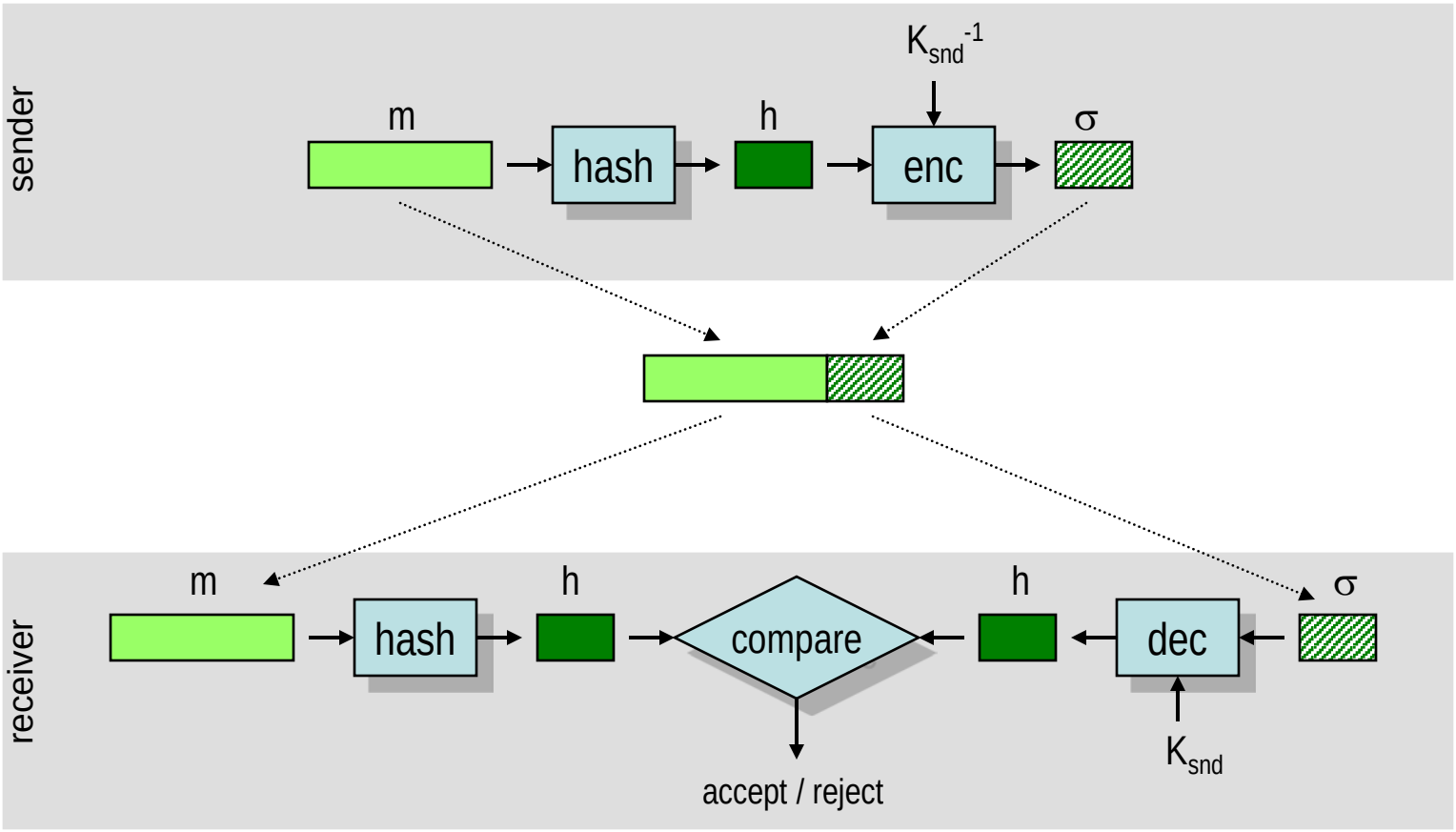
PGP services

- messages
  - authentication
  - confidentiality
  - compression
  - e-mail compatibility
  - segmentation and reassembly

- key management
  - generation, distribution, and revocation of public/private keys
  - generation and transport of session keys and IVs

