# Email Security

“Why do we have to hide from the police, Daddy?”
“Because we use PGP, son. They use S/MIME”

## Email Security

Problems with using email for secure communications include

- Doesn’t handle binary data
- Messages may be modified by the mail transport mechanism
  - Trailing spaces deleted
  - Tabs ↔ spaces
  - Character set conversion
  - Lines wrapped/truncated
- Message headers mutate considerably in transit

Data formats have to be carefully designed to avoid these problems

# Email Security Requirements

Main requirements

- Confidentiality
- Authentication
- Integrity

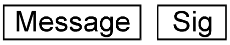
Other requirements

- Non-repudiation
- Proof of submission
- Proof of delivery
- Anonymity
- Revocability
- Resistance to traffic analysis

Many of these are difficult or impossible to achieve

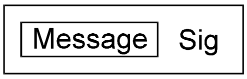
# Security Mechanisms

Detached signature



- Leaves the original message untouched
- Signature can be transmitted/stored separately
- Message can still be used without the security software

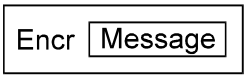
Signed message



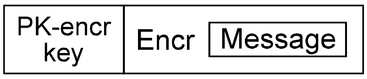
- Signature is always included with the data

## Security Mechanisms (ctd)

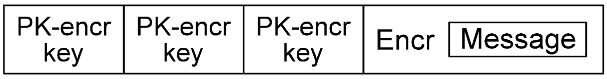
Encrypted message


Encr Message

Usually implemented using public-key encryption


PK-encr key | Encr Message

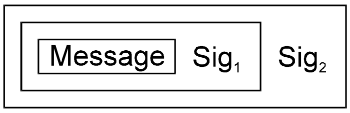
Mailing lists use one public-key encrypted header per recipient


PK-encr key | PK-encr key | PK-encr key | Encr Message

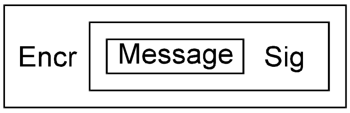
- Any of the corresponding private keys can decrypt the session key and therefore the message

## Security Mechanisms (ctd)

Countersigned data


Message Sig$_1$ Sig$_2$

Encrypted and signed data


Encr Message Sig

- Always sign first, then encrypt

  S( E( “Pay the signer $1000” ))

  vs.

  E( S( “Pay the signer $1000” ))

## PEM

Privacy Enhanced Mail, 1987

Attempt to add security to SMTP (MIME didn’t exist yet)

- Without MIME to help, this wasn’t easy

Attempt to build a CA hierarchy along X.500 lines

- Without X.500 available, this wasn’t easy

Solved the data formatting problem with base64 encoding

- Encode 3 binary bytes as 4 ASCII characters
- The same encoding was later used in PGP, MIME, ...

## PEM Protection Types

Unsecured data

Integrity-protected (MIC-CLEAR)

- MIC = message integrity check = digital signature

Integrity-protected encoded (MIC-ONLY)

Encrypted integrity-protected (ENCRYPTED)

General format based on RFC822 messages

```
-----BEGIN PRIVACY-ENHANCED MESSAGE-----
Type: Value                Encapsulated header
Type: Value
Type: Value
                           Blank line
Data                       Encapsulated content
-----END PRIVACY-ENHANCED MESSAGE-----
```

# PEM Protection Types (ctd)

## MIC-ONLY

```
-----BEGIN PRIVACY-ENHANCED MESSAGE-----
Proc-Type: 4,MIC-ONLY
Content-Domain: RFC822
Originator-Certificate:
 MIIBlTCCAScCAWUwDQYJKoZIhvcNAQECBQAwUTELMAkGA1UEBhMCVVMxIDAeBgNV
 BAoTF1JTQSDRiNKcOCaCoLAyaXR5LCBJbmMuMQ8wDQYDVQQLEwZCFNOrDDExDzAN
 …
 iWlFPuN5jJ79Khfg7ASFxskYkEMjRNZV/HZDZQEhtVaU7Jxfzs2wfX5byMp2X3U/
 5XUXGx7qusDgHQGs7Jk9W8CW1fuSWUgN4w==
Issuer-Certificate:
 MIIB3FNoRDgCAQowDQYJKoZIhvcNAQECBQAwTzEWiGEbLUMenKraFTMxIDAeBgNV
 BAoTF1JTQSBEYXRhIFNlY3VyaXR5LCBJbmMuMQ8wDQYDVQQLEwZCZXRhIDExDTAL
 …
 dD2jMZ/3HsyWKWgSF0eH/AJB3qr9zosG47pyMnTf3aSy2nBO7CMxpUWRBcXUpE+x
 EREZd9++32ofGBIXaialnOgVUn0OzSYgugiQReSIsTKEYeSCrOWizEs5wUJ35a5h
MIC-Info: RSA-MD5,RSA,
 jV2OfH+nnXFNorDL8kPAad/mSQlTDZlbVuxvZAOVRZ5q5+Ejl5bQvqNeqOUNQjr6
 EtE7K2QDeVMCyXsdJlA8fA==

LSBBIG1lc3NhZ2UgZm9yIHVzZSBpbiB0ZXN0aW5nLg0KLSBGb2xsb3dpbmcgaXMg
YSBibGFuayBsaW5lOg0KDQpUaGlzIGlzIHRoZSBlbmQuDQo=
-----END PRIVACY-ENHANCED MESSAGE-----
```

