Cryptography and Data Security

Peter Gutmann

University of Auckland

http://www.cs.auckland.ac.nz/~pgut001

Security Requirements

Confidentiality

- Protection from disclosure to unauthorised persons

Integrity

- Maintaining data consistency

Authentication

- Assurance of identity of person or originator of data

Non-repudiation

- Originator of communications can’t deny it later

Security Requirements (ctd)

Availability

- Legitimate users have access when they need it

Access control

- Unauthorised users are kept out

These are often combined

- User authentication used for access control purposes
- Non-repudiation combined with authentication

Security Threats

Information disclosure/information leakage

Integrity violation

Masquerading

Denial of service

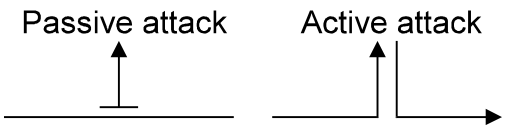
Illegitimate use

Generic threat: Backdoors, trojan horses, insider attacks

Most Internet security problems are access control or authentication ones

- Denial of service is also popular, but mostly an annoyance

Attack Types



Passive attack can only observe communications or data

Active attack can actively modify communications or data

- Often difficult to perform, but very powerful
  - Mail forgery/modification
  - TCP/IP spoofing/session hijacking

Security Services

From the OSI definition:

- Access control: Protects against unauthorised use
- Authentication: Provides assurance of someone's identity
- Confidentiality: Protects against disclosure to unauthorised identities
- Integrity: Protects from unauthorised data alteration
- Non-repudiation: Protects against originator of communications later denying it

Security Mechanisms

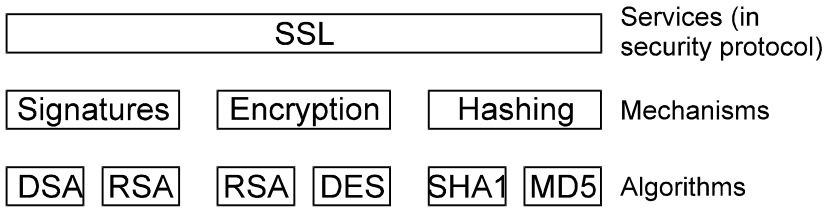
Three basic building blocks are used:

- Encryption is used to provide confidentiality, can provide authentication and integrity protection
- Digital signatures are used to provide authentication, integrity protection, and non-repudiation
- Checksums/hash algorithms are used to provide integrity protection, can provide authentication

One or more security mechanisms are combined to provide a security service

Services, Mechanisms, Algorithms

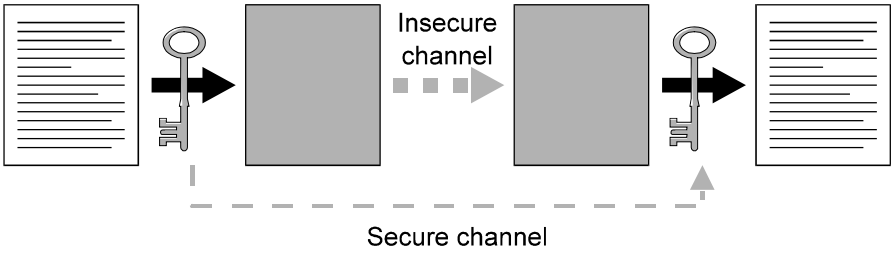
A typical security protocol provides one or more services



- Services are built from mechanisms
- Mechanisms are implemented using algorithms

Conventional Encryption

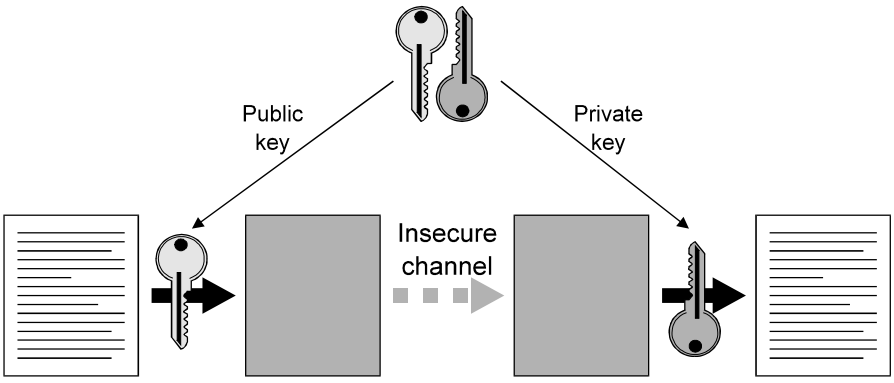
Uses a shared key



Problem of communicating a large message in secret reduced to communicating a small key in secret

Public-key Encryption

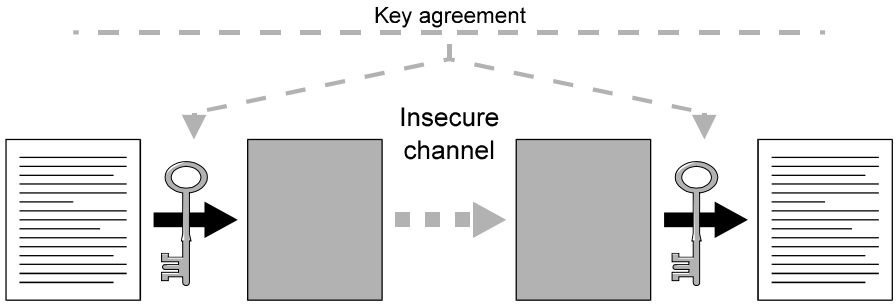
Uses matched public/private key pairs



Anyone can encrypt with the public key, only one person can decrypt with the private key

Key Agreement

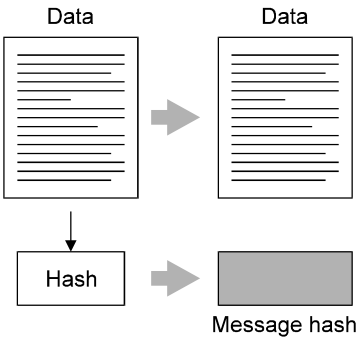
Allows two parties to agree on a shared key



Provides part of the required secure channel for exchanging a conventional encryption key

Hash Functions

Creates a unique “fingerprint” for a message

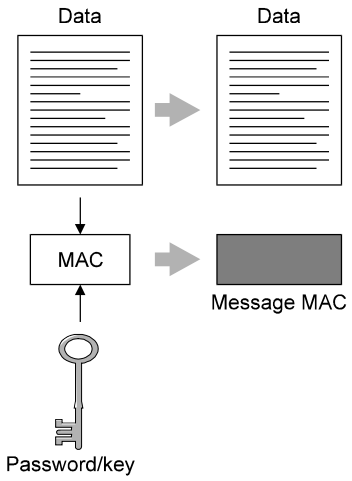


Anyone can alter the data and calculate a new hash value

- Hash has to be protected in some way

MAC's

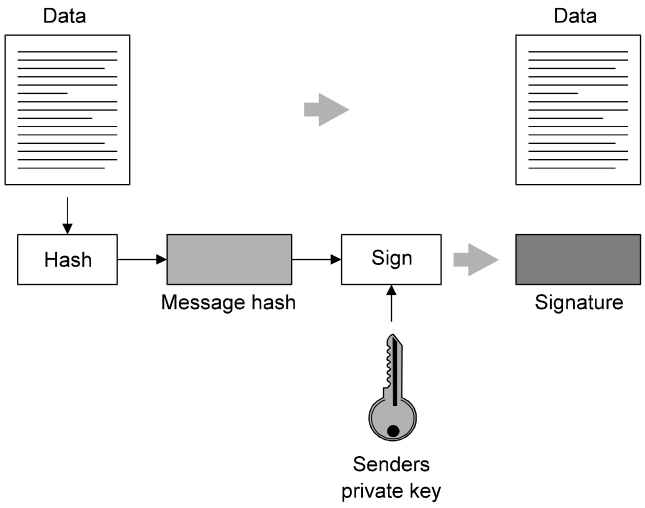
Message Authentication Code, adds a password/key to a hash