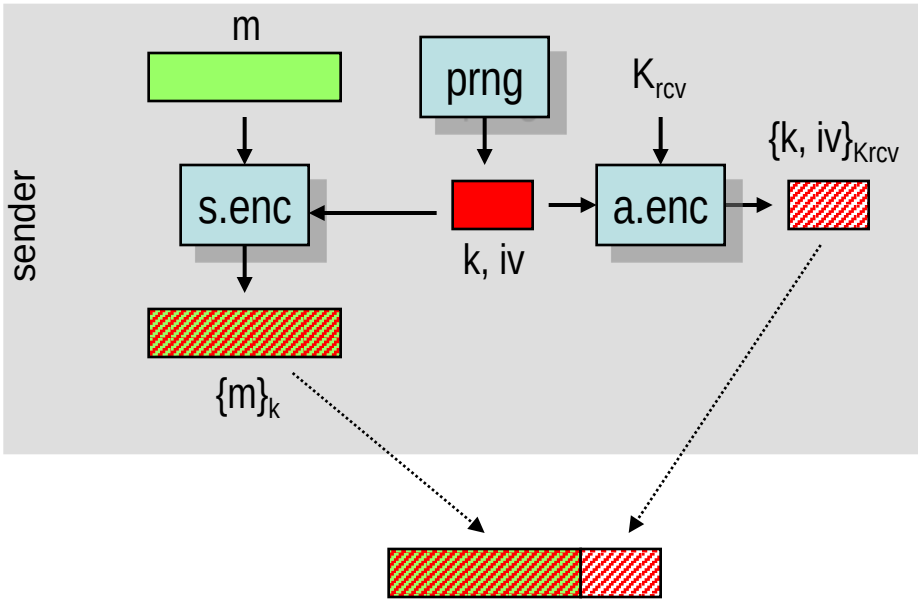
Message authentication

- based on digital signatures
- supported algorithms: RSA/SHA and DSS/SHA



Message confidentiality

- symmetric key encryption in CFB mode with a random session key and IV
- session key and IV is encrypted with the public key of the receiver
- supported algorithms:
  - symmetric: CAST, IDEA, 3DES
  - asymmetric: RSA, ElGamal

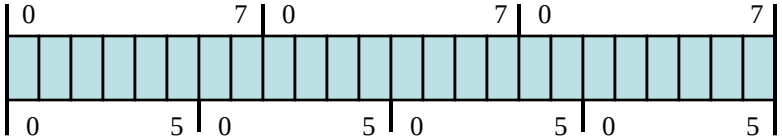


Compression

- applied after the signature
  - enough to store clear message and signature for later verification
  - it would be possible to dynamically compress messages before signature verification, but ...
  - then all PGP implementations should use the same compression algorithm
  - however, different PGP versions use slightly different compression algorithms
- applied before encryption
  - compression reduces redundancy → makes cryptanalysis harder
- supported algorithm: ZIP

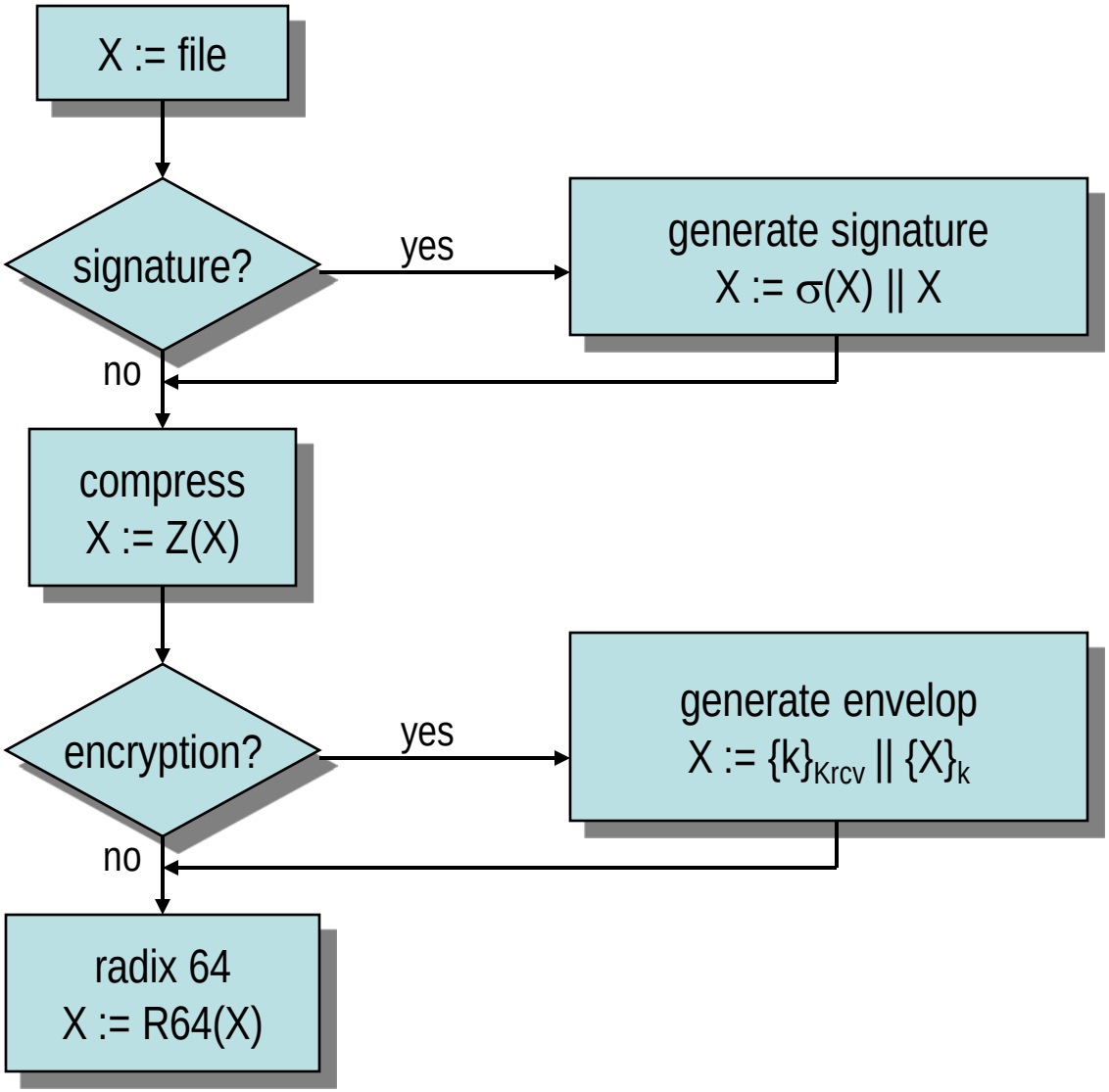
E-mail compatibility

- encrypted messages and signatures may contain arbitrary octets
- most e-mail systems support only ASCII characters
- PGP converts an arbitrary binary stream into a stream of printable ASCII characters
- radix 64 conversion: 3 8-bit blocks → 4 6-bit blocks