# PEM Protection Types (ctd)

## ENCRYPTED (explicitly includes MIC)

```
-----BEGIN PRIVACY-ENHANCED MESSAGE-----
Proc-Type: 4,ENCRYPTED
Content-Domain: RFC822
DEK-Info: DES-CBC,BFF968AA74691AC1
Originator-Certificate:
 MIIBlTCCAScCAWUwDQYJKoZIhvcNAQECBQAwUTELMAkGA1UEBhMCVVMxIDAeBgNV
 …
 5XUXGx7qusDgHQGs7Jk9W8CW1fuSWUgN4w==
Issuer-Certificate:
 MIIB3DCCAUgCAQowDQYJKoZIhvcNAQECBQAwTzELMAkGA1UEBhMCVVMxIDAeBgNV
 …
 EREZd9++32ofGBIXaialnOgVUn0OzSYgugiQ077nJLDUj0hQehCizEs5wUJ35a5h
MIC-Info: RSA-MD5,RSA,
 UdFJR8u/TIGhfH65ieewe2lOW4tooa3vZCvVNGBZirf/7nrgzWDABz8w9NsXSexv
 AjRFbHoNPzBuxwmOAFeA0HJszL4yBvhG
```

*Continues*

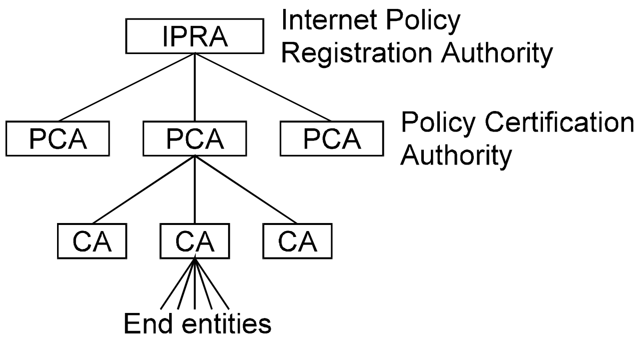
# PEM Protection Types (ctd)

*Continued*

```
Recipient-ID-Asymmetric:
 MFExCzAJBgNVBAYTAlVTMSAwHgYDVQQKExdSU0EgRGF0YSBTZWN1cml0eSwgSW5j
 LjEPMA0GA1UECxMGQmV0YSAxMQ8wDQYDVQQLEwZOT1RBUlk=,66
Key-Info: RSA,
 O6BS1ww9CTyHPtS3bMLD+L0hejdvX6Qv1HK2ds2sQPEaXhX8EhvVphHYTjwekdWv
 7x0Z3Jx2vTAhOYHMcqqCjA==

qeWlj/YJ2Uf5ng9yznPbtD0mYloSwIuV9FRYx+gzY+8iXd/NQrXHfi6/MhPfPF3d
jIqCJAxvld2xgqQimUzoS1a4r7kQQ5c/Iua4LqKeq3ciFzEv/MbZhA==
-----END PRIVACY-ENHANCED MESSAGE-----
```

# PEM CA Hierarchy


IPRA
Internet Policy Registration Authority
PCA
PCA
PCA
Policy Certification Authority
CA
CA
CA
End entities

Single grand unified hierarchy

- No choice of alternate CAs, policies, …

Although PEM itself failed, the PEM CA terminology still crops up in various products