Only the password holder(s) can generate the MAC

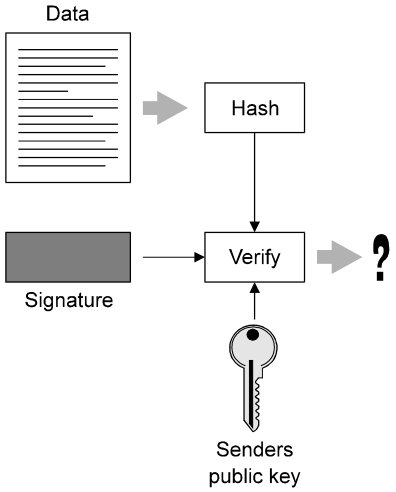
Digital Signatures

Combines a hash with a digital signature algorithm



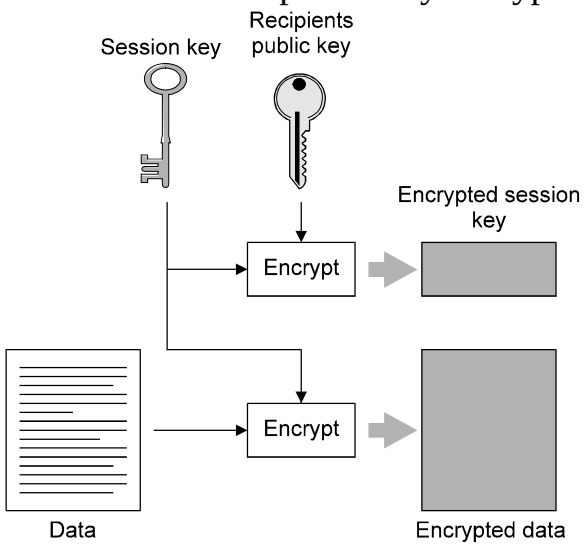
Digital Signatures (ctd)

Signature checking:

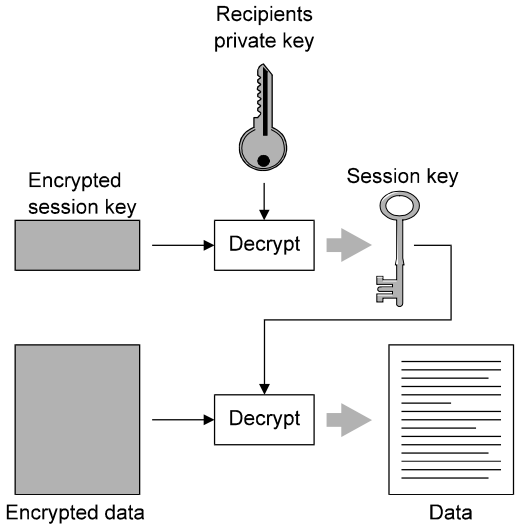


Message/Data Encryption

Combines conventional and public-key encryption

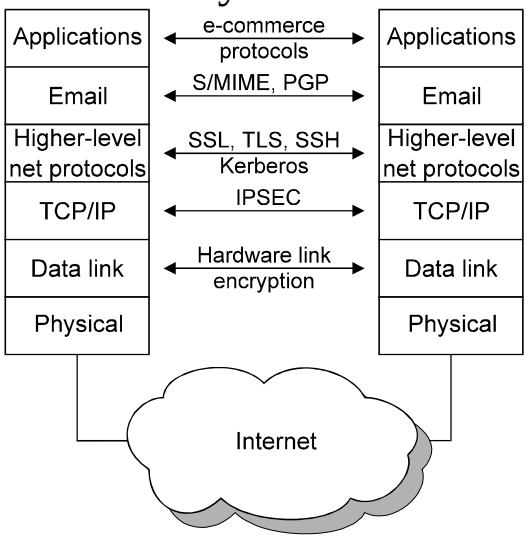


Message/data Encryption (ctd)



Public-key encryption provides a secure channel to exchange conventional encryption keys

Security Protocol Layers



The further down you go, the more transparent it is
The further up you go, the easier it is to deploy

Encryption and Authentication Algorithms and Technology

Cryptography is nothing more than a mathematical framework for discussing the implications of various paranoid delusions

- Don Alvarez

Historical Ciphers

Nonstandard hieroglyphics, 1900BC

Atbash cipher (Old Testament, reversed Hebrew alphabet, 600BC)

Caesar cipher:

letter = letter + 3

'fish' → 'ilvk'

rot13: Add 13/swap alphabet halves

- Usenet convention used to hide possibly offensive jokes
- Applying it twice restores original text

Substitution Ciphers

Simple substitution cipher:

a = p, b = m, c = f, ...

Break via letter frequency analysis

Polyalphabetic substitution cipher

1. a = p, b = m, c = f, ...
2. a = l, b = t, c = a, ...
3. a = f, b = x, c = p, ...

Break by decomposing into individual alphabets, then solve as simple substitution

One-time Pad (1917)

Message	s	e	c	r	e	t
	18	5	3	17	5	19
OTP	+15	8	1	12	19	5
	7	13	4	3	24	24
	g	m	d	c	x	x

OTP is unbreakable *provided*

- Pad is never reused (VENONA)
- Unpredictable random numbers are used (physical sources, eg radioactive decay)

One-time Pad (ctd)

Used by

- Russian spies
- The Washington-Moscow “hot line”
- CIA covert operations

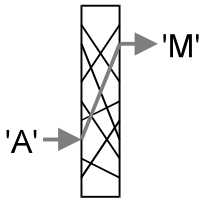
Many snake oil algorithms claim unbreakability by claiming to be a OTP

- Pseudo-OTP’s give pseudo-security

Cipher machines attempted to create approximations to OTP’s, first mechanically, then electronically

Cipher Machines (~1920)

1. Basic component = wired rotor



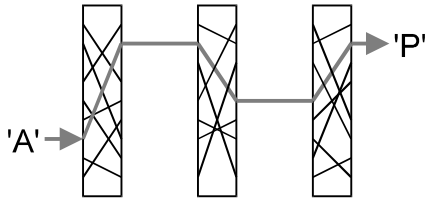
- Simple substitution

2. Step the rotor after each letter

- Polyalphabetic substitution, period = 26

Cipher Machines (ctd)

3. Chain multiple rotors



Each steps the next one when a full turn is complete

Cipher Machines (ctd)

Two rotors, period = 26×26

$= 676$

Three rotors, period = $26 \times 26 \times 26$

$= 17{,}576$

Rotor sizes are chosen to be relatively prime to give maximum-length sequence

Key = rotor wiring

= rotor start position

Cipher Machines (ctd)

Famous rotor machines

- US: Converter M-209
- UK: TYPEX
- Japan: Red, Purple
- Germany: Enigma

Many books on Enigma

- Kahn, Siezing the Enigma
- Levin, Ultra Goes to War
- Welchman, The Hut Six Story
- Winterbothm, The Ultra Secret

“It would have been secure if used properly”

Use of predictable openings:

- “Mein Fuehrer! ...”
- “Nothing to report”

Use of the same key over an extended period

Encryption of the same message with old (compromised) and new keys

Device treated as a magic black box, a mistake still made today

Inventors believed it was infallible, " " " " "

Cipher Machines (ctd)

Various kludges made to try to improve security — none worked

Enigmas were sold to friendly nations after the war

Improved rotor machines were used into the 70's and 80's

Further reading:

Kahn, The Codebreakers

Cryptologia, quarterly journal

Stream Ciphers

Binary pad (keystream), use XOR instead of addition

Plaintext = original, unencrypted data
Ciphertext = encrypted data

Plaintext		1	0	0	1	0	1	1
Keystream	XOR	0	1	0	1	1	0	1
Ciphertext		1	1	0	0	1	1	0
Keystream	XOR	0	1	0	1	1	0	1
Plaintext		1	0	0	1	0	1	1

Two XOR's with the same data always cancel out

Stream Ciphers (ctd)

Using the keystream and ciphertext, we can recover the plaintext

but

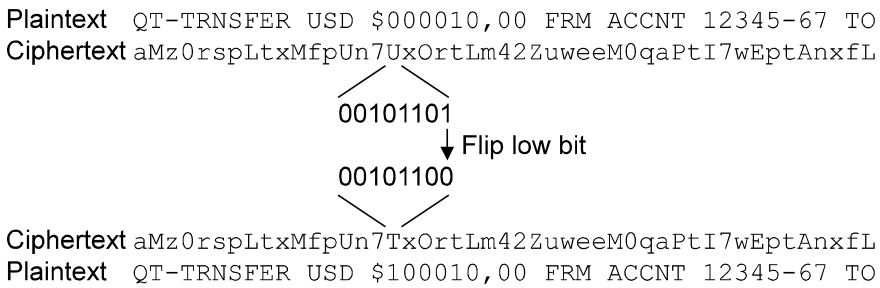
Using the plaintext and ciphertext, we can recover the keystream

Using two ciphertexts from the same keystream, we can recover the XOR of the plaintexts

- Any two components of an XOR-based encryption will recover the third
- Never reuse a key with a stream cipher
- Better still, never use a stream cipher

Stream Ciphers (ctd)

Vulnerable to bit-flipping attacks



RC4

Stream cipher optimised for fast software implementation

2048-bit key, 8-bit output

Former trade secret of RSADSI, reverse-engineered and posted to the net in 1994

```
while( length-- )
  {
  x++; sx = state[ x ]; y += sx;
  sy = state[ y ]; state[ y ] = sx; state[ x ] = sy;
  *data++ ^= state[ ( sx+sy ) & 0xFF ];
  }
```

Takes about a minute to implement from memory

RC4 (ctd)

Extremely fast

Used in SSL (Netscape, MSIE), Lotus Notes, Windows password encryption, MS Access, Adobe Acrobat, MS PPTP, Oracle Secure SQL, ...

