History of e-mail

- Early 1980: Secure/32, Charli Merritt, using PKC
- 1986: Mail Safe, RSADSI, DOS
- 1990:
  - PEM (Privacy Enhanced Mail)
    - RIPEM (Riordan’s Internet PEM)
    - TIS/PEM
  - PGP (Pretty Good Privacy)
  - S/MIME: Multimedia e-mail

Document of PEM

- RFC 1421, Part I: Message Encryption and Authentication Procedure
- RFC 1422, Part II: Certificate-based Key Management
- RFC 1423, Part III: Algorithms, Modes, and Identifiers
- RFC 1424, Part IV: Key Certification and Related Services
Design Environments of PEM

- Work with existing e-mail system in Internet
- Not restricted to particular host or OS
- Compatible with normal, non secure e-mail
- Performed on PC as well as on large system
- Compatible with a variety of key-management approach including manual distribution, centralized key distribution

Security Services of PEM

- Confidentiality
- Data origin authentication
- Message Integrity
- Non-repudiation of origin
- Key Management
Cryptographic Algorithm

- Data Encryption: DES in CBC
- Key Management: DES in ECB, CBC, and RSA
- MIC: RSA+MD2, RSA+MD5
- Digital Signature: RSA+MD2, RSA+MD5

Style of message

- Ordinary, unsecured data
- MIC-Clear: integrity and authentication, but no confidentiality (integrity-protected unmodified data)
- MIC-Only: MIC-Clear + encoding (integrity-protected encoded data)
- ENCRYPTED: MIC-Only + confidentiality (encoded encrypted integrity-protected data)
PEM Message

BEGIN-PRIVACY-ENHANCED-MESSAGE

<table>
<thead>
<tr>
<th>Processing Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content Domain</td>
</tr>
<tr>
<td>Message text encryption algorithm</td>
</tr>
<tr>
<td>Issuing authority</td>
</tr>
<tr>
<td>Version/expiration</td>
</tr>
<tr>
<td>Origination certificate</td>
</tr>
<tr>
<td>Originator key information</td>
</tr>
<tr>
<td>Issuer certificate</td>
</tr>
<tr>
<td>MIC information</td>
</tr>
<tr>
<td>Issuing authority</td>
</tr>
<tr>
<td>Version/expiration</td>
</tr>
<tr>
<td>Encrypted DEK</td>
</tr>
<tr>
<td>User Text</td>
</tr>
</tbody>
</table>

END-PRIVACY-ENHANCED-MESSAGE

Processing steps of PEM Message

- **Sending**
  - Canonicalization
  - Message Integrity and originator authentication
  - Encryption(optional)
  - Transmission encoding(optional)

- **Receiving**
  - Decoding(optional)
  - Decrypting(optional)
  - Verifying message integrity and authenticity
  - Translation
Certification Hierarchy

IPRA
- HACA
  - HACA
    - individual
  - HACA
  - individual
- DACA
  - CA
  - individual
- NACA
  - CA
  - individual
  - CA
  - individual
  - CA
  - individual
  - CA
  - individual
  - CA
  - individual
  - etc

HA: High Assurance, DA: Discretionary Assurance, NA: No Assurance

PGP

- Program for confidentiality and authentication service
- Select best available algorithm
  - Integrate algorithms into general-purpose
  - Made the package and its document, including source code, freely available via Internet
  - Low-cost commercial version by Viacrypt and Public-domain version
Background of PGP

- Available in various platforms
- Use algorithm survived extensively public review like RSA, DSS, DH, CAST-128, IDEA and 3DES, SHA-1
- Wide range of applicability from cooperation to individual
- Not developed by, nor controlled by, any government and standards organization

History of PGP(I)

- Designed by Phil Zimmerman
  - High security
  - public domain S/W
  - popular for personal use
- PGP Classic : Can’t handle Internet Mail
  - PGP v.1.0 : ‘91.6
  - PGP v.2.0 : ‘92. 9
  - PGP v.2.3a : ‘93. 7 (last version of PGP didn’t use RSAREF)
  - PGP v.2.4 : original ViaCrypt PGP
  - PGP v.2.5 : Interim release of PGP with RSAREF
  - PGP v.2.6 : Freeware version of PGP
  - PGP v.2.7 : Commercial version by ViaCrypt
History of PGP(II)

- **4 versions**
  - PGP Classic: non commercial use
  - PGP 5.0: Improve security but don’t adapt RSA
  - PGP/MIME
    - MIME-based
    - Use special certificate
    - Handle Internet Mail
  - OpenPGP
  - Use DH, DSA, SHA-1
  - Interoperability with S/MIME