## PEM CA Hierarchy (ctd)

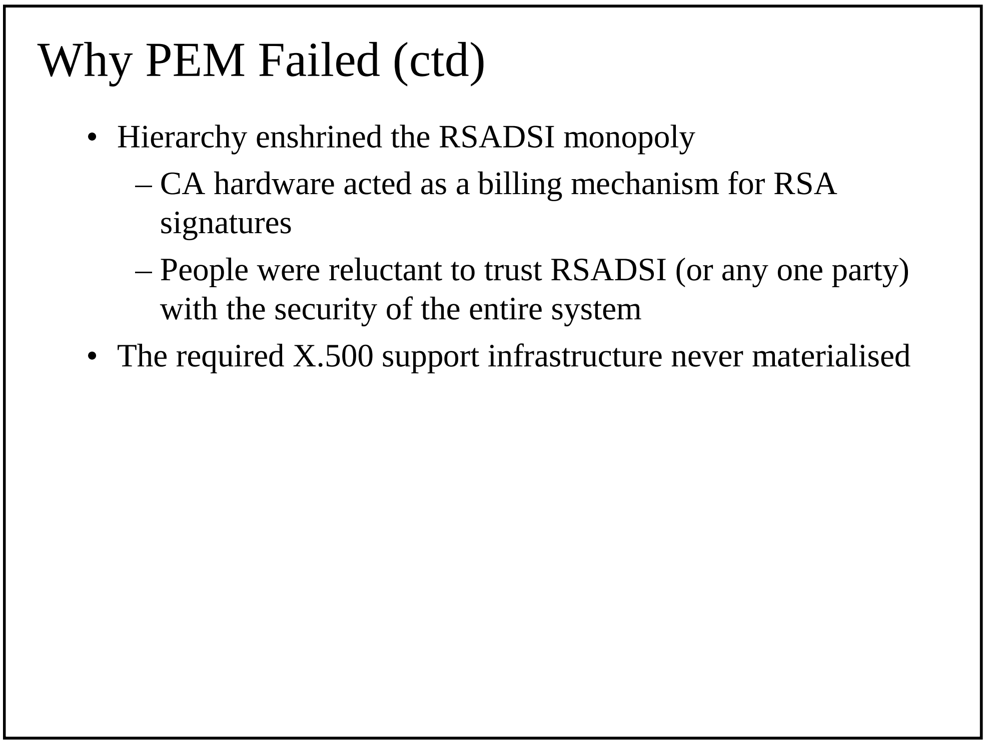
Policy CA's guarantee certain things such as uniqueness of names

- High-assurance policies (secure hardware, drug tests for users, etc)
  - Can't issue certificates to anything other than other high-assurance CA's
- Standard CA's
- No-assurance CA's (persona CA's)
  - Certificate vending machines
  - Clown suit certificates

## Why PEM Failed

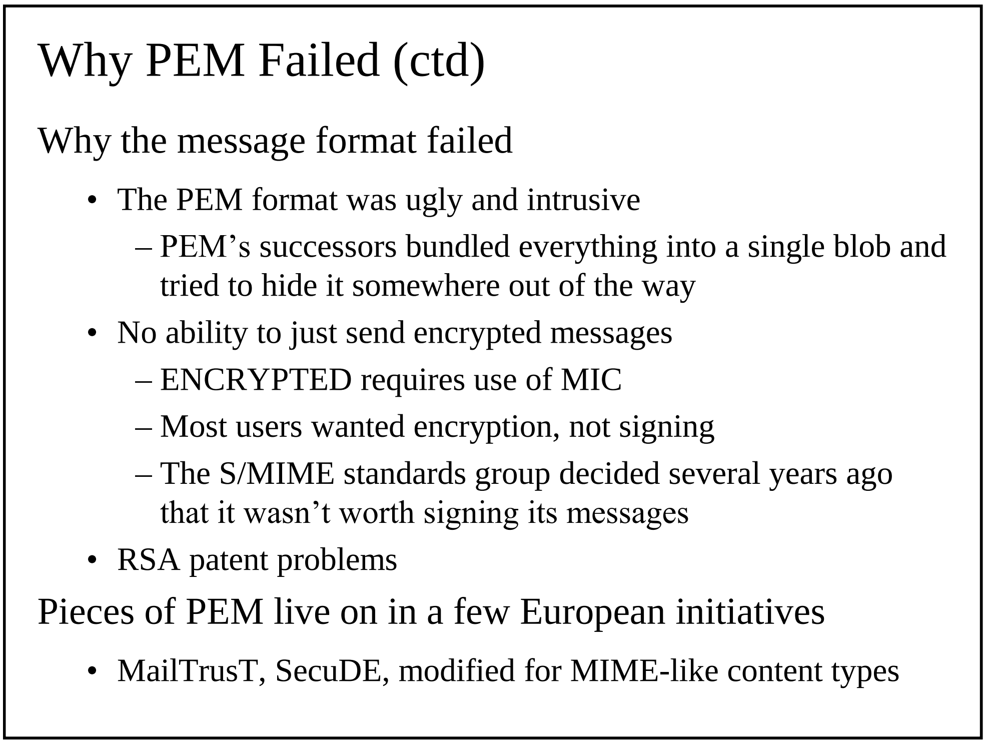
Why the CA's failed

- The Internet uses email addresses, not X.500 names
  - Actually, no-one uses X.500 names
- CA's for commercial organisations and universities can't meet the same requirements as government defence contractors for high-assurance CA's
  - Later versions of PEM added lower-assurance CA hierarchies to fix this
- CA hardware was always just a few months away
  - When it arrived, it was hideously expensive
- CA's job was made so onerous no-one wanted it
  - Later versions made it easier



