Session-Level Security

PGP, ssh, S/WAN, satan & crack: Securing the internet by any means necessary

— Don Kitchen

Session-level Security Overview

Most session security protocols use some variation of

1. Decide on security parameters
2. Establish shared secret to protect further communications
3. Authenticate the previous exchange

IPSEC

IP security — security built into the IP layer

Provides host-to-host (or firewall-to-firewall) encryption and authentication

Required for IPv6, optional for IPv4

Comprised of two parts:

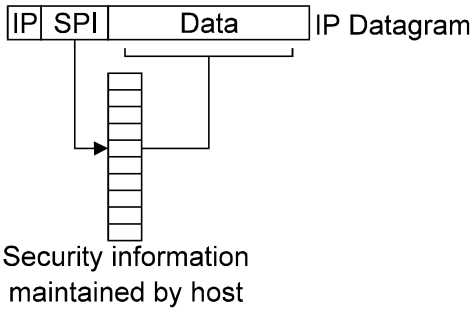
- IPSEC proper (authentication and encryption)
- IPSEC key management

Domain of interpretation (DOI) nails down the precise details for an application of IPSEC

IPSEC Architecture

Key management establishes a security association (SA) for a session

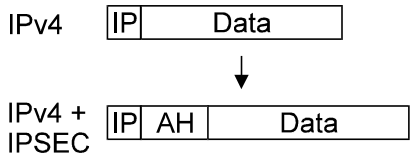
- SA used to provide authentication/confidentiality for that session
- SA is referenced via a security parameter index (SPI) in each IP datagram header



AH

Authentication header — integrity protection only

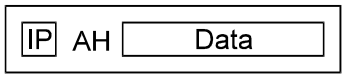
Inserted into IP datagram:



Integrity check value (ICV) is 96-bit HMAC

AH (ctd)

Authenticates entire datagram:



Mutable fields (time-to-live, IP checksums) are zeroed before AH is added

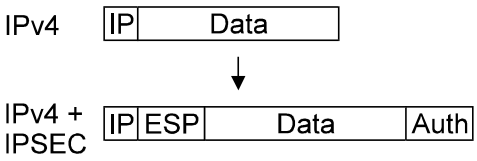
Sequence numbers provide replay protection

- Receiver tracks packets within a 64-entry sliding window

ESP

Encapsulating security protocol — authentication (optional) and confidentiality

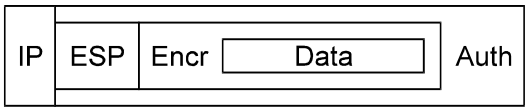
Inserted into IP datagram:



Contains sequence numbers and optional ICV as for AH

ESP (ctd)

Secures data payload in datagram:



Encryption protects payload

- Authentication protects header and encryption

SA bundling is possible

- ESP without authentication inside AH
- Authentication covers more fields this way than just ESP with authentication

IPSEC Algorithms

DES in CBC mode for encryption

HMAC/MD5 and HMAC/SHA (truncated to 96 bits) for authentication

Later versions added optional, DOI-dependent algorithms

- 3DES
- Blowfish
- CAST-128
- IDEA
- RC5
- Triple IDEA (!!!)

Processing

Use SPI to look up security association (SA)

Perform authentication check using SA

Perform decryption of authenticated data using SA

Operates in two modes

- Transport mode (secure IP), protects payload
- Tunneling mode (secure IP inside standard IP), protects entire packet
  - Popular in routers
  - Communicating hosts don't have to implement IPSEC themselves
  - Nested tunneling possible

IPSEC Key Management

ISAKMP

- Internet Security Association and Key Management Protocol

Oakley

- DH-based key management protocol

Photuris

- DH-based key management protocol

SKIP

- Sun's DH-based key management protocol

Protocols changed considerably over time, most borrowed ideas from each other

Photuris

Latin for "firefly", Firefly is the NSA's key exchange protocol for STU-III secure phones

Three-stage protocol

1. Exchange cookies
2. Use DH to establish a shared secret
   Agree on security parameters
3. Identify other party
   Authenticate data exchanged in steps 1 and 2

n. Change session keys or update security parameters

Photuris (ctd)

Cookie based on IP address and port, stops flooding attacks

- Attacker requests many key exchanges and bogs down host (clogging attack)

Cookie depends on

- IP address and port
- Secret known only to host
- Cookie = hash(source and dest IP and port + local secret)

Host can recognise a returned cookie

- Attacker can’t generate fake cookies

Later adopted by other IPSEC key management protocols

Photuris (ctd)