Usually used in a manner which allows the keystream to be recovered (Windows password encryption, early Netscape server key encryption, some MS server/browser key encryption, MS PPTP, Access, ...)

Illustrates the problem of treating a cipher as a magic black box

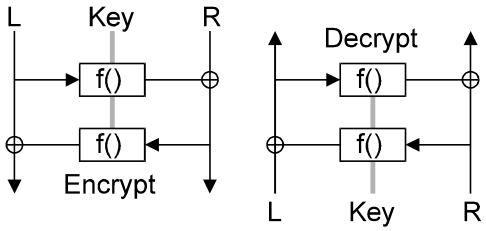
Recommendation: Avoid this, it's too easy to get wrong

Block Ciphers

Originated with early 1970's IBM effort to develop banking security systems

First result was Lucifer, most common variant has 128-bit key and block size

- It wasn't secure in any of its variants



Called a Feistel or product cipher

Block Ciphers (ctd)

f()-function is a simple transformation, doesn't have to be reversible

Each step is called a round; the more rounds, the greater the security (to a point)

Most famous example of this design is DES:

- 16 rounds
- 56 bit key
- 64 bit block size (L,R = 32 bits)

Designed by IBM with, uh, advice from the NSA

Attacking Feistel Ciphers

Differential cryptanalysis

- Looks for correlations in f()-function input and output

Linear cryptanalysis

- Looks for correlations between key and cipher input and output

Related-key cryptanalysis

- Looks for correlations between key changes and cipher input/output

Differential cryptanalysis discovered in 1990; virtually all block ciphers from before that time are vulnerable...

...except DES. IBM (and the NSA) knew about it 15 years earlier

Strength of DES

Key size = 56 bits

Brute force = 2^{55} attempts

Differential cryptanalysis = 2^{47} attempts

Linear cryptanalysis = 2^{43} attempts

(but the last two are impractical)

> 56 bit keys don't make it any stronger

> 16 rounds don't make it any stronger

DES Key Problems

Key size = 56 bits

= 8 × 7-bit ASCII chars

Alphanumeric-only password converted to uppercase

= 8 × ~5-bit chars

= 40 bits

DES uses low bit in each byte for parity

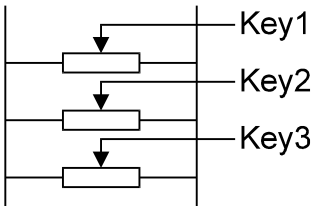
= 32 bits

- Forgetting about the parity bits is so common that the NSA probably designs its keysearch machines to accomodate this

Breaking DES

DES was designed for efficiency in early-70's hardware

Makes it easy to build pipelined brute-force breakers in late-90's hardware



16 stages, tests 1 key per clock cycle

Breaking DES (ctd)

Can build a DES-breaker using

- Field-programmable gate array (FPGA), software-programmable hardware
- Application-specific IC (ASIC)

100 MHz ASIC = 100M keys per second per chip

Chips = $10 in 5K+ quantities

$50,000 = 500 billion keys/sec

= 20 hours/key (40-bit DES takes 1 second)

Breaking DES (ctd)

$1M = 1 hour per key ($^1/_{20}$ sec for 40 bits)

$10M = 6 minutes per key ($^1/_{200}$ sec for 40 bits)

(US black budget is ~$25-30 billion)

(distributed.net = ~70 billion keys/sec with 20,000 computers)

EFF (US non-profit organisation) broke DES in 2½ days

Amortised cost over 3 years = 8 cents per key

- If your secret is worth more than 8 cents, don't encrypt it with DES

September 1998: German court rules DES "out of date and unsafe" for financial applications

Brute-force Encryption Breaking

Type of Attacker	Budget	Tool	Time and cost per key recovered		Keylen (bits) for security	
			40 bits	56 bits	1995	2015
Pedestrian hacker	Tiny	PC	1 week	Infeasible	45	59
	$400	FPGA	5 hours $0.08	38 years $5,000	50	64
Small business	$10K	FPGA	12 mins $0.08	556 days $5,000	55	69
Corporate department	$300K	FPGA	24 secs $0.08	19 days $5,000	60	74
		ASIC	0.18 secs $0.001	3 hours $38		
Big company	$10M	FPGA	0.7 secs $0.08	13 hours $5,000	70	84
		ASIC	0.005 s $0.001	6 mins $38		
Intelligence agency	$300M	ASIC	0.0002 s $0.001	12 secs $38	75	89

Other Block Ciphers

Triple DES (3DES)

- Encrypt + decrypt + encrypt with 2 (112 bits) or 3 (168 bits) DES keys
- By late 1998, banking auditors were requiring the use of 3DES rather than DES

RC2

- Companion to RC4, 1024 bit key
- RSADSI trade secret, reverse-engineered and posted to the net in 1996
- RC2 and RC4 have special status for US exportability

Other Block Ciphers (ctd)

IDEA

- Developed as PES (proposed encryption standard), adapted to resist differential cryptanalysis as IPES, then IDEA
- Gained popularity via PGP, 128 bit key
- Patented

Blowfish

- Optimised for high-speed execution on 32-bit processors
- 448 bit key, relatively slow key setup

CAST-128

- Used in PGP 5.x, 128 bit key

Other Block Ciphers

Skipjack

- Classified algorithm originally designed for Clipper, declassified in 1998
- 32 rounds, breakable with 31 rounds
- 80 bit key, inadequate for long-term security

GOST

- GOST 28147, Russian answer to DES
- 32 rounds, 256 bit key
- Incompletely specified

Other Block Ciphers

AES

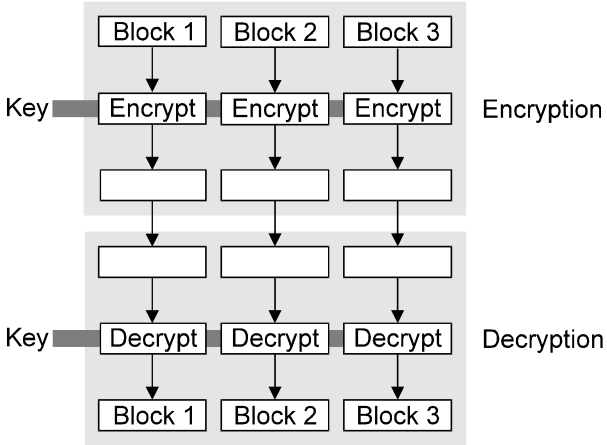
- Advanced Encryption Standard, replacment for DES
- 128 bit block size, 128/192/256 bit key
- Will take several years to be finalised

Many, many others

- No good reason not to use one of the above, proven algorithms

Using Block Ciphers

ECB, Electronic Codebook



Each block encrypted independently

Using Block Ciphers (ctd)

Original text

Deposit	$10,000	in acct.	number	12-3456-	789012-3

Intercepted encrypted form

H2nx/GHE	KgvldSbq	GQHbrUt5	tYf6K7ug	S4CrMTvH	7eMPZcE2

Second intercepted message

H2nx/GHE	KgvldSbq	GQHbrUt5	tYf6K7ug	Pts21LGb	a8oaNWpj

Cut and paste blocks with account information

H2nx/GHE	KgvldSbq	GQHbrUt5	tYf6K7ug	S4CrMTvH	a8oaNWpj

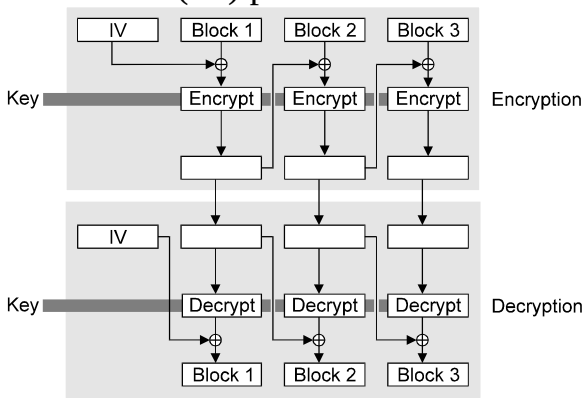
Decrypted message will contain the attackers account — without them knowing the encryption key

Using Block Ciphers (ctd)

Need to

- Chain one block to the next to avoid cut & paste attacks
- Randomise the initial block to disguise repeated messages

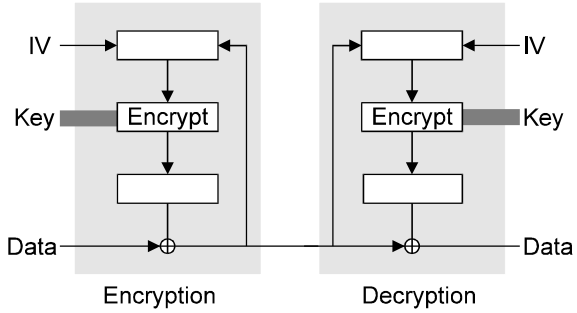
CBC (cipher block chaining) provides chaining, random initialisation vector (IV) provides randomisation



Using Block Ciphers (ctd)

Both ECB and CBC operate on entire blocks

CFB (ciphertext feedback) operates on bytes or bits



This converts a block cipher to a stream cipher (with the accompanying vulnerabilities)

Relative Performance

Fast

RC4
Blowfish, CAST-128, AES
Skipjack
DES, IDEA, RC2
3DES, GOST

Slow

Typical speeds

- RC4 = Tens of MB/second
- 3DES = MB/second

Recommendations

- For performance, use Blowfish
- For job security, use 3DES

Public Key Encryption

How can you use two different keys?