## Why PEM Failed (ctd)

- Hierarchy enshrined the RSADSI monopoly
  - CA hardware acted as a billing mechanism for RSA signatures
  - People were reluctant to trust RSADSI (or any one party) with the security of the entire system
- The required X.500 support infrastructure never materialised



## Why PEM Failed (ctd)

Why the message format failed

- The PEM format was ugly and intrusive
  - PEM’s successors bundled everything into a single blob and tried to hide it somewhere out of the way
- No ability to just send encrypted messages
  - ENCRYPTED requires use of MIC
  - Most users wanted encryption, not signing
  - The S/MIME standards group decided several years ago that it wasn’t worth signing its messages
- RSA patent problems

Pieces of PEM live on in a few European initiatives

- MailTrusT, SecuDE, modified for MIME-like content types

# PGP

Pretty Good Privacy

- Hastily released in June 1991 by Phil Zimmerman (PRZ) in response to S.266
- MD4 + RSA signatures and key exchange
- Bass-O-Matic encryption
- LZH data compression
- uuencoding ASCII armour
- Data format based on a 1986 paper by PRZ

PGP was immediately distributed worldwide via a Usenet post

# PGP (ctd)

PGP 1.0 lead to an international effort to develope 2.0

- Bass-O-Matic was weak, replaced by the recently-developed IDEA
- MD4 " " " " MD5
- LZH replaced by the newly-developed InfoZip (now zlib)
- uuencoding replaced with the then-new base64 encoding
- Ports for Unix, Amiga, Atari, VMS added
- Internationalisation support added

## Legal Problems

PGP was the centre of an ongoing legal dispute with RSADSI over patents

- RSADSI released the free RSAREF implementation for (non-commercial) PEM use
- PGP 2.6 was altered to use RSAREF in the US
- Commercial versions were sold by Viacrypt, who had an RSA license

Later versions deprecated RSA in favour of the non-patented Elgamal

- Elgamal was referred to in the documentation as Diffie-Hellman for no known reason
  - Both are DLP algorithms, but DH != Elgamal

## Government Problems

In early 1993, someone apparently told US Customs that PRZ was exporting misappropriated crypto code

- US Customs investigation escalated into a Federal Grand Jury (US Attorney) in September 1993

They were pretty serious, e.g.:

26 February 1995: San Francisco Examiner and SF Chronicle publish an article criticising the government’s stand on encryption and the PGP investigation

27 February 1995: Author of the article was subpoena’d to appear before the Grand Jury

Investigation was dropped in January 1996 with no charges laid

# PGP Message Formats

Unsecured

Compressed

Signed/clearsigned

Encrypted

+ optional encoding

General format

`-----BEGIN PGP` *message type*`-----`<br>
*data*<br>
`-----END PGP` *message type*`-----`

# PGP Message Formats (ctd)

Clearsigned message

```
-----BEGIN PGP SIGNED MESSAGE-----

We've got into Peter's presentation.  Yours is next.  Resistance is
   useless.

-----BEGIN PGP SIGNATURE-----
Version: 2.3

iQCVAgUBK9IAl2v14aSAK9PNAQEvxgQAoXrviAggvpVRDLWzCHbNQo6yHuNuj8my
cvPx2zVkhHjzkfs5lUW6z63rRwejvHxegV79EX4xzsssWVUzbLvyQUkGS08SZ2Eq
bLSuij9aFXalv5gJ4jB/hU40qvU6I7gKKrVgtLxEYpkvXFd+tFC4n9HovumvNRUc
ve5ZY8988pY=
=NOcG
-----END PGP SIGNATURE-----
```

- Remember that all this predates MIME
- Also had to work with things like Fidonet

# PGP Message Formats (ctd)

## Anything else

```
-----BEGIN PGP MESSAGE-----
Version: 2.3

hQEMAlkhsM216BqRAQf/f938A6hglX51/hwa42oCdrQDRGw6HJd+5OqQX/58JB8Y
UAlrYBHYZ5md46ety62phvbwfsNuF9igSx2943CHrnuIVtkSXZRpKogtSE1oMfab
5ivD4I+h3Xk0Jpkn5SXYAzC6/cjAZAZSJjoqy28LBIwzlfNNqrzIuEW8lbLPWAt1
eqdS18ukiOUvnQAI1QfJipGUG+Db1KnpqJP7wHUl/4RG1Qi50p3BCDIspC8jzQ/y
GsKFlckA132dMx6b80vsUZga/tmJOwrgBjSbnOJ8UzLrNe+GjFRyBS+qGuKgLd9M
ymYgMyNOqo/LXALSlLIc/Dr1nKC0cAC01a/RZ00w4KYAAFrxX9a1BQq1nb40/OSB
CgrPqi61jBks2NW2EPoIC7nV5xLjflZwlRjY/V5sZS6XDycJ9YOf6fOclNwCoBsB
HRshmNtMHH2tq2//OozKZ8/GHGNysN8QQWNQYElgRCgH3ou1E+CJoyoPwrMqjSYC
oGp4fezQpiI83Ve/QMMV276KntTFLRpQ2H+lLDvX9Wfjg1+xTw==
=ZuOF
-----END PGP MESSAGE-----
```