SKIP

Each machine has a public DH value authenticated via

- X.509 certificates
- PGP certificates
- Secure DNS

Public DH value is used as an implicit shared key calculation parameter

- Shared key is used once to exchange encrypted session key
- Session key is used for further encryption/authentication

Clean-room non-US version developed by Sun partner in Moscow

- US government forced Sun to halt further work with non-US version

Oakley

Exchange messages containing any of

- Client/server cookies
- DH information
- Offered/chosen security parameters
- Client/server ID's

until both sides are satisfied

Oakley is extremely open-ended, with many variations possible

- Exact details of messages exchange depends on exchange requirements
  - – Speed vs thoroughness
  - – Identification vs anonymity
  - – New session establishment vs rekey
  - – DH exchange vs shared secrets vs PKC-based exchange

ISAKMP

NSA-designed protocol to exchange security parameters (but not establish keys)

- Protocol to establish, modify, and delete IPSEC security associations
- Provides a general framework for exchanging cookies, security parameters, and key management and identification information
- Exact details left to other protocols

Two phases

1. Establish secure, authenticated channel (“SA”)
2. Negotiate security parameters (“KMP”)

ISAKMP/Oakley

ISAKMP merged with Oakley

- ISAKMP provides the protocol framework
- Oakley provides the security mechanisms

Combined version clarifies both protocols, resolves ambiguities

ISAKMP/Oakley (ctd)

Phase 1 example

Client		Server
Client cookie Client ID Key exchange information	→	
	←	Server cookie Server ID Key exchange information Server signature
Client signature	→	

Other variants possible (data spread over more messages, authentication via shared secrets)

- Above example is aggressive exchange which minimises the number of messages

ISAKMP/Oakley (ctd)

Phase 2 example

Client		Server
Encrypted, MAC'd Client nonce Security parameters offered	→	
	←	Encrypted, MAC'd Server nonce Security parameters accepted
Encrypted, MAC'd Client nonce Server nonce	→	

SSL

Secure sockets layer — TCP/IP socket encryption

Usually authenticates server using digital signature

Can authenticate client, but this is never used

Confidentiality protection via encryption

Integrity protection via MAC's

Provides end-to-end protection of communications sessions

History

SSLv1 designed by Netscape, broken by members of the audience while it was being presented

SSLv2 shipped with Navigator 1.0

Microsoft proposed PCT: PCT != SSL

SSLv3 was peer-reviewed, proposed for IETF standardisation

- Never finalised, still exists only as a draft

SSL Handshake

1. Negotiate the cipher suite
2. Establish a shared session key
3. Authenticate the server (optional)
4. Authenticate the client (optional)
5. Authenticate previously exhanged data

SSL Handshake (ctd)

SSL Handshake (ctd)

Client hello:

- Client nonce
- Available cipher suites (eg RSA + RC4/40 + MD5)

Server hello:

- Server nonce
- Selected cipher suite

Server adapts to client capabilities

Optional certificate exchange to authenticate server/client

- In practice only server authentication is used

SSL Handshake (ctd)

Client key exchange:

- RSA-encrypt(premaster secret)

Both sides:

- 48-byte master secret = hash(premaster + client-nonce + server-nonce)

Client/server change cipher spec:

- Switch to selected cipher suite and key

SSL Handshake (ctd)

Client/server finished

- MAC of previously exchanged parameters (authenticates data from Hello and other exchanges)
  - Uses an early version of HMAC

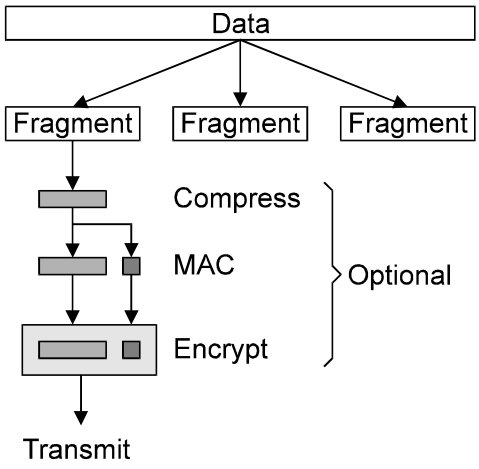
Can reuse previous session data via session ID's in Hello

Can bootstrap weak crypto from strong crypto:

- Server has > 512 bit certificate
- Generates 512-bit temporary key
- Signs temporary key with > 512 bit certificate
- Uses temporary key for security