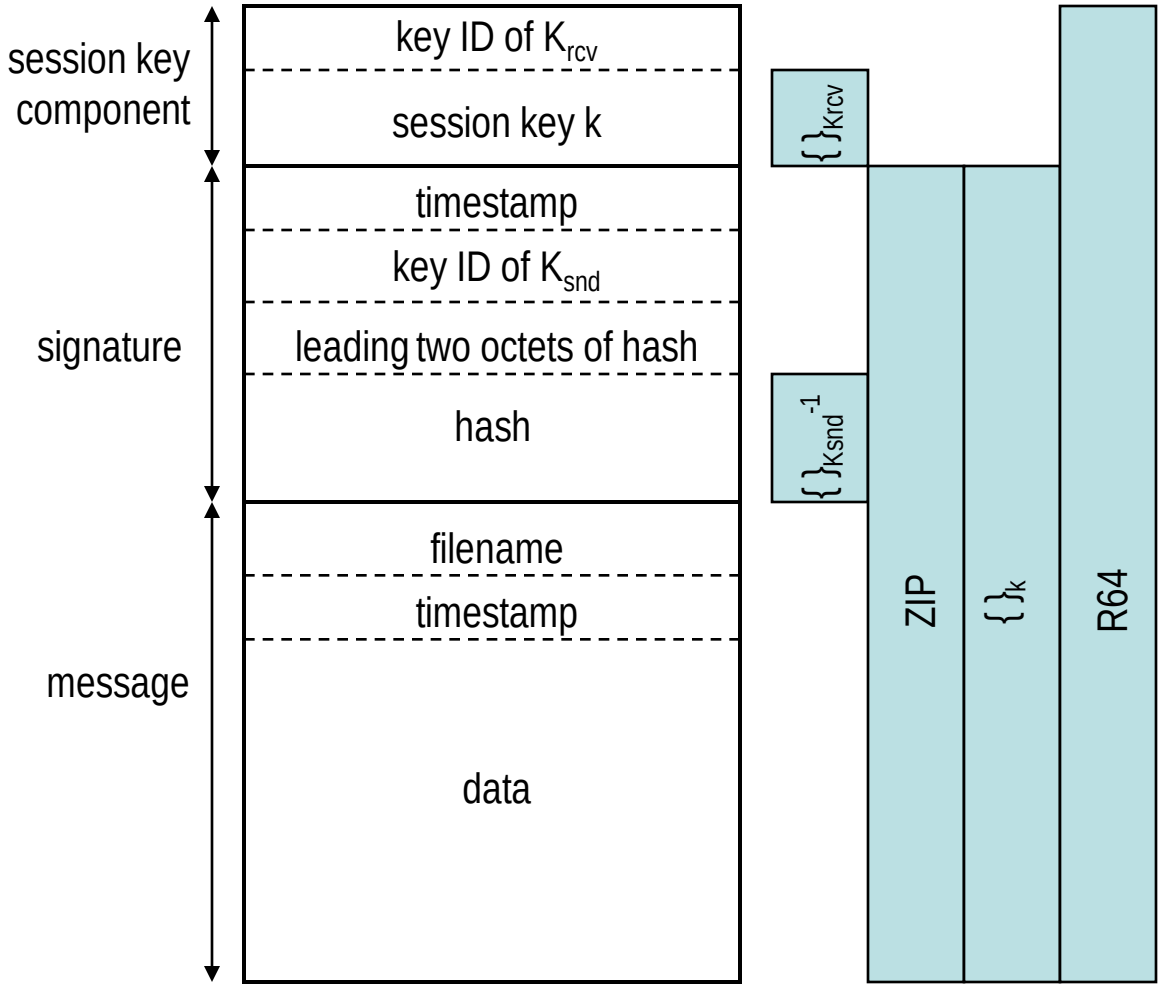


6-bit value	character encoding	6-bit value	character encoding
0	A	52	0
…	…	…	…
25	Z	61	9
26	a	62	+
…	…	63	/
51	z	(pad)	=

Combining services



PGP message format



Key IDs

- a user may have several public key – private key pairs
  - which private key to use to decrypt the session key?
  - which public key to use to verify a signature?
- transmitting the whole public key would be wasteful
- associating a random ID to a public key would result in management burden
- PGP key ID: least significant 64 bits of the public key
  - unique within a user with very high probability