Features of PGP

<table>
<thead>
<tr>
<th>Function</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital Signature</td>
<td>DSS/SHA or RSA/SHA</td>
</tr>
<tr>
<td>Message Encryption</td>
<td>CAST-128 or IDEA or 3DES (64bCFB) w/ DH or RSA</td>
</tr>
<tr>
<td>Compression</td>
<td>ZIP</td>
</tr>
<tr>
<td>E-mail compatibility</td>
<td>Radix 64</td>
</tr>
<tr>
<td>Segmentation</td>
<td></td>
</tr>
</tbody>
</table>

(Note) Signing before compression
Encryption after compression
Notation

- $K_s$: session key for conventional algorithm
- $K_{Ra}$: Private key of user A for PKC
- $K_{Ua}$: Public key of user A for PKC
- $EP$: PK encryption
- $DP$: PK decryption
- $EC$: conventional encryption
- $DC$: conventional decryption
- $H$: Hash function
- $||$: concatenation
- $Z$: compression
- $R64$: conversion to radix 64 ASCII format

Security Service in PGP(I)

(a) Authentication only

(b) Confidentiality only

(c)ICU Kwangjo Kim
Security Service in PGP(II)

Steps of sending a message

(c) Confidentiality and Authentication

(c) ICU Kwangjo KIm
Steps of receiving a message

1. Convert from radix 64
   \[ X \leftarrow R64^{-1}[X] \]

2. Confidentiality Req'd?
   \[ \begin{array}{c}
   Y \\
   N
   \end{array} \]

   - Y: Decrypt \{key, X\}
     \[ K_s \leftarrow D_{Ks}[K_s]; X \leftarrow D_K[X] \]
   - N: Decompress \( X \leftarrow Z^{-1}(X) \)

3. Signature Req'd?
   \[ \begin{array}{c}
   Y \\
   N
   \end{array} \]

   - Y: Strip signature from X
     Verify signature
   - N: [Diagram continues with arrows and decision branches]

Message format (A->B)

- Session Key Component
  - Session Key (Ks)
  - Timestamp
  - Key ID of sender's public key (KUb)
  - Leading 2 octets of message digest
  - File Name
  - Message

- Encryption:
  - \( E_{KUb} \): encryption with user b’s public key
  - \( E_{KRa} \): encryption with user b’s private key
  - \( E_{Ks} \): encryption with session key

- Compression:
  - ZIP

- Conversion:
  - R64: Radix-94 conversion function

(c)ICU Kwangjo KIm
Key Management

- One-time session key: 128 bit for CAST or IDEA, 168 bit for 3DES)
- Public Key
- Private Key
- Passphrase-based conventional key

Key Rings of PGP

Private Key ring: store his own public and private keys

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>KeyID</th>
<th>Public Key</th>
<th>Encrypted Private key</th>
<th>UserID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ti</td>
<td>KU $\text{mod } 2^{64}$</td>
<td>KUI</td>
<td>$E_{\text{RXP}}[KRI]$</td>
<td>User i</td>
</tr>
</tbody>
</table>

Public Key ring: store all known entities' public key

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>KeyID</th>
<th>Public Key</th>
<th>Owner Trust</th>
<th>UserID</th>
<th>Key Legitimacy</th>
<th>Signature(s)</th>
<th>Signature Trust(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ti</td>
<td>KU $\text{mod } 2^{64}$</td>
<td>KUI</td>
<td>trust_flagi</td>
<td>User i</td>
<td>trust_flagi</td>
<td>ERj[H([KUI])]</td>
<td>complete marginal</td>
</tr>
</tbody>
</table>
Use of private key

- Using IDEA, store encrypted key
  - User selects passphrase
  - When generating private/public key pairs, use passphrase
  - Passphrase is inputted to hash ft. MD5 (SHA-1), Use 128 (160)-bit hash value as key of IDEA
- After use, delete it from system

Message sending (Detailed)
Message receiving (Detailed)

- Passphrase
- Private key ring
- Private Key
- Encrypted Private key
- Receiver's Key ID
- Encrypted Session Key
- Encrypted message + signature
- Select

Sender's Key ID
Encrypted Digest
Message

Passphrase
DC
H

Select
Public key ring
Public Key
KUa

DC
H

Session Key
Ks

Encrypted Session Key
DP

Encrypted Private key
DC
H

Senders Key ID

Distribution of Public Key

- Direct delivery (floppy disk, mail, ..)
- Sending e-mail and confirm by telephone
- TTP
- CA

B → A
 Key (by e-mail)

(128-bit MD5 digest)

fingerprint
(by telephone)

B (matching)
PGP’s Trust Model

- Key's owner is trusted by you to sign keys
- Key's owner is partly trusted by you to sign keys
- Key is deemed legitimate by you
- Unknown signatory

Revocation of Public key

- Issue public key revocation signature
  - Similar form of usual Signature Certificate
  - Signature using secret key of public key to be revoked
  - Propagate as many as possible
- All public keys signed by revocated key
  - Make Owner_trust and key_legitimacy to untrust