# PGP Key Formats

Unlike PEM, PGP also defined public/private key formats

- OpenSSL's 'PEM' format is a homebrew invention

| | |
|---|---|
| KeyID | |
| Public key | Key trust |
| UserID | UserID trust |
| Signature | Sig.trust |
| Signature | Sig.trust |

- Key trust = how much the key is trusted to sign things (set by the user)
- userID trust = how much the userID is trusted to belong to this key
- Signing trust = copy of the signing key's trust

PGP calculates userID trust = sum of signing trusts

# PGP Trust

UserID trust = trust of binding between userID and key

Key trust = trust of key owner

Example: UserID = Politician

- UserID trust = High
- Key trust = Low

Trust levels

- Unknown
- None
- Casual
- Heavy-duty

# PGP Trust (ctd)

Each key can contain multiple userIDs with their own trust levels

- userID = Peter Gutmann, trust = high
- userID = University Vice-Chancellor, trust = none

Keys are revoked with a signed revocation (suicide note) that PGP adds to the key

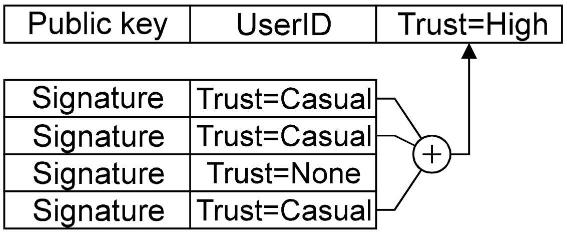
- Unlike X.509, you don’t need to go to an external agency to cancel your key

PGP philosophy: Scram switch, in case of an emergency shut down as quickly as possible

- X.509 philosophy: DoS, make it as difficult as possible to revoke a key

# PGP Trust Computation

Trust levels are automatically computed by PGP



User can define the required trust levels (e.g. 3 casuals = 1 high)

In practice, the web of trust doesn't really deliver

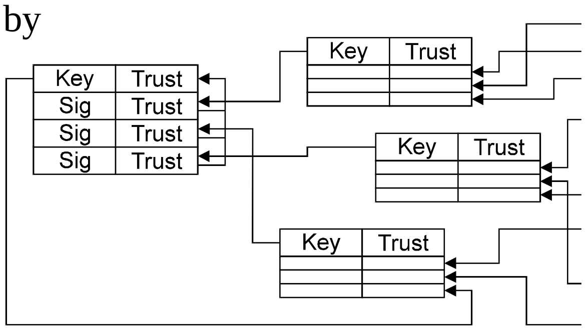
- It can also be used hierarchically, like X.509

# PGP Keyrings

One or more keys stored together constitute a keyring

Keys are looked up by

- userID (free-form name)
- keyID (64-bit value derived from the public key)



The owner's key is ultimately trusted and can convey this to other keys

## Key Distribution

Key distribution doesn’t rely on an existing infrastructure

- Email
- Personal contact
  - Keysigning services
- Keys on web pages
- PGP keyservers
  - email/HTTP interface to a PGP keyring
  - HKP = undocumented protocol based on variations of a student project

Verification by various out-of-band means (personal contact, phone, mail)

- PGP key fingerprint was specifically designed for this purpose

## Advantages of PGP over PEM

You can pick your own name(s)

You don’t have to register with an authority

PGP requires no support infrastructure

The trust mechanism more closely matches real life

Key/certificate distribution can be manual or automatic (just include it with the message)

# MIME-based Security

Multipurpose Internet Mail Extensions

- Provides a convenient mechanism for transferring composite data

Security-related information is sent as sections of a multipart message

- multipart/signed
- multipart/encrypted

Binary data is handled via base64 encoding

MIME-aware mailers can automatically process the security information (or at least hide it from the user)

# MIME-based Security (ctd)

General format

```
Content-Type: multipart/type; boundary="Boundary"
Content-Transfer-Encoding: base64

--Boundary
encryption info

--Boundary
message

--Boundary
signature
--Boundary--
```