Maintains separate send and receive states

SSL Data Transfer



SSL Characteristics

Protects the session only

Designed for multiple protocols (HTTP, SMTP, NNTP, POP3, FTP) but only really used with HTTP

Compute-intensive:

- 3 CPU seconds on Sparc 10 with 1Kbit RSA key
- 200 MHz NT box allows about a dozen concurrent SSL handshakes
  - Use multiple servers
  - Use hardware SSL accelerators

Crippled crypto predominates

- Strong servers freely available (Apache), but most browsers US-sourced and crippled

Strong SSL Encryption

Most implementations based on SSLeay,
`http://www.ssleay.org/`

Server

- Some variation of Apache + SSLeay

Browser

- Hacked US browser
- Non-US browser

SSL Proxy

- Strong encryption tunnel using SSL

Strong SSL Browsers

Fortify, `http://www.fortify.net/`

Patches Netscape (any version) to do strong encryption

Original:

```
POLICY-BEGINS-HERE:                 Export policy
Software-Version:                   Mozilla/4.0
MAX-GEN-KEY-BITS:                   512
PKCS12-DES-EDE3:                    false
PKCS12-RC2-128:                     false
PKCS12-RC4-128:                     false
PKCS12-DES-56:                      false
PKCS12-RC2-40:                      true
PKCS12-RC4-40:                      true
…
SSL3-RSA-WITH-RC4-128-MD5:          conditional
SSL3-RSA-WITH-3DES-EDE-CBC-SHA:     conditional
…
```

Strong SSL Browsers (ctd)

Patched version

```
POLICY-BEGINS-HERE:                 Cypherpunk policy
Software-Version:                   Mozilla/4.0
MAX-GEN-KEY-BITS:                   1024
PKCS12-DES-EDE3:                    true
PKCS12-RC2-128:                     true
PKCS12-RC4-128:                     true
PKCS12-DES-56:                      true
PKCS12-RC2-40:                      true
PKCS12-RC4-40:                      true
…
SSL3-RSA-WITH-RC4-128-MD5:          true
SSL3-RSA-WITH-3DES-EDE-CBC-SHA:     true
…
```

Strong SSL Browsers (ctd)

Opera, `http://www.operasoftware.com/`

- Norwegian browser, uses SSLeay

Cryptozilla, `http://www.cryptozilla.org/`

- Based on open-source Netscape
- Strong crypto added within one day of release from the US

Exported US-only versions,
`ftp://ftp.replay.com/pub/replay/pub/`

- Contains copies of most non-exportable software

Strong SSL Servers

Based on SSLeay + some variant of Apache

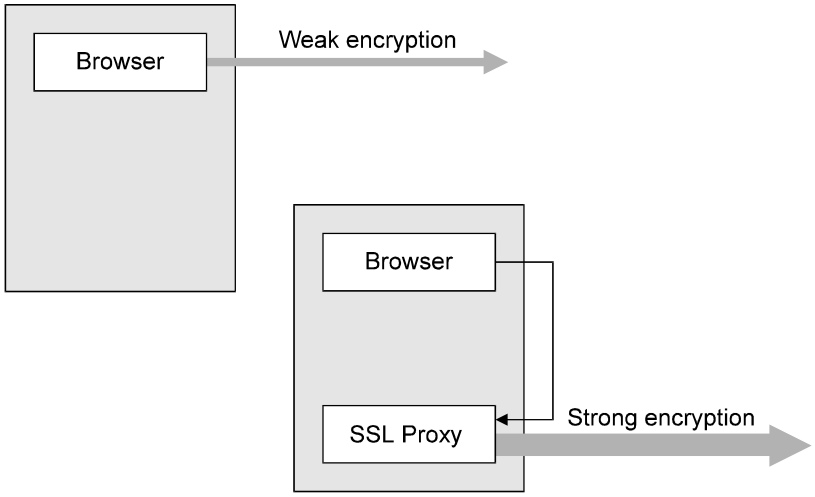
Mostly Unix-only, some NT ports in progress

SSL portion is somewhat painful to configure

Howtos available on the net

Strong SSL Proxies

Tunnel weak or no SSL over strong SSL



SGC

Server Gated Cryptography

Allows strong encryption on a per-server basis

Originally available only to “qualified financial institutions”, later extended slightly (hospitals, some government departents)

Requires special SGC server certificate from Verisign

Enables strong encryption for one server (www.bank.com)

SGC (ctd)