Random number generation

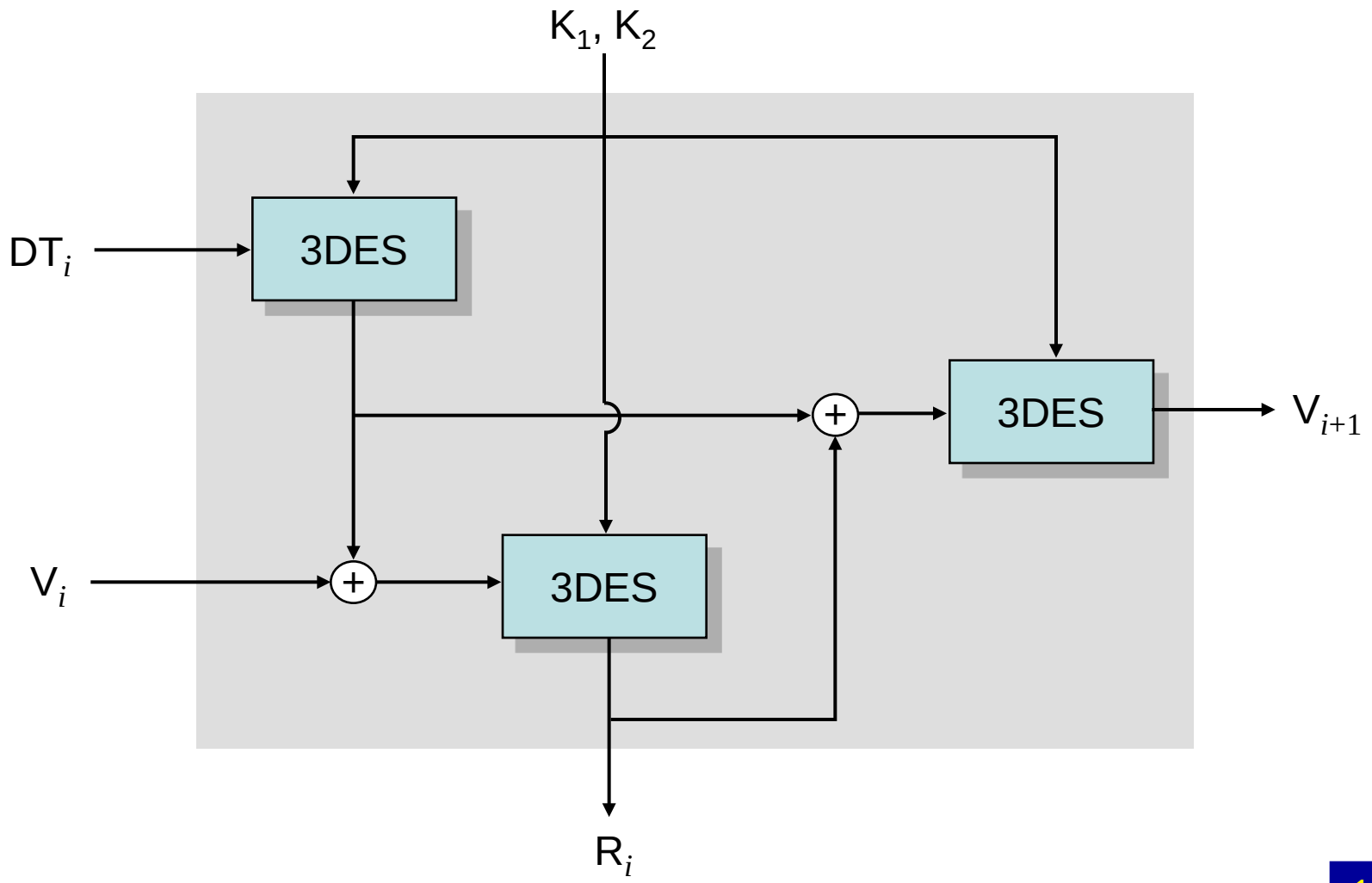
- true random numbers
  - used to generate public key – private key pairs
  - provide the initial seed for the pseudo-random number generator (PRNG)
  - provide additional input during pseudo-random number generation
- pseudo-random numbers
  - used to generate session keys and IVs