Both PEM and PGP were adapted to fit into the MIME framework

# MOSS

## MIME Object Security Services

- PEM shoehorned into MIME
- MOSS support was added to MIME types via `application/moss-signature` and `application/moss-keys`

# MOSS (ctd)

## MOSS Signed

```
Content-Type: multipart/signed; protocol="application/moss-
   signature"; micalg="rsa-md5"; boundary="Signed Message"

--Signed Message
Content-Type: text/plain

Support PGP: Show MOSS to your friends.

--Signed Message
Content-Type: application/moss-signature

Version: 5
Originator-ID:
  jV2OfH+nnXHU8bnL8kPAad/mSQlTDZlbVuxvZAOVRZ5q5+Ejl5bQvqNeqOUNQjr6
  EtE7K2QDeVMCyXsdJlA8fA==
MIC-Info: RSA-MD5,RSA,
 UdFJR8u/TIGhfH65ieewe2lOW4tooa3vZCvVNGBZirf/7nrgzWDABz8w9NsXSexv
 AjRFbHoNPzBuxwmOAFeA0HJszL4yBvhG

--Signed Message--
```

# MOSS (ctd)

## MOSS Encrypted

```
Content-Type: multipart/encrypted; protocol="application/moss-keys";
  boundary="Encrypted Message"

--Encrypted Message
Content-Type: application/moss-keys

Version: 5
DEK-Info: DES-CBC,BFF968AA74691AC1
Recipient-ID:
 MFExCzAJBgNVBAYTAlVTMSAwHgYDVQQKExdSU0EgRGF0YSBTZWN1cml0eSwgSW5j
 LjEPMA0GA1UECxMGQmV0YSAxMQ8wDQYDVQQLEwZOT1RBUlk=,66
Key-Info: RSA,
 O6BS1ww9CTyHPtS3bMLD+L0hejdvX6Qv1HK2ds2sQPEaXhX8EhvVphHYTjwekdWv
 7x0Z3Jx2vTAhOYHMcqqCjA==

--Encrypted Message
Content-Type: application/octet-stream

qeWlj/YJ2Uf5ng9yznPbtD0mYloSwIuV9FRYx+gzY+8iXd/NQrXHfi6/MhPfPF3d
jIqCJAxvld2xgqQimUzoS1a4r7kQQ5c/Iua4LqKeq3ciFzEv/MbZhA==

--Encrypted Message--
```

# PGP/MIME

PGP shoehorned into MIME

- PGP support added to MIME types via application/pgp-signature and application/pgp-encrypted

PGP already uses ‘--’ so PGP/MIME escapes this with ‘- ’

```
-----BEGIN PGP MESSAGE-----
```

becomes

```
- -----BEGIN PGP MESSAGE-----
```

# PGP/MIME (ctd)

## PGP/MIME Signed:

```
Content-Type: multipart/signed; protocol="application/pgp-signature";
   micalg=pgp-md5; boundary=Signed

--Signed
Content-Type: text/plain

Our message format is uglier than your message format!

--Signed
Content-Type: application/pgp-signature

- -----BEGIN PGP MESSAGE-----
Version: 2.6.2

iQCVAwUBMJrRF2N9oWBghPDJAQE9UQQAtl7LuRVndBjrk4EqYBIb3h5QXIX/LC//
jJV5bNvkZIGPIcEmI5iFd9boEgvpirHtIREEqLQRkYNoBActFBZmh9GC3C041WGq
uMbrbxc+nIs1TIKlA08rVi9ig/2Yh7LFrK5Ein57U/W72vgSxLhe/zhdfolT9Brn
HOxEa44b+EI=
=ndaj
- -----END PGP MESSAGE-----
--Signed--
```

# PGP/MIME (ctd)

## PGP/MIME Encrypted

```
Content-Type: multipart/encrypted; protocol="application/pgp-
   encrypted"; boundary=Encrypted

--Encrypted
Content-Type: application/pgp-encrypted

Version: 1

--Encrypted
Content-Type: application/octet-stream

-----BEGIN PGP MESSAGE-----
Version: 2.6.2

hIwDY32hYGCE8MkBA/wOu7d45aUxF4Q0RKJprD3v5Z9K1YcRJ2fve87lMlDlx4Oj
g9VGQxFeGqzykzmykU6A26MSMexR4ApeeON6xzZWfo+0yOqAq6lb46wsvldZ96YA
AABH78hyX7YX4uT1tNCWEIIBoqqvCeIMpp7UQ2IzBrXg6GtukS8NxbukLeamqVW3
1yt21DYOjuLzcMNe/JNsD9vDVCvOOG3OCi8=
=zzaA
-----END PGP MESSAGE-----

--Encrypted--
```