- One is the inverse of the other:
  key1 = 3, key2 = 1/3, message M = 4
  Encryption: Ciphertext C = M × key1
  = 4 × 3
  = 12
  Decryption: Plaintext M = C × key2
  = 12 × 1/3
  = 4

One key is published, one is kept private → public-key cryptography, PKC

Example: RSA

n, e = public key, n = product of two primes p and q

d = private key

Encryption: $C = M^e \bmod n$

Decryption: $M = C^d \bmod n$

p, q = 5, 7

$n = p \times q$
$= 35$

e = 3

$d = e^{-1} \bmod ((p-1)(q-1))$
$= 16$

Example: RSA (ctd)

Message M = 4

Encryption: $C = 4^3 \bmod 35$

$= 29$

Decryption: $M = 29^{16} \bmod 35$

$= \sim 2.5 \times 10^{23} \bmod 35$

$= 4$

(Use mathematical tricks otherwise the numbers get dangerous)

Public-key Algorithms

RSA (Rivest-Shamir-Adleman), 1977

- Digital signatures and encryption in one algorithm
- Private key = sign and decrypt
- Public key = signature check and encrypt
- Patented, expires September 2000

DH (Diffie-Hellman), 1976

- Key exchange algorithm

Elgamal

- DH variant, one algorithm for encryption, one for signatures
- Non-patented alternative to RSA

Public-key Algorithms (ctd)

DSA (Digital Signature Algorithm)

- Elgamal signature variant, designed by the NSA as the US government digital signature standard
- Intended for signatures only, but can be adapted for encryption

All have roughly the same strength:

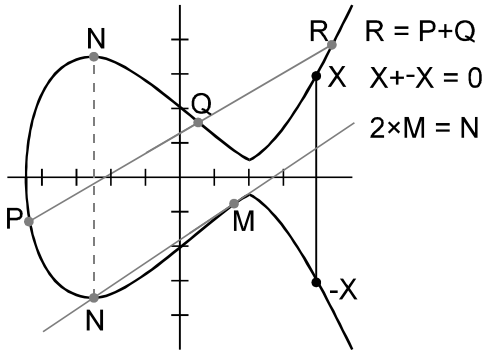
512 bit key is marginal
1024 bit key is recommended minimum size
2048 bit key is better for long-term security

Recommendation

- Anything suitable will do, RSA has wide acceptance but has patent problems in the US

Elliptic Curve Algorithms

Use mathematical trickery to speed up public-key operations



Elliptic Curve Algorithms (ctd)

Now we can add, subtract, etc. So what?

- Calling it "addition" is arbitrary, we can just as easily call it multiplication
- We can now move (some) conventional PKC's over to EC PKC's (DSA → ECDSA)

Now we have a funny way to do PKC's. So what?

- Breaking PKC's over elliptic curve groups is much harder than beaking conventional PKC's
- We can use much shorter keys
- Encryption/decryption is faster since keys are shorter
- Key sizes are much smaller

Advantages/Disadvantages of ECC's

Advantages

- Useful for smart cards because of their low resource requirements
- Useful where high-speed operation is required

Disadvantages

- New, details are still being resolved
- Many techniques are still too new to trust
- ECC's are a minefield of patents, pending patents, and submarine patents

Recommendation: Don't use them unless you really need their special features

Key Sizes and Algorithms

Conventional vs public-key vs ECC key sizes

Conventional	Public-key	ECC
(40 bits)	—	—
56 bits	(400 bits)	—
64 bits	512 bits	—
80 bits	768 bits	—
90 bits	1024 bits	160 bits
112 bits	1792 bits	195 bits
120 bits	2048 bits	210 bits
128 bits	2304 bits	256 bits

(Your mileage may vary)

Key Sizes and Algorithms (ctd)

However

- Conventional key is used once per message
- Public key is used for hundreds or thousands of messages

A public key compromise is much more serious than a conventional key compromise

- Compromised logon password, attacker can
  - Delete your files
- Compromised private key, attacker can
  - Drain credit card
  - Clean out bank account
  - Sign contracts/documents
  - Identity theft

Key Sizes and Algorithms (ctd)

512 bit public key vs 40 bit conventional key is a good balance for weak security

Recommendations for public keys:

- Use 512-bit keys only for micropayments/smart cards
- Use 1K bit key for short-term use (1 year expiry)
- Use 1.5K bit key for longer-term use
- Use 2K bit key for certification authorities (keys become more valuable further up the hierarchy), long-term contract signing, long-term secrets

The same holds for equivalent-level conventional and ECC keys

Hash Algorithms

Reduce variable-length input to fixed-length (128 or 160 bit) output

Requirements

- Can’t deduce input from output
- Can’t generate a given output (CRC fails this requirement)
- Can’t find two inputs which produce the same output (CRC also fails this requirement)

Used to

- Produce fixed-length fingerprint of arbitrary-length data
- Produce data checksums to enable detection of modifications
- Distill passwords down to fixed-length encryption keys

Also called message digests or fingerprints

MAC Algorithms

Hash algorithm + key to make hash value dependant on the key

Most common form is HMAC (hash MAC)

hash(key, hash(key, data))

- Key affects both start and end of hashing process

Naming: hash + key = HMAC-hash

MD5 → HMAC-MD5

Algorithms

MD2: 128-bit output, deprecated

MD4: 128-bit output, broken

MD5: 128-bit output, weaknesses

SHA-1: 160-bit output, NSA-designed US government secure hash algorithm, companion to DSA

RIPEMD-160: 160-bit output

HMAC-MD5: MD5 turned into a MAC

HMAC-SHA: SHA-1 turned into a MAC

Recommendation: Use SHA-1, HMAC-SHA

Key Management and Certificates

By the power vested in me I now declare this text and this bit string 'name' and 'key'. What RSA has joined, let no man put asunder

— Bob Blakley

Key Management

Key management is the hardest part of cryptography

Two classes of keys

- Short-term session keys (sometimes called ephemeral keys)
  - Generated automatically and invisibly
  - Used for one message or session and discarded
- Long-term keys
  - Generated explicitly by the user

Long-term keys are used for two purposes

- Authentication (including access control, integrity, and non-repudiation)
- Confidentiality (encryption)
  - Establish session keys
  - Protect stored data

Key Management Problems

Key certification

Distributing keys

- Obtaining someone else's public key
- Distributing your own public key

Establishing a shared key with another party

- Confidentiality: Is it really known only to the other party?
- Authentication: Is it really shared with the intended party?

Key storage

- Secure storage of keys

Revocation

- Revoking published keys
- Determining whether a published key is still valid

Key Lifetimes and Key Compromise

Authentication keys

- Public keys may have an extremely long lifetime (decades)
- Private keys/conventional keys have shorter lifetimes (a year or two)

Confidentiality keys

- Should have as short a lifetime as possible

If the key is compromised

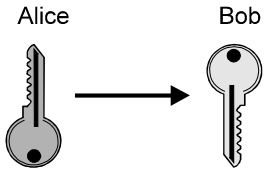
- Revoke the key

Effects of compromise

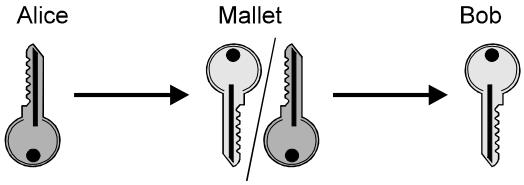
- Authentication: Signed documents are rendered invalid unless timestamped
- Confidentiality: All data encrypted with it is compromised

Key Distribution

Alice retains the private key and sends the public key to Bob



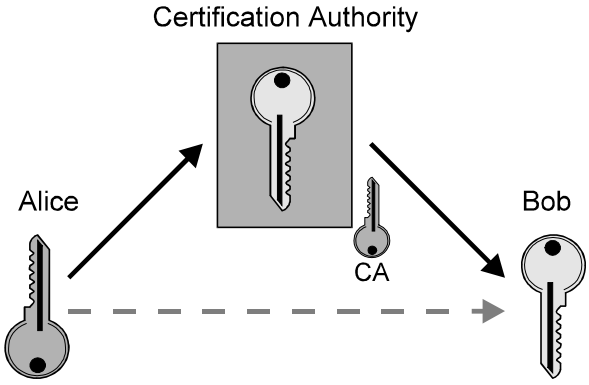
Mallet intercepts the key and substitutes his own key