True random numbers

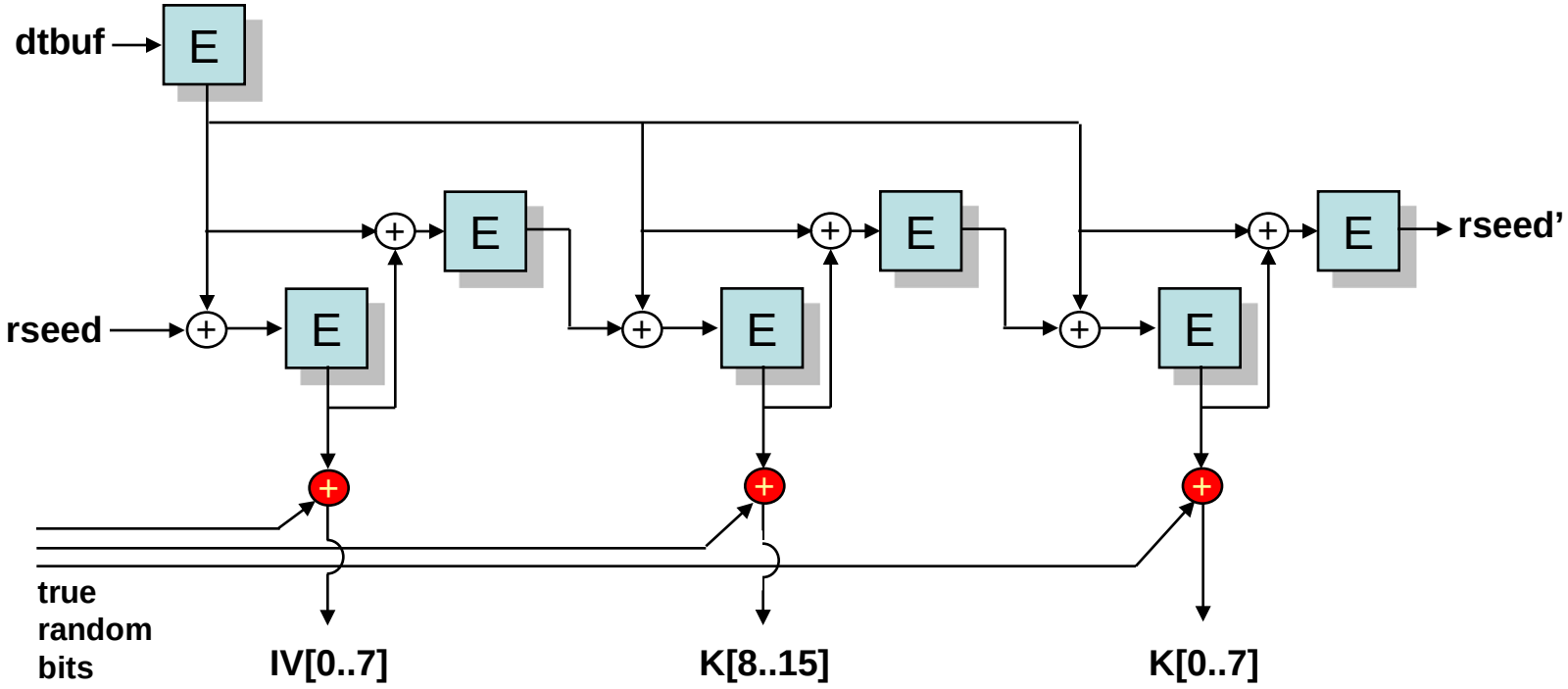
- PGP maintains a 256-byte buffer of random bits
- each time PGP expects a keystroke from the user, it records
  - the time when it starts waiting (32 bits)
  - the time when the key was pressed (32 bits)
  - the value of the key stroke (8 bits)
- the recorded information is used to generate a key
- the generated key is used to encrypt the current value of the random-bit buffer

Pseudo-random numbers

- based on the ANSI X9.17 PRNG standard



Pseudo-random numbers



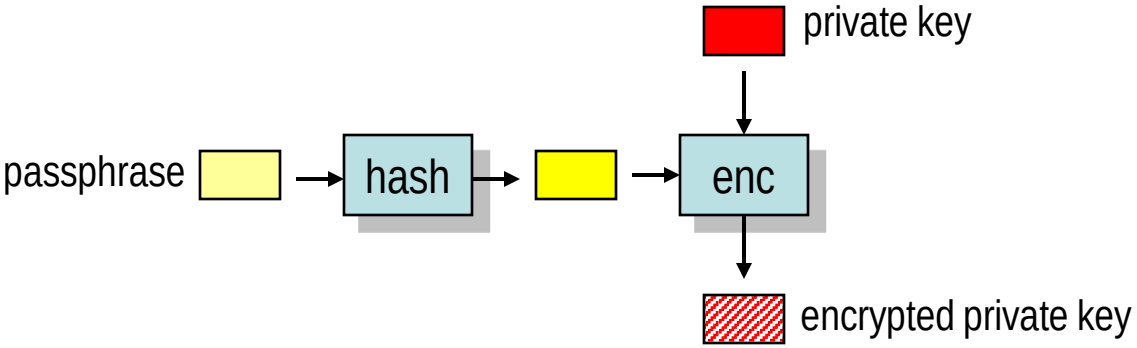
- **CAST-128 is used instead of 3DES with key *rkey***

Pseudo-random numbers

- dtbuf[0..3] = current time, dtbuf[4..7] = 0
- pre-wash
  - take the hash of the message
    - this has already been generated if the message is being signed
    - otherwise the first 4K of the message is hashed
  - use the result as a key, use a null IV, and encrypt $(\text{rkey, rseed})_{previous}$ in CFB mode
    - if $(\text{rkey, rseed})_{previous}$ is empty, it is filled up with true random bits
  - set $(\text{rkey, rseed})_{current}$ to the result of the encryption
- post-wash
  - generate 24 more bytes as before but without XORing in true random bytes
  - encrypt the result in CFB mode using K and IV
  - set $(\text{rkey, rseed})_{previous}$ to the result of the encryption

Private-key ring

- used to store the public key – private key pairs owned by a given user
- essentially a table, where each row contains the following entries:
  - timestamp
  - key ID (indexed)
  - public key
  - encrypted private key
  - user ID (indexed)



Public-key ring

- used to store public keys of other users
- a table, where each row contains the following entries:
  - timestamp
  - key ID (indexed)
  - public key
  - user ID (indexed)
  - owner trust
  - signature(s)
  - signature trust(s)
  - key legitimacy

Trust management

- owner trust
  - assigned by the user
  - possible values:
    - *unknown user*
    - *usually not trusted to sign*
    - *usually trusted to sign*
    - *always trusted to sign*
    - *ultimately trusted* (own key, present in private key ring)