# MOSS and PGP/MIME

MOSS never took off

PGP/MIME never took off either

# S/MIME

Originally based on proprietary RSADSI standards wrapped in MIME

- PKCS, Public Key Cryptography Standards
  - RC2, RC4 for data encryption
  - PKCS #1, RSA encryption, for key exchange
  - PKCS #7, cryptographic message syntax, for message formatting

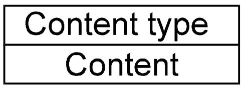
Newer versions added non-proprietary and non-patented ciphers

- Widely-supported, little-used
  - Every Windows box and many Unix boxes have this built in
  - Outlook makes it (moderately) easy to use

# CMS

Cryptographic Message Syntax

- Type-and-value format



Content type

Content

Data content types

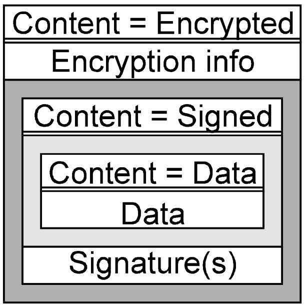
- Data
- Signed data
- Encrypted data (conventional encryption)
- Enveloped data (PKC-encrypted)
- Authenticated (MAC'd) data
- Compressed data

# CMS (ctd)

Other content types possible

- Key management messages
- Protocol-specific message data

Content can be arbitrarily nested



Content = Encrypted

Encryption info

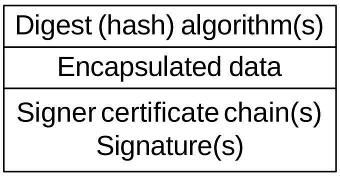
Content = Signed

Content = Data

Data

Signature(s)

# Signed Data Format



Digest (hash) algorithm(s)

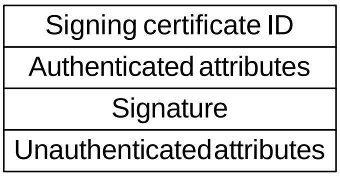
Encapsulated data

Signer certificate chain(s)
Signature(s)

Presence of hash algorithm information before the data and certificates before the signatures allows one-pass processing

- Streaming implementations can generate and verify messages on the fly

# Signature Format



Signing certificate ID

Authenticated attributes

Signature

Unauthenticated attributes

Authenticated attributes are signed along with the encapsulated content

- Signing time
- Signature type
  - “I agree completely”
  - “I agree in principle”
  - “I disagree but can’t be bothered going into the details”
  - “A flunky handed me this to sign”

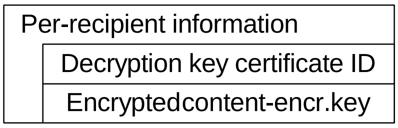
## Signature Format (ctd)

- Receipt request
- Security label
- Mailing list information

Unauthenticated attributes provide a means of adding further information without breaking the original signature

- Countersignature
  - Countersigns an existing signature
  - Signs the signature on the content rather than the content itself, so the other content doesn't have to be present
  - Countersignatures can contain further countersignatures

## Enveloped Data Format



Per-recipient information

Decryption key certificate ID

Encryptedcontent-encr.key

Newer versions added support for further mechanisms like previously distributed shared content-encryption keys

- Makes mailing-list support easier

## CMS → S/MIME

Wrap each individual CMS layer in MIME

base64 encode + wrap content<br>Encode as CMS data<br>base64 encode + wrap content<br>Encode as CMS signed data<br>base64 encode + wrap content<br>Encode as CMS enveloped data<br>base64 encode + wrap content

Result is 2:1 message expansion

## S/MIME Problems

Earlier versions used mostly crippled crypto

- Only way to interoperate was 40-bit RC2
  - RC2/40 is still the lowest-common-denominator default
  - User is given no warning of the use of crippled crypto
    - Message forwarding may result in a security downgrade
- S/MIME-cracking screen saver released in 1997
  - Performs an optimised attack using RC2 key setup cycles
  - Looks for the MIME header in the decrypted data

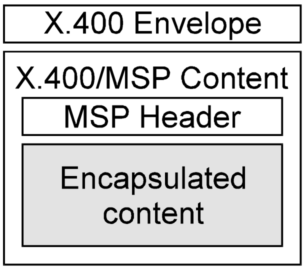
Original S/MIME was based on patented RSA and proprietary RC2, rejected by the IETF as a standard

- IETF developed S/MIME v3 using strong crypto and non-patented, non-proprietary technology

# MSP

Message Security Protocol, used in the Defence Messaging System (DMS)

- X.400 message contains an envelope + content
- MSP encapsulates the X.400 content and adds a security header



X.400 Envelope

X.400/MSP Content

MSP Header

Encapsulated content

X.400 security required using (and trusting) an X.400 MTA; MSP requires only trusted endpoints