Mallet can decrypt all traffic and generate fake signed message

Key Distribution (ctd)

A certification authority (CA) solves this problem



CA signs Alice's key to guarantee its authenticity to Bob

- Mallet can't substitute his key since the CA won't sign it

Certification Authorities

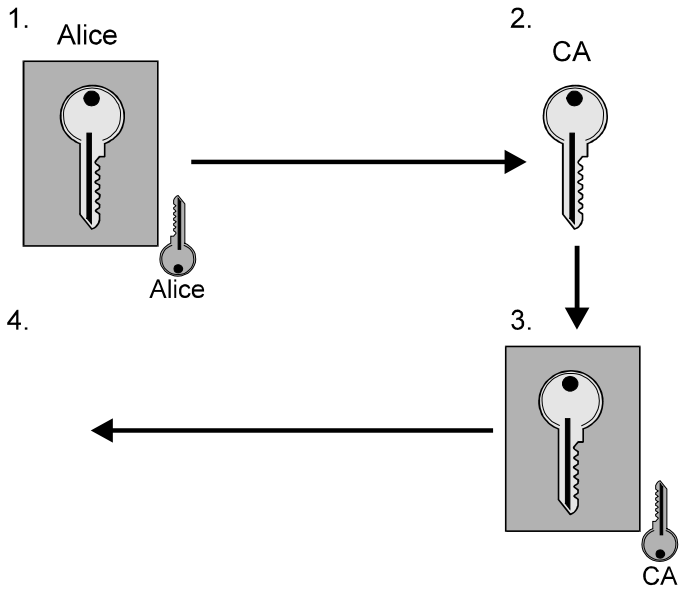
A certification authority (CA) guarantees the connection between a key and an end entity

An end entity is

- A person
- A role (“Director of marketing”)
- An organisation
- A pseudonym
- A piece of hardware or software
- An account (bank or credit card)

Some CA’s only allow a subset of these types

Obtaining a Certificate



Obtaining a Certificate (ctd)

1. Alice generates a key pair and signs the public key and identification information with the private key
   - Proves that Alice holds the private key corresponding to the public key
   - Protects the public key and ID information while in transit to the CA
2. CA verifies Alices signature on the key and ID information

2a. Optional: CA verifies Alices ID through out-of-band means
   - email/phone callback
   - Business/credit bureau records, in-house records

Obtaining a Certificate (ctd)

3. CA signs the public key and ID with the CA key, creating a certificate
   - CA has certified the binding between the key and ID
4. Alice verifies the key, ID, and CA's signature
   - Ensures the CA didn't alter the key or ID
   - Protects the certificate in transit
5. Alice and/or the CA publish the certificate

Role of a CA

Original intent was to certify that a key really did belong to a given party

Role was later expanded to certify all sorts of other things

- Are they a bona fide business?
- Can you trust their web server?
- Can you trust the code they write?
- Is their account in good standing?
- Are they over 18?

When you have a certificate-shaped hammer, everything looks like a nail

Certificate History

Certificates were originally intended to protect access to the X.500 directory

- All-encompassing, global directory run by monopoly telco's

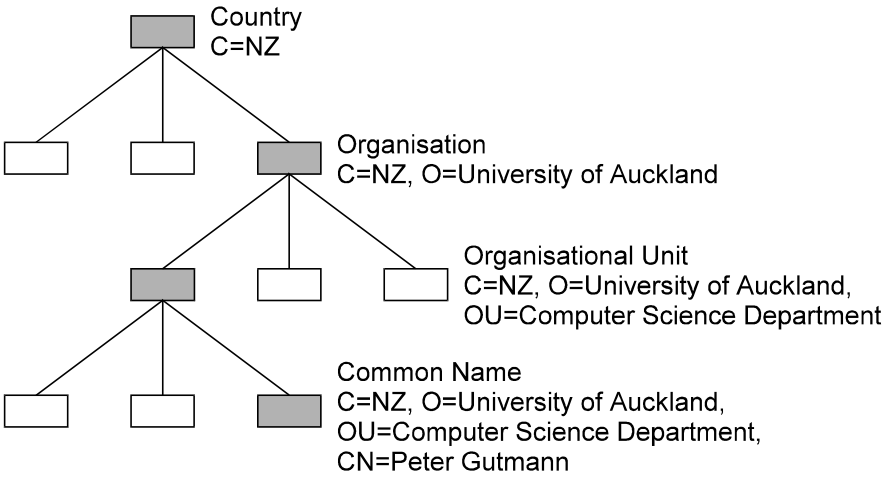
Concerns about misuse of the directory

- Companies don't like making their internal structure public
  - Directory for corporate headhunters
- Privacy concerns
  - Directory of single women
  - Directory of teenage children

X.509 certificates were developed as part of the directory access control mechanisms

X.500 Naming

X.500 introduced the Distinguished Name (DN), a guaranteed unique name for everything on earth



X.500 Naming (ctd)

Typical DN components

- Country C
- State or province SP
- Locality L
- Organisation O
- Organisational unit OU
- Common name CN

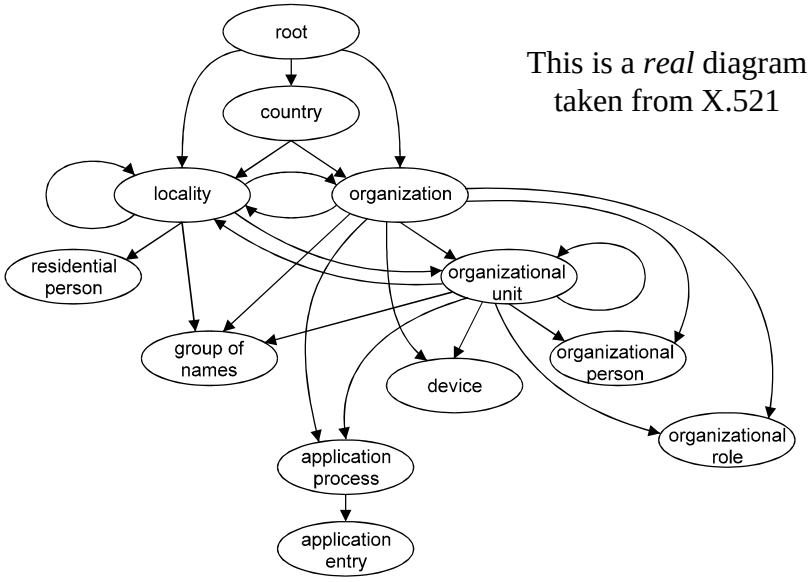
Typical X.500 DN

C=US/L=Area 51/O=Hanger 18/OU=X.500 Standards Designers/CN=John Doe

– When the X.500 revolution comes, your name will be lined up against the wall and shot

Problems with X.500 Names

Noone ever managed to figure out how to make DN's work



This is a *real* diagram taken from X.521

Problems with X.500 Names (ctd)

No clear plan on how to organise the hierarchy

- Attempts were made to define naming schemes, but nothing really worked
- People couldn't even agree on what things like 'localities' were

Hierarchical naming model fits the military and governments, but doesn't work for businesses or individuals

Solving the DN Problem

Two solutions were adopted

1. Users put whatever they felt like into the DN
2. X.509v3 added support for alternative (non-DN) names

General layout for a business-use DN

Country + Organisation + Organisational Unit + Common Name

– C=New Zealand
  O=Dave's Wetaburgers
  OU=Procurement
  CN=Dave Taylor

Solving the DN Problem (ctd)

General layout for a personal-use DN

Country + State or Province + Locality + Common Name

– C=US
  SP=California
  L=San Francisco
  CN=John Doe

There are dozens of other odd things which can be specified

- teletexTerminalIdentifier
- destinationIndicator
- supportedApplicationContext

Luckily these are almost never used

Non-DN Names

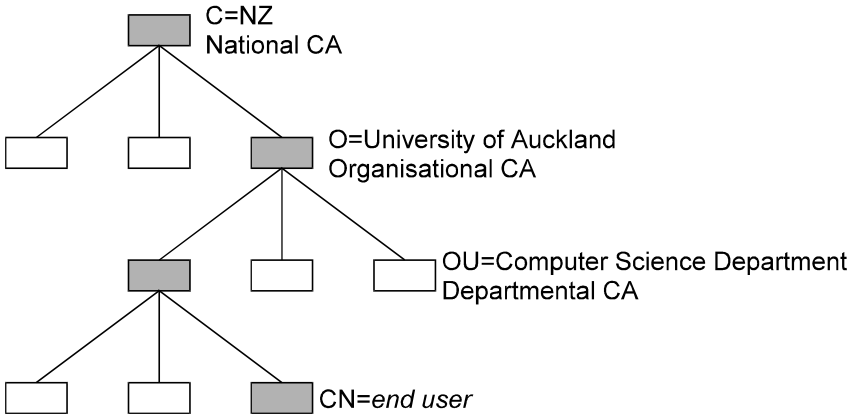
X.509 v3 added support for other name forms