Exportable SSL

Client		Server
Hello	→	
	←	Hello + certificate
Weak encryption key	→	
Weak encryption	↔	Weak encryption

SSL with SGC

Client		Server
Hello	→	
	←	Hello + SGC certificate
Strong encryption key	→	
Strong encryption	↔	Strong encryption

TLS

Transport layer security

IETF-standardised evolution of SSLv3

- Non-patented technology
- Non-crippled crypto
- Updated for newer algorithms

Substantially similar to SSL

- TLS identifies itself as SSL 3.1

Not finalised yet, little implementation support

TLS standards work,
`http://www.consensus.com/ietf-tls/`

S-HTTP

Designed by Terisa in response to CommerceNet RFP,
`http://www.terisa.com/shttp/intro.html`

Predates SSL and S/MIME

Security extension for HTTP (and only HTTP)

Document-based:

- (Pre-)signed documents
- Encrypted documents

Large range of algorithms and formats supported

Not supported by browsers (or much else)

SSH

Originally developed in 1995 as a secure replacement for rsh, rlogin, et al (ssh = secure shell),
`http://www.cs.hut.fi/ssh/`

Also allows port forwarding (tunneling over SSH)

Built-in support for proxies/firewalls

Includes Zip-style compression

Originally implemented in Finland, available worldwide

SSH v2 submitted to IETF for standardisation

Can be up and running in minutes

SSH Protocol

Server uses two keys:

- Long-term server identification key
- Short-term encryption key, changed every hour

Long-term server key binds the connection to the server

Short-term encryption key makes later recovery impossible

- Short-term keys regenerated as a background task

SSH Authentication

Multiple authentication mechanisms

- Straight passwords (protected by SSH encryption)
- RSA-based authentication (client decrypts challenge from server, returns hash to server)
- Plug-in authentication mechanisms, eg SecurID

Developed outside US, crippled crypto not even considered:

- 1024 bit RSA long-term key
- 768 bit RSA short-term key (has to fit inside long-term key for double encryption)
- Triple DES session encryption (other ciphers available)

DNSSEC

DNS name space is divided into zones, each zone has resource records (RR's)

```
Owner_name Type Class TTL Rdlength Rdata
```

- Owner = name of node
- Type = RR type
  - A = Host address
  - NS = authoritative name server
  - CNAME = canonical name for alias
  - SOA = start of zone authority
  - PTR = domain name pointer
  - MX = mail exchange
- Class = IN (Internet)
- TTL = time for which RR may be cached

DNSSEC (ctd)

Name servers hold zone information

- Each zone has primary and secondary servers
- Secondaries perform zone transfers to obtain new data from primaries

Resolvers extract information from name servers

- Cached entry is returned directly
- Interative query returns referral to the appropriate server
- Recursive query queries other server and returns result

All of these points present security vulnerabilities

DNSSEC (ctd)

DNSSEC splits the service into name server and zone manager

- Zone manager signs zone data
- Name server publishes signed data
  - Compromise of name server doesn't compromise DNSSEC

Resolvers need to store at least one top-level zone key

DNSSEC (ctd)

RR's are extended with new types

- KEY, server public key
- SIG, signature on RR
- NXT, chains from one name in a zone to the next
  - Allows authenticated denial of the existence of a name
- These RR's have signature start and end times, require coordinated clocks on hosts

DNSSEC (ctd)

Transaction signature guarantees the response came from a given server

- Signature covers query and response

Also used for

- Secure zone transfer
- Secure dynamic update (replaces editing the zone's master file)
- Offline update
  - Uses authorising dynamic update key for update
  - Zone data is signed later with the zone key

SNMP Security

General SNMP security model: Block it at the router

Authentication: hash(secret value + data)

Confidentiality: encrypt(data + hash)

Many devices are too limited to handle the security themselves

- Handled for them by an element manager
- Device talks to element manager via a single shared key

Users generally use a centralised enterprise manager to talk to element managers

- Enterprise manager is to users what element manager is to devices

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 wrapper/truncated
- Message headers mutate considerably in transit

Data formats have to be carefully designed to avoid problems

Email Security Requirements

Main requirements

- Confidentiality
- Authentication
- Integrity

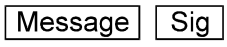
Other requirements

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

Many of these are difficult or impossible to achieve

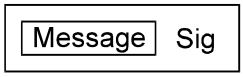
Security Mechanisms

Detached signature:



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

Signed message

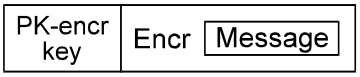


- Signature is always included with the data

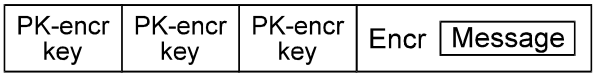
Security Mechanisms (ctd)

Encrypted message

Usually implemented using public-key encryption



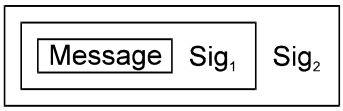
Mailing lists use one public-key encrypted header per recipient



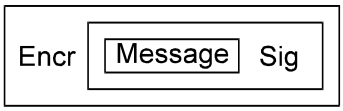
- Any of the corresponding private keys can decrypt the session key and therefore the message

Security Mechanisms (ctd)

Countersigned data



Encrypted and signed data



- 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 2.x, 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:

```
-----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

```
-----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

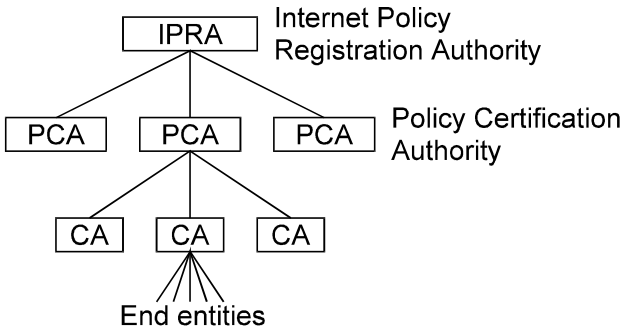
PEM Protection Types (ctd)

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



Hierarchy allows only a single path from the root to the end entity (no cross-certificates)

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, noone 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 noone 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

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
- The reqired X.500 support infrastructure never materialised
- 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 has been 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 have an RSA license

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

- Elgamal referred to in documentation as Diffie-Hellman for no known reason

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

US government was pretty serious, eg:

- 26 February 1995: San Francisco Examiner and SF Chronicle publish an article criticising the governments stand on encryption and the PGP investigation
- 27 February 1995: Author of article subpoena'd to appear before the Grand Jury

Investigation dropped in January 1996 with no charges laid

PGP Message Formats

Unsecured

Compressed

Signed/clearsigned

Encrypted

+ optional encoding

General format

```
-----BEGIN PGP message type-----
data
-----END PGP message type-----
```

PGP Message Formats (ctd)

Clearsigned message:

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

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

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

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

PGP Message Formats (ctd)

Anything else

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

hQEMAlkhsM216BqRAQf/f938A6hglX51/hwa42oCdrQDRGw6HJd+5OqQX/58JB8Y
UAlrYBHYZ5md46ety62phvbwfsNuF9igSx2943CHrnuIVtkSXZRpKogtSE1oMfab
5ivD4I+h3Xk0Jpkn5SXYAzC6/cjAZAZSJjoqy28LBIwzlfNNqrzIuEW8lbLPWAt1
eqdS18ukiOUvnQAI1QfJipGUG+Db1KnpqJP7wHUl/4RG1Qi50p3BCDIspC8jzQ/y
GsKFlckA132dMx6b80vsUZga/tmJOwrgBjSbnOJ8UzLrNe+GjFRyBS+qGuKgLd9M
ymYgMyNOqo/LXALSlLIcz3inDSC5NJj04RbRZ00w4KYAAFrxX9a1BQq1nb40/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

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 user)
- userID trust = how much the userID is trusted to belong to this key
- Signing trust = copy of the signing keys 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

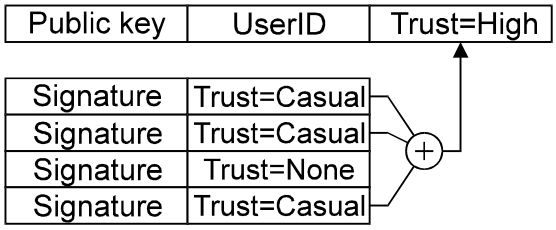
- UserID trust = High
- Key trust = Low

Trust levels

- Unknown
- None
- Casual
- Heavy-duty

PGP Trust (ctd)

Trust levels are automatically computed by PGP



User can define the required trust levels (eg 3 casuals = 1 high)

PGP Trust (ctd)