- MSP was later used with MIME

# MSP Services

Services provided

- Authentication
- Integrity
- Confidentiality
- Non-repudiation of origin (via message signatures)
- Non-repudiation of delivery (via signed receipts)

MSP also provides rule-based access control (RBAC) based on message sensitivity and classification levels of sender, receiver, and workstation

- Receiving MUA checks that the receiver and workstation are cleared for the messages security classification
- MSP rule-based access control (RBAC) ≠ role-based access control (also RBAC)

## MSP Certificates

MSP defines three X.509 certificate types

- Signature-only
- Encryption (key management) only
- Signature and encryption (two keys in one certificate)
  - Non-standard extension to X.509v1

Certificate also includes R(ule)BAC authorisations

## MSP Protection Types

MSP Signature

- MUA/MLA signs with signature-only certificate

Non-repudiation

- User signs with signature or dual-key certificate

Confidentiality, integrity, R(ule)BAC

- Encrypted with key management or dual-key certificate

Non-repudiation + confidentiality, integrity, R(ule)BAC

- Sign + encrypt using either signature and key management certificates or dual-key certificate

Any of the above can be combined with MSP signatures

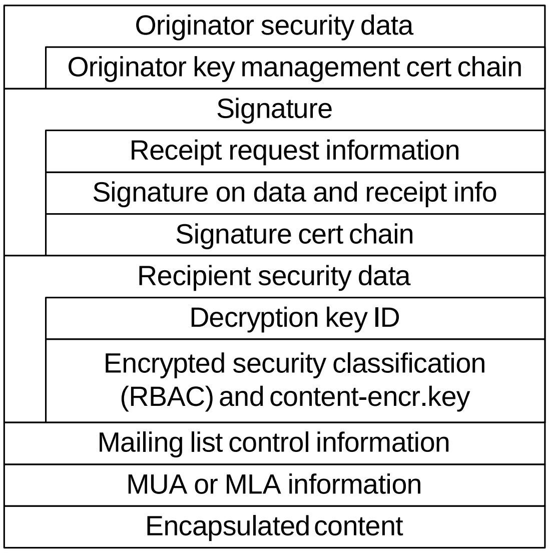
# MSP Protection Types (ctd)

MSP signature covers MSP header and encapsulated content

- Mandatory for mailing lists

User signature covers encapsulated content and receipt request information

# MSP Message Format



- Originator security data
  - Originator key management cert chain
- Signature
  - Receipt request information
  - Signature on data and receipt info
  - Signature cert chain
- Recipient security data
  - Decryption key ID
  - Encrypted security classification (RBAC) and content-encr.key
- Mailing list control information
- MUA or MLA information
- Encapsulated content

## MSP Message Format (ctd)

Extremely complex format

- Many optional features in S/MIME are part of base MSP
- Conversely, looking at MSP explains some of the weird stuff found in S/MIME

Awkward-to-process format

- One-pass processing is impossible
- Signature precedes signed data
- Signing certificates are present after the signature

Fits well with the rest of X.400

## MSP in Practice

MSP is heavily tied into US DoD crypto hardware, e.g. Fortezza:

- DSA signatures
- KEA key management
- Skipjack encryption

MSP was later kludged to work with MIME a la MOSS and PGP/MIME

## Opportunistic email Encryption

After 10-15 years of effort, S/MIME and PGP use is lost in the noise floor (MSP is lost in space)

- Most mail clients include S/MIME support
- Many (OSS) clients include PGP support
- Usage is virtually nonexistent
  - Too hard to use
  - Too much bother to use

## Opportunistic email Encryption (ctd)

Encrypt data using keys managed via key continuity

- Completely transparent to end users
- Requires no extra effort to use
- Effectively free (except for the slight CPU overhead)

Most commonly encountered in SMTP/POP3/IMAP

- Protects mail in transit
- Authenticates the sender
- Prevents unauthorised relaying/spamming

# STARTTLS/STLS/AUTH TLS

Opportunistic encryption for SMTP/POP/IMAP/FTP

```
220 mail.foo.com ESMTP server ready
EHLO server.bar.com
250-STARTTLS
STARTTLS
220 Ready to start TLS
<encrypted transfer>
```

- Upgrades the unprotected link to a TLS-protected one
- Totally transparent, (almost) idiot-proof, etc

# STARTTLS/STLS/AUTH TLS (ctd)

A year after appearing, STARTTLS was protecting more email than all other email encryption protocols combined, despite their 10-15 year lead

- Just as SSH has displaced telnet, so STARTTLS is displacing (or augmenting) straight SMTP/POP3/IMAP
- Auckland Uni turned off unencrypted mail to local servers after STARTTLS appeared, just as they turned off telnet after SSH appeared

Not perfect, but boxes attackers into narrower and narrower channels

Biggest benefit to MTA admins is as an access control mechanism