- email addresses
- DNS names
- URL's
- IP addresses
- EDI and X.400 names
- Anything else (type+value pairs)

For historical reasons, email addresses are often stuffed into DN's rather than being specified as actual email addresses

CA Hierarchy in Theory

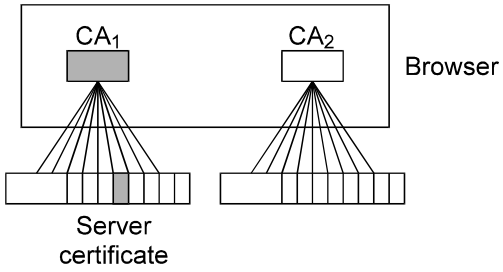
Portions of the X.500 hierarchy have CA's attached to them



Top-level CA is called the root CA, aka "the single point of failure"

CA Hierarchy in Practice

Flat or Clayton's hierarchy



CA certificates are hard-coded into web browsers or email software

- Later software added the ability to add new CA's to the hardcoded initial set

Key Databases/Directories

Today, keys are stored in

- Flat files (one per key)
- Relational databases
- Proprietary databases (Netscape)
- Windows registry (MSIE)

Pragmatic solution uses a conventional RDBMS

- Already exists in virtually all corporates
- Tied into the existing corporate infrastructure
- Amenable to key storage
  - SELECT key WHERE name='John Doe'
  - SELECT key WHERE expiryDate < today + 1 week

In the future keys might be stored in X.500 directories

The X.500 Directory

The directory contains multiple objects in object classes defined by schemas

A schema defines

- Required attributes
- Optional attributes
- The parent class

Attributes are type-and-value pairs

- Type = DN, value = John Doe
- Type may have multiple values associated with it
- Collective attributes are attributes shared across multiple entries (eg a company-wide fax number)

The X.500 Directory (ctd)

Each instantiation of an object is a directory entry

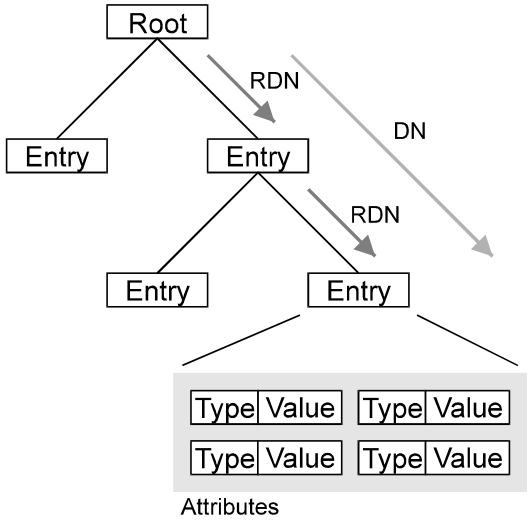
Entries are identified by DN's

- The DN is comprised of relative distinguished names (RDN's) which define the path through the directory

Directory entries may have aliases which point to the actual entry

The entry contains one or more attributes which contain the actual data

The X.500 Directory (ctd)



Data is accessed by DN and attribute type

Searching the Directory

Searching is performed by subtree refinement

- Base specifies where the start in the subtree
- Chop specifies how much of the subtree to search
- Filter specifies the object class to filter on

Example

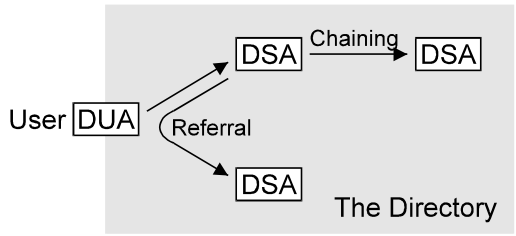
- Base = C=NZ
- Chop = 1 RDN down from the base
- Filter = organisation

Typical application is to populate a tree control for directory browsing

- SELECT name WHERE O=*

Directory Implementation

The directory is implemented using directory service agents (DSA's)



Users access the directory via a directory user agent (DUA)

- Access requests may be satisfied through referrals or chaining

One or more DSA's are incorporated into a management domain

Directory Access

Typical directory accesses:

- Read attribute or attributes from an entry
- Compare supplied value with an attribute of an entry
- List DN's of subordinate entries
- Search entries using a filter
  - Filter contains one or more matching rules to apply to attributes
  - Search returns attribute or attributes which pass the filter
- Add a new leaf entry
- Remove a leaf entry
- Modify an entry by adding or removing attributes
- Move an entry by modifying its DN

LDAP

X.500 Directory Access Protocol (DAP) adapted for Internet use

- Originally Lightweight Directory Access Protocol, now closer to HDAP

Provides access to LDAP servers (and hence DSA's) over a TCP/IP connection

- `bind` and `unbind` to connect/disconnect
- `read` to retrieve data
- `add`, `modify`, `delete` to update entries
- `search`, `compare` to locate information

LDAP (ctd)

LDAP provides a complex heirarchical directory containing information categories with sub-categories containing nested object classes containing entries with one or more (usually more) attributes containing actual values

Simplicity made complex

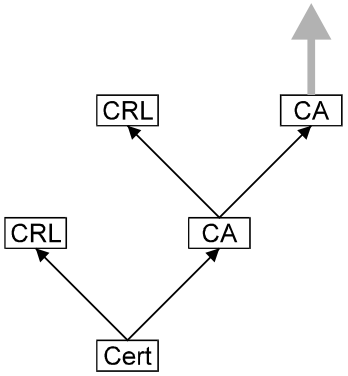
"It will scale up into the billions. We have a pilot with 200 users running already"

Most practical way to use it is as a simple database

SELECT key WHERE name='John Doe'

Certificate Verification using the Directory

Checking works in reverse order to normal lookup



Check certificate
Check certificates CRL
repeat
    Check CA's certificate
    Check CA's CRL
until root reached

Alternative Verification Schemes

Checking as per the X.509 design isn't really practical

- Extremely high overhead for each cert access/use
- Lack of a directory makes locating a cert difficult

Best solution may be some form of DNS-style verification system

- Lightweight at the user end
- User is told either "Yes" or "No"
  - Follows the credit card authorisation model: Reject, card stolen, card expired, amount exceeded, ..., accept
- Revocation is easy
- Information is held at easy-to-maintain centralised sites

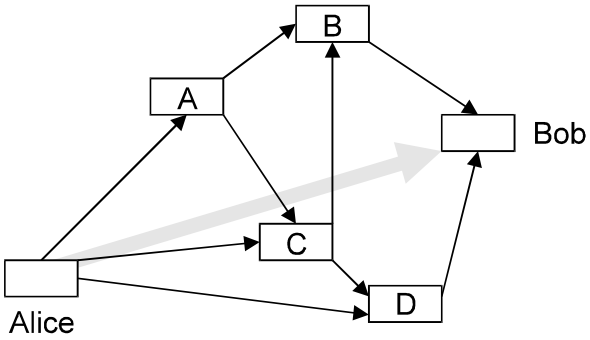
Alternative Verification Schemes (ctd)

Problems with centralised verification

- Requires trusted verification servers
- Users need to register certificates of interest or the problem becomes too large
- Servers become a high-value target

Alternative Trust Hierarchies

PGP web of trust



Bob knows B and D who know A and C who know Alice
⇒ Bob knows the key came from Alice

Web of trust more closely reflects real-life trust models

Certificate Revocation

Revocation is managed with a certificate revocation list (CRL), a form of anti-certificate which cancels a certificate

- Equivalent to 1970's-era credit card blacklist booklets
- Relying parties are expected to check CRL's before using a certificate
  - "This certificate is valid unless you here somewhere that it isn't"

CRL's don't really work

- Difficult to implement and use
- Uncertain performance
- Vulnerable to simple denial-of-service attacks (attacker can prevent revocation by blocking CRL's)

Certificate Revocation (ctd)

Many applications require prompt revocation

- CA's (and X.509) don't really support this
- CA's are inherently an offline operation

Online protocols have been proposed to fix CRL's

- Online Certificate Status Protocol, OCSP
  - Inquires of the issuing CA whether a given certificate is still valid
  - Acts as a simple responder for querying CRL's
  - Still requires the use of a CA to check validity

Certificate Revocation (ctd)

Alternative revocation techniques

- Self-signed revocation (suicide note)
- Certificate of health/warrant of fitness for certificates (anti-CRL)

The general problem may be fixable with quick-turnaround online revocation authorities

- Anyone who can figure out how to make revocation work, please see me afterwards

Key Backup/Archival

Need to very carefully balance security vs backup requirements

- Every extra copy of your key is one more failure point
- Communications and signature keys never need to be recovered — generating a new key only takes a minute or so
- Long-term data storage keys should be backed up