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

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

Each key can contain multiple userID's with their own trust levels

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

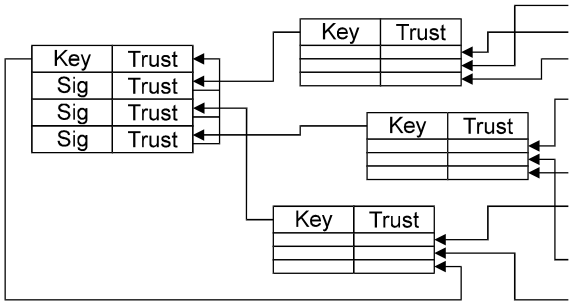
Keys are revoked with a signed revocation which PGP adds to the key

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 owners 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
- Mailed floppies

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

- PGP key fingerprint designed for this purpose

First-generation keyservers

- email/HTTP interface to PGP keyring

Second-generation keyservers

- LDAP kludged to handle PGP ID's

PGP Key Problems

KeyID is 64 least significant bits of public key

- Can construct keys with arbitrary ID's
- Allows signature spoofing

Key fingerprints can also be spoofed

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

Certificate distribution can be manual or automatic (just include it with the message)

PGP is unique among email security protocols in having no crippled encryption

PGP's compression speeds up encryption and signing, reduces message overhead

MIME-based Security

Multipurpose Internet Mail Extensions

Provides a convenient mechanism for transferring composite data

Security-related information sent as sections of a multipart message

- multipart/signed
- multipart/encrypted

Binary data handled via base64 encoding

MIME-aware mailers can automatically process the security informtion (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 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 and MIME

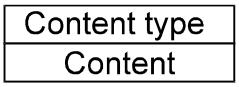
- PKCS, Public Key Cryptography Standards
  - RC2, RC4 for data encryption
  - PKCS #1, RSA encryption, for key exchange
  - PKCS #7, cryptographic message syntax, for message formatting

Newer versions added non-proprietary and non-patented ciphers

CMS

Cryptographic Message Syntax

- Type-and-value format



Data content types

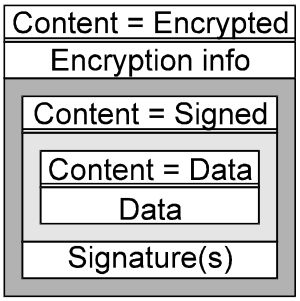
- Data
- Signed data
- Encrypted data (conventional encryption)
- Enveloped data (PKC-encrypted)
- Digested (hashed) data
- Authenticated (MAC'd) data

CMS (ctd)

Other content types possible

- Private keys
- Key management messages

Content can be arbitrarily nested



Signed Data Format

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

Signature Format

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 signature on content rather than content itself, so other content doesn't have to be present
  - Countersignatures can contain further countersignatures

Enveloped Data Format

Newer versions add support for key agreement algorithms and previously distributed shared conventional keys

CMS → S/MIME

Wrap each individual CMS layer in MIME

base64 encode + wrap content
Encode as CMS data
base64 encode + wrap content
Encode as CMS signed data
base64 encode + wrap content
Encode as CMS enveloped data
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 security downgrade
- S/MIME-cracking screen saver released in 1997, `http://www.counterpane.com/smime.html`
  - Performs optimised attack using RC2 key setup cycles
  - Looks for MIME header in decrypted data

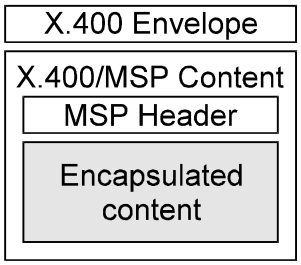
Original S/MIME based on patented RSA and proprietary RC2, rejected by IETF as a standard

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

MSP

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

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



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

- MSP later used with MIME

MSP Services

Services provided

- Authentication
- Integrity
- Confidentiality
- Non-repudiation of origin (via message signature)
- 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 Certificates

MSP defines three X.509 certificate types

- Signature-only
- Encryption (key management) only
- Signature and encryption (two keys in one certificate)

Certificate also includes RBAC 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, RBAC

- Encrypted with key management or dual-key certificate

Non-repudiation + confidentiality, integrity, RBAC

- 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 Encrypted RBAC information (additional to per-recipient RBAC info)
Signature Receipt request information Signature on encapsulated data and receipt info Signature cert chain
Recipient security data Per-recipient Key management certificate identifier (KMID) Encrypted security classification(s) (RBAC) + secret key
Mailing list control information
MUA or MLA Signature
Encapsulated content

- RBAC is encrypted to protect it if no signatures are used

MSP in Practice

MSP is heavily tied into Fortezza hardware

- DSA signatures
- KEA key management
- Skipjack encryption

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