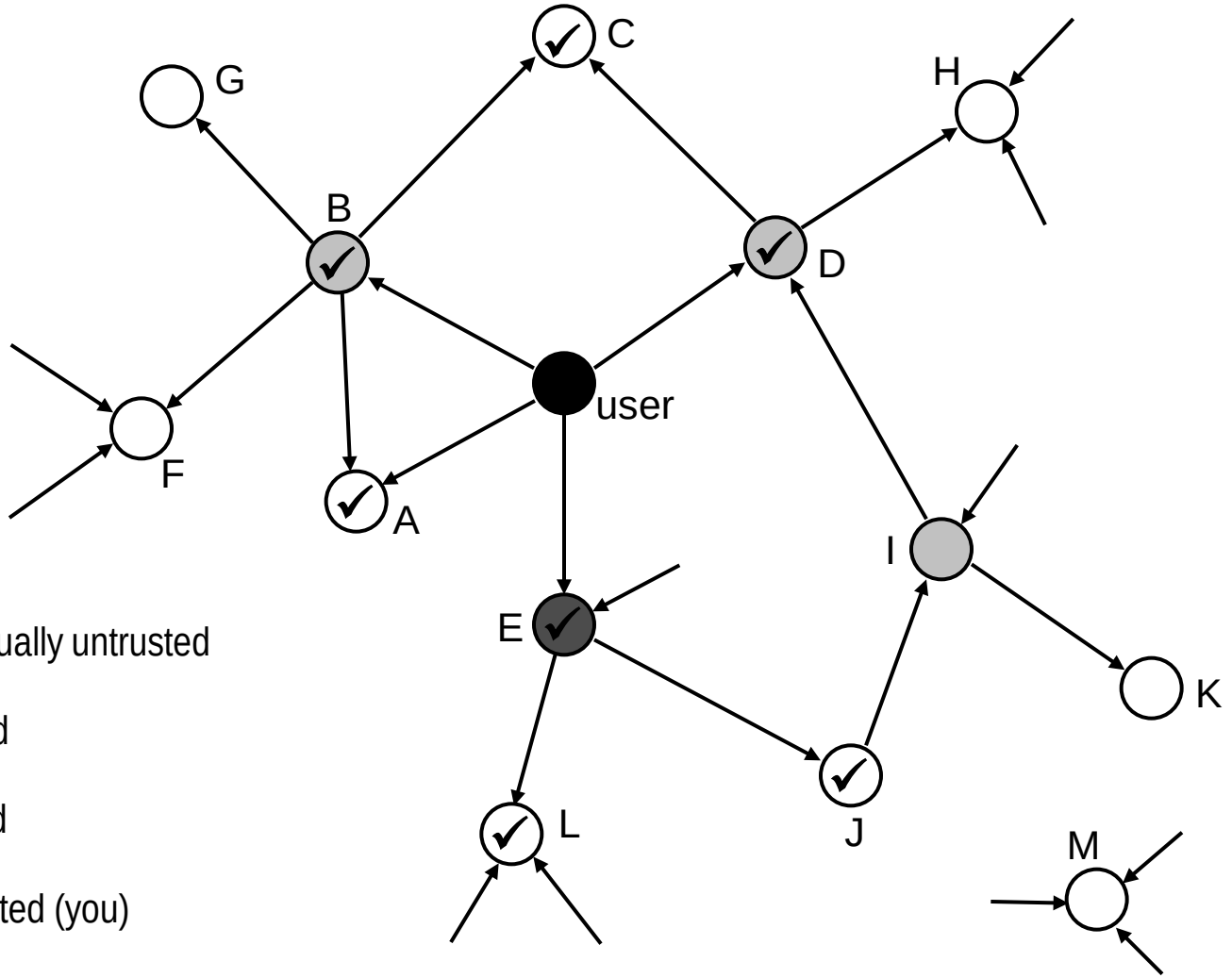
- signature trust
  - assigned by the PGP system
  - if the corresponding public key is already in the public-key ring, then its owner trust entry is copied into signature trust
  - otherwise, signature trust is set to *unknown user*

Trust management

- key legitimacy
  - computed by the PGP system
  - if at least one signature trust is ultimate, then the key legitimacy is 1 (complete)
  - otherwise, a weighted sum of the signature trust values is computed
    - always trusted signatures has a weight of 1/X
    - usually trusted signatures has a weight of 1/Y
    - X, Y are user-configurable parameters
  - example: X=2, Y=4
    - 1 ultimately trusted, or
    - 2 always trusted, or
    - 1 always trusted and 2 usually trusted, or
    - 4 usually trusted signatures are needed to obtain full legitimacy

Example – key legitimacy

X = 1, Y = 2



untrusted / usually untrusted

usually trusted

always trusted

ultimately trusted (you)

signature

✓ legitimate

Public-key revocation

- why to revoke a public key?
  - suspected to be compromised (private key got known by someone)
  - re-keying
- the owner issues a revocation certificate …
  - has a similar format to normal public-key certificates
  - contains the public key to be revoked
  - signed with the corresponding private key
- and disseminates it as widely and quickly as possible
- if a key is compromised:
  - e.g., Bob knows the private key of Alice
  - Bob can issue a revocation certificate to revoke the public key of Alice
  - even better for Alice

What is S/MIME?