Never give the entire key to someone else

- By extension, never use a key given to you by someone else (eg generated for you by a third party)

Key Backup/Archival (ctd)

Use a threshold scheme to handle key backup

- Break the key into *n* shares
- Any *m* of *n* shares can recover the original
- Store each share in a safe, different location (locked in the company safe, with a solicitor, etc)
- Shares can be reconstructed under certain conditions (eg death of owner)

Defeating this setup requires subverting multiple shareholders

Never give the entire key to someone else

Never give the key shares to an outside third party

Certificate Structure

Version (X.509 v3)
Serial number Issuer name (DN)
Validity (start and end time)
Subject Name (DN) Subject public key
Extensions (added in v3) Extra identification information, usage constraints, policies, etc

Usually either the subject name or issuer and serial number identify the certificate

Certificate Structure (ctd)

Typical certificate

- Serial Number = 177545
- Issuer Name = Verisign
- ValidFrom = 12/09/98
- ValidTo = 12/09/99
- Subject Name = John Doe
- Public Key = RSA public key

Certificate Extensions

Extensions consist of a type-and-value pair, with optional critical flag

Critical flag is used to protect CA's against assumptions made by software which doesn't implement support for a particular extension

- If flag is set, extension must be processed (if recognised) or the certificate rejected
- If flag is clear, extension may be ignored

Ideally, implementations should process and act on all components of all fields of an extension in a manner which is compliant with the semantic intent of the extension

Certificate Extensions (ctd)

Actual definitions of critical flag usage are extremely vague

- X.509: Noncritical extension "is an advisory field and does not imply that usage of the key is restricted to the purpose indicated"
- PKIX: "CA's are required to support constraint extensions", but "support" is never defined
- S/MIME: Implementations should "correctly handle" certain extensions
- MailTrusT: "non-critical extensions are informational only and may be ignored"
- Verisign: "all persons shall process the extension... or else ignore the extension"

Certificate Extensions (ctd)

Extensions come in two types

Usage/informational extensions

- Provide extra information on the certificate and its owner

Constraint extensions

- Constrain the user of the certificate
- Act as a Miranda warning ("You have the right to remain silent, you have the right to an attorney, ...") to anyone using the certificate

Certificate Usage Extensions

Key Usage

- Defines the purpose of the key in the certificate

digitalSignature

- Short-term authentication signature (performed automatically and frequently)

nonRepudiation

- Binding long-term signature (performed consciously)
- Another school of thought holds that nonRepudiation acts as an additional service on top of digitalSignature

Certificate Usage Extensions (ctd)

keyEncipherment

- Exchange of encrypted session keys (RSA)

keyAgreement

- Key agreement (DH)

keyCertSign/cRLSign

- Signature bits used by CA’s

Certificate Usage Extensions (ctd)

Extended Key Usage

Extended forms of the basic key usage fields

- serverAuthentication
- clientAuthentication
- codeSigning
- emailProtection
- timeStamping

Netscape cert-type

An older Netscape-specific extension which performed the same role as keyUsage, extKeyUsage, and basicConstraints

Certificate Usage Extensions (ctd)

Private Key Usage Period

Defines start and end time in which the private key for a certificate is valid

- Signatures may be valid for 10-20 years, but the private key should only be used for a year or two

Alternative Names

Everything which doesn't fit in a DN

- rfc822Name
  - email address, `dave@wetaburgers.com`
- dNSName
  - DNS name for a machine, `ftp.wetaburgers.com`

Certificate Usage Extensions (ctd)

- uniformResourceIdentifier
  - URL, `http://www.wetaburgers.com`
- iPAddress
  - 202.197.22.1 (encoded as CAC51601)
- x400Address, ediPartyName
  - X.400 and EDI information
- directoryName
  - Another DN, but containing stuff you wouldn't expect to find in the main certificate DN
  - Actually the alternative name is a form called the GeneralName, of which a DN is a little-used subset
- otherName
  - Type-and-value pairs (type=MPEG, value=MPEG-of-cat)

Certificate Usage Extensions (ctd)

Certificate Policies

Information on the CA policy under which the certificate is issued

- Policy identifier
- Policy qualifier(s)
- Explicit text ("This certificate isn't worth the paper it's not printed on")

Certificate Usage Extensions (ctd)

X.509 delegates most issues of certificate semantics or trust to the CA's policy

- Many policies serve mainly to protect the CA from liability
  - "Verisign disclaims any warranties... Verisign makes no representation that any CA or user to which it has issued a digital ID is in fact the person or organisation it claims to be... Verisign makes no assurances of the accuracy, authenticity, integrity, or reliability of information"
- Effectively these certificates have null semantics
- If CA's didn't do this, their potential liability would be enormous

Certificate Usage Extensions (ctd)

Policy Mappings

- Maps one CA's policy to another CA
- Allows verification of certificates issued under other CA policies
  - "For verification purposes we consider our CA policy to be equivalent to the policy of CA *x*"

Certificate Constraint Extensions

Basic Constraints

Whether the certificate is a CA certificate or not

- Prevents users from acting as CA's and issuing their own certificates

Name Constraints

Constrain the DN subtree under which a CA can issue certificates

- Constraint of C=NZ, O=University of Auckland would enable a CA to issue certificates only for the University of Auckland
- Main use is to balkanize the namespace so a CA can buy or license the right to issue certificates in a particular area
- Constraints can also be applied to email addresses, DNS names, and URL's

Certificate Constraint Extensions (ctd)

Policy Constraints

Can be used to disable certificate policy mappings

- Policy = "For verification purposes we consider our CA policy to be equivalent to the policy of CA *x*"
- Policy constraint = "No it isn't"

Certificate Profiles

X.509 is extremely vague and nonspecific in many areas

- To make it usable, standards bodies created certificate profiles which nailed down many portions of X.509

PKIX

Internet PKI profile

- Requires certain extensions (basicConstraints, keyUsage) to be critical
  - Doesn't require basicConstraints in end entity certificates, interpretation of CA status is left to chance
- Uses digitalSignature for general signing, nonRepudiation specifically for signatures with nonRepudiation
- Defines Internet-related altName forms like email address, DNS name, URL

Certificate Profiles (ctd)

FPKI

(US) Federal PKI profile

- Requires certain extensions (basicConstraints, keyUsage, certificatePolicies, nameConstraints) to be critical
- Uses digitalSignature purely for ephemeral authentication, nonRepudiation for long-term signatures
- Defines (in great detail) valid combinations of key usage bits and extensions for various certificate types

MISSI

US DoD profile

- Similar to FPKI but with some DoD-specific requirements (you'll never run into this one)

Certificate Profiles (ctd)

ISO 15782

Banking — Certificate Management Part 1: Public Key Certificates

- Uses digitalSignature for entity authentication and nonRepudiation strictly for nonrepudiation (leaving digital signatures for data authentication without nonrepudiation hanging)

Certificate Profiles (ctd)

SEIS

Secured Electronic Information in Society

- Leaves extension criticality up to certificate policies
- Uses digitalSignature for ephemeral authentication and some other signature types, nonRepudiation specifically for signatures with nonRepudiation
- Disallows certain fields (policy and name constraints)

Certificate Profiles (ctd)

TeleTrusT/MailTrusT

German MailTrusT profile for TeleTrusT (it really is capitalised that way)

- Requires keyUsage to be critical in some circumstances
- Uses digitalSignature for general signatures, nonRepudiation specifically for signatures with nonRepudiation

Certificate Profiles (ctd)

Australian Profile

Profile for the Australian PKAF

- Requires certain extensions (basicConstraints, keyUsage) to be critical
- Defines key usage bits (including digitalSignature and nonRepudiation) in terms of which bits may be set for each algorithm type
- Defines (in great detail) valid combinations of key usage bits and extensions for various certificate types

German Profile

Profile to implement the German digital signature law

- Requires that private key be held only by the end user

Certificate Profiles (ctd)

SIRCA Profile

(US) Securities Industry Association

- Requires all extensions to be non-critical
- Requires certificates to be issued under the SIA DN subtree

Microsoft Profile (de facto profile)

- Rejects certificates with critical extensions
- Always seems to set nonRepudiation flag when digitalSignature flag set
- Ignores keyUsage bit
- Treats all certificate policies as the hardcoded Verisign policy

Setting up a CA

Noone makes money running a CA

- You make money by selling CA services and products

Typical cost to set up a proper CA from scratch: $1M

Writing the policy/certificate practice statement (CPS) requires significant effort

Getting the top-level certificate (root certificate) installed and trusted by users can be challenging

- Root certificate is usually self-signed

Bootstrapping a CA

Get your root certificate signed by a known CA

- Your CA's certificate is certified by the existing CA
- Generally requires becoming a licensee of the existing CA
- Your CA is automatically accepted by existing software

Get users to install your CA certificate in their applications

- Difficult for users to do
- Specific to applications and OS's
- Not transparent to users
- No trust mechanism for the new certificate