- Secure / Multipurpose Internet Mail Extension
- a security enhancement to MIME
- provides similar services to PGP
- based on technology from RSA Security
- industry standard for commercial and organizational use
- RFC 2630, 2632, 2633

RFC 822

- defines a format for text messages to be sent using e-mail
- Internet standard
- structure of RFC 822 compliant messages
  - header lines (e.g., from: ..., to: ..., cc: ...)
  - blank line
  - body (the text to be sent)
- example

```
Date: Tue, 16 Jan 1998 10:37:17 (EST)
From: “Levente Buttyan” <buttyan@hit.bme.hu>
Subject: Test
To: afriend@otherhost.bme.hu

Blablabla
```

Problems with RFC 822 and SMTP

- executable files must be converted into ASCII
  - various schemes exist (e.g., Unix UUencode)
  - a standard is needed
- text data that includes special characters (e.g., Hungarian text)
- some servers
  - reject messages over a certain size
  - 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

- defines new message header fields
- defines a number of content formats (standardizing representation of multimedia contents)
- defines transfer encodings that protects the content from alteration by the mail system

MIME - New header fields

- MIME-Version
- Content-Type
  - describes the data contained in the body
  - receiving agent can pick an appropriate method to represent the content
- Content-Transfer-Encoding
  - indicates the type of the transformation that has been used to represent the body of the message
- Content-ID
- Content-Description
  - description of the object in the body of the message
  - useful when content is not readable (e.g., audio data)

MIME – Content types and subtypes

- text/plain, text/enriched
- image/jpeg, image/gif
- video/mpeg
- audio/basic
- application/postscript, application/octet-stream
- multipart/mixed, multipart/parallel, multipart/alternative, multipart/digest (each part is message/rfc822)
- message/rfc822, message/partial, message/external-body

MIME – Transfer encodings

- 7bit
  - short lines of ASCII characters
- 8bit
  - short lines of non-ASCII characters
- binary
  - non-ASCII characters
  - lines are not necessarily short
- quoted-printable
  - non-ASCII characters are converted into hexa numbers (e.g., =EF)
- base64 (radix 64)
  - 3 8-bit blocks into 4 6-bit blocks
- x-token
  - non-standard encoding

MIME – Example

MIME-Version: 1.0
From: Nathaniel Borenstein <nsb@nsb.fv.com>
To: Ned Freed <ned@innosoft.com>
Date: Fri, 07 Oct 1994 16:15:05 -0700 (PDT)
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
Content-type: text/plain; charset=US-ASCII

… Some text …

--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--

MIME – Example cont'd

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

This is <bold><italic>enriched.</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--

S/MIME services

- enveloped data (application/pkcs7-mime; smime-type = enveloped-data)
  - standard digital envelop
- signed data (application/pkcs7-mime; smime-type = signed-data)
  - standard digital signature (“hash and sign”)
  - content + signature is encoded using base64 encoding
- clear-signed data (multipart/signed)
  - standard digital signature
  - only the signature is encoded using base64
  - recipient without S/MIME capability can read the message but cannot verify the signature
- signed and enveloped data
  - signed and encrypted entities may be nested in any order

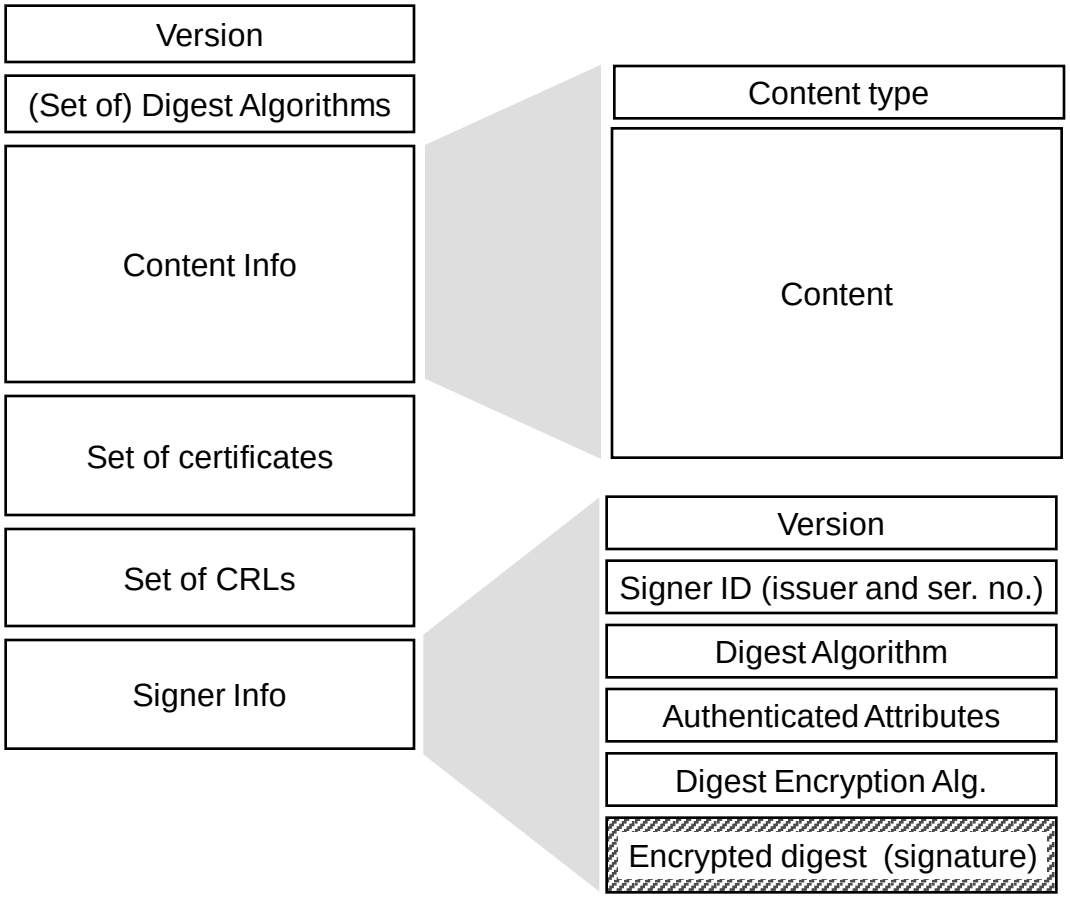
Cryptographic algorithms

- message digest
  - must: SHA-1
  - should (receiver): MD5 (backward compatibility)
- digital signature
  - must: DSS
  - should: RSA
- asymmetric-key encryption
  - must: ElGamal
  - should: RSA
- symmetric-key encryption
  - sender:
    - should: 3DES, RC2/40
  - receiver:
    - must: 3DES
    - should: RC2/40

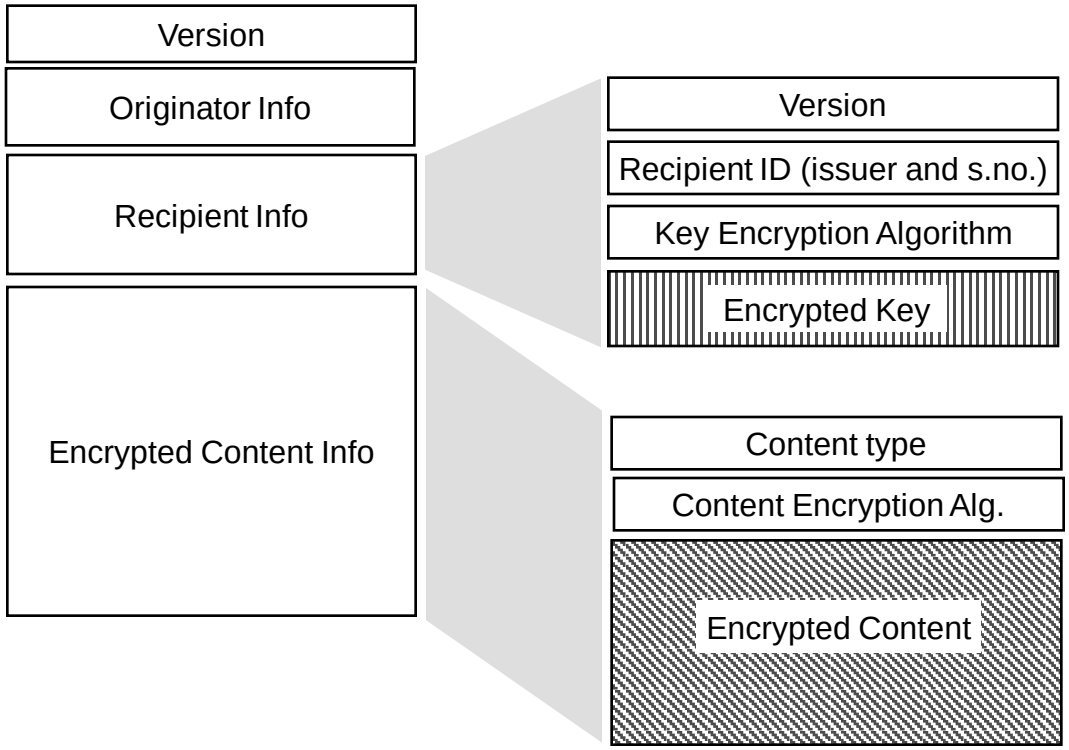
Securing a MIME entity

- MIME entity is prepared according to the normal rules for MIME message preparation
- prepared MIME entity is processed by S/MIME to produce a PKCS object
- the PKCS object is treated as message content and wrapped in MIME

PKCS7 “signed data”



PKCS7 “enveloped data”



Enveloped data – Example

Content-Type: application/pkcs7-mime; smime-type=enveloped-data; name=smime.p7m
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m

rfvbnj756tbBghyHhHUujhJhjH77n8HHGT9HG4VQpfyF467GhIGfHfYT6
7n8HHGghyHhHUujhJh4VQpfyF467GhIGfHfYGTrfvbnjT6jH7756tbB9H
f8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
0GhIGfHfQbnj756YT64V

Clear-signed data – Example

```
Content-Type: multipart/signed; protocol="application/pkcs7-signature";
  micalg=sha1; boundary=boundary42

--boundary42
Content-Type: text/plain

This is a clear-signed message.

--boundary42
Content-Type: application/pkcs7-signature; name=smime.p7s
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7s

ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6
4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj
n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
7GhIGfHfYT64VQbnj756

--boundary42--
```

Key management

- S/MIME certificates are X.509 conformant
- key management scheme is between strict certification hierarchy and PGP's web of trust
  - certificates are signed by certification authorities (CA)
  - key authentication is based on chain of certificates
  - users/managers are responsible to configure their clients with a list of trusted root keys