Bootstrapping a CA (ctd)

Publish your CA certificate(s) by traditional means

- Global Trust Register, `http://www.cl.cam.ac.uk/Research/Security/Trust-Register/`
- Book containing register of fingerprints of the world's most important public keys
- Implements a top-level CA using paper and ink

Install custom software containing the certificate on user PC's

- Even less transparent than manually installing CA certificates
- No trust mechanism for the new certificate

CA Policies

Serves two functions

- Provides a CA-specific mini-profile of X.509
- Defines the CA terms and conditions/indemnifies the CA

CA policy may define

- Obligations of the CA
  - Checking certificate user validity
  - Publishing certificates/revocations
- Obligations of the user
  - Provide valid, accurate information
  - Protect private key
  - Notify CA on private key compromise

CA Policies (ctd)

- List of applications for which issued certs may be used/may not be used
- CA liability
  - Warranties and disclaimers
- Financial responsibility
  - Indemnification of the CA by certificate users
- Certificate publication details
  - Access mechanism
  - Frequency of updates
  - Archiving
- Compliance auditing
  - Frequency and type of audit
  - Scope of audit

CA Policies (ctd)

- Security auditing
  - Which events are logged
  - Period for which logs are kept
  - How logs are protected
- Confidentiality policy
  - What is/isn't considered confidential
  - Who has access
  - What will be disclosed to law enforcement/courts

CA Policies (ctd)

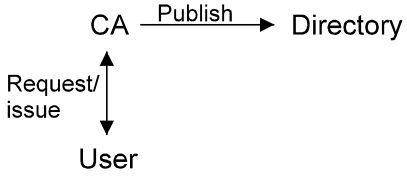
- Certificate issuing
  - Type of identification/authentication required for issuance
  - Type of name(s) issued
  - Resolution of name disputes
  - Handling of revocation requests
    - Circumstances under which a certificate is revoked, who can request a revocation, type of identification/authentication required for revocation, how revocation notices are distributed
- Key changeover
  - How keys are rolled over when existing ones expire
- Disaster recovery

CA Policies (ctd)

- CA security
  - Physical security
    - Site location, access control, fire/flood protection, data backup
  - Personnel security
    - Background checks, training
  - Computer security
    - OS type used, access control mechanisms, network security controls
  - CA key protection
    - Generation, key sizes, protection (hardware or software, which protection standards are employed, key backup/archival, access/control over the key handling software/hardware)
- Certificate profiles
  - Profile amendment procedures
  - Publication

CA's and Scaling

The standard certification model involves direct user interaction with a CA

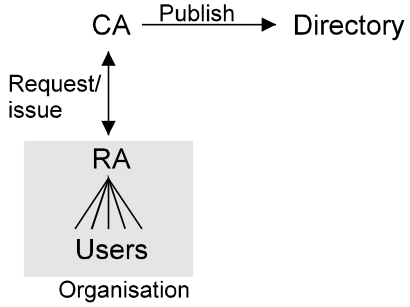


This doesn't scale well

- CA has to verify details for each user
- Processing many users come from a similar background (eg a single organisation) results in unnecessary repeated work

RA's

Registration authorities offload user processing and checking from the CA



RA acts as a trusted intermediary

- RA has a trusted relationship with CA
- RA has access to user details

Timestamping

Certifies that a document existed at a certain time

Used for added security on existing signatures

- Timestamped countersignature proves that the original signature was valid at a given time
- Even if the original signatures key is later compromised, the timestamp can be used to verify that the signature was created before the compromise

Requires a data format which can handle multiple signatures

- Only PGP keys and S/MIME signed data provide this capability

Problems with X.509

Most of the required infrastructure doesn't exist

- Users use an undefined certification request protocol to obtain a certificate which is published in an unclear location in a nonexistant directory with no real means to revoke it
- Various workarounds are used to hide the problems
  - Details of certificate requests are kludged together via web pages
  - Complete certificate chains are included in messages wherever they're needed
  - Revocation is either handled in an ad hoc manner or ignored entirely

Standards groups are working on protocols to fix this

- Progress is extremely slow

Problems with X.509 (ctd)

Certificates are based on owner identities, not keys

- Owner identities don't work very well as certificate ID's
  - Real people change affiliations, email addresses, even names
  - An owner will typically have multiple certificates, all with the same ID
- Owner identity is rarely of security interest (authorisation/capabilities are what count)
- Revoking a key requires revoking the identity of the owner
- Renewal/replacement of identity certificates is nontrivial

Problems with X.509 (ctd)

Authentication and confidentiality certificates are treated the same way for certification purposes

- X.509v1 and v2 couldn't even distinguish between the two

Users should have certified authentication keys and use these to certify their own confidentiality keys

- No real need to have a CA to certify confidentiality keys
- New confidentiality keys can be created at any time
- Doesn't require the cooperating of a CA to replace keys

Aggregation of attributes shortens the overall certificate lifetime

Problems with X.509 (ctd)

Certificates rapidly become a dossier as more attributes are added

```
SEQUENCE {
 OBJECT IDENTIFIER signedData (1 2 840 113549 1 7 2)
 [0] {
  SEQUENCE {
   INTEGER 1
   SET {
    SEQUENCE {
     OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
     NULL
     }
    }
   SEQUENCE {
    OBJECT IDENTIFIER data (1 2 840 113549 1 7 1)
    }
   [0] {
    SEQUENCE {
     SEQUENCE {
      [0] {
       INTEGER 2
       }
      INTEGER 145
      SEQUENCE {
       OBJECT IDENTIFIER md5withRSAEncryption (1 2 840 113549 1 1 4)
       NULL
       }
```

continues

```
SEQUENCE {
 SET {
  SEQUENCE {
   OBJECT IDENTIFIER countryName (2 5 4 6)
   PrintableString 'CH'
   }
  }SET {
  SEQUENCE {
   OBJECT IDENTIFIER organizationName (2 5 4 10)
   PrintableString 'Swisskey AG'
   }
  }
 SET {
  SEQUENCE {
   OBJECT IDENTIFIER organizationalUnitName (2 5 4 11)
   PrintableString 'Public CA Services'
   }
  }
 SET {
  SEQUENCE {
   OBJECT IDENTIFIER localityName (2 5 4 7)
   PrintableString 'Zuerich'
   }
  }
 SET {
  SEQUENCE {
   OBJECT IDENTIFIER commonName (2 5 4 3)
   PrintableString 'Swisskey ID CA 1024'
   }
  }
}
```

Problems with X.509 (ctd)

```
SEQUENCE {
 UTCTime '980929093816Z'
 UTCTime '000929093800Z'
 }
SEQUENCE {
 SET {
  SEQUENCE {
   OBJECT IDENTIFIER organizationName (2 5 4 10)
   PrintableString 'Swisskey AG'
   }
  }
 SET {
  SEQUENCE {
   OBJECT IDENTIFIER organizationalUnitName (2 5 4 11)
   PrintableString '00851000005020000000128'
   }
  }
 SET {
  SEQUENCE {
   OBJECT IDENTIFIER organizationalUnitName (2 5 4 11)
   PrintableString 'Product Management'
   }
  }
 SET {
  SEQUENCE {
   OBJECT IDENTIFIER postalCode (2 5 4 17)
   PrintableString '8008'
   }
  }
 SET {
  SEQUENCE {
   OBJECT IDENTIFIER localityName (2 5 4 7)
   PrintableString 'Zuerich'
   }
  }
 SET {
  SEQUENCE {
   OBJECT IDENTIFIER countryName (2 5 4 6)
   PrintableString 'CH'
   }
  }
 SET {
  SEQUENCE {
   OBJECT IDENTIFIER commonName (2 5 4 3)
   PrintableString 'Juerg Spoerndli'
   }
  }
 SET {
  SEQUENCE {
   OBJECT IDENTIFIER emailAddress (1 2 840 113549 1 9 1)
   IA5String 'jspoerndli@swisskey.ch'
   }
  }
 }
SEQUENCE {
 SEQUENCE {
  OBJECT IDENTIFIER rsaEncryption (1 2 840 113549 1 1 1)
  NULL
  }
 BIT STRING 0 unused bits
  30 81 89 02 81 81 00 EE 7B BA 00 A0 1A C2 05 8B
  8F 52 26 E9 01 C4 A3 7A C9 6E C5 4C 2B FD 3A 2A
  44 48 72 29 7E E3 57 03 2A C9 F3 BB 1D C2 12 2D
  E7 7E 8D B3 3C 58 AD D6 8A 29 4D D1 9F 0F 1E 45
  F3 1E 67 39 9D 83 0B 1A 0D 1F 82 35 B0 D7 2A 6E
  35 6B 76 C2 05 9B 67 E4 3F 8B 6A 8F A6 04 85 F7
  56 EB 51 D9 69 D6 C9 23 AF 5E 0A AE D3 90 7F 60
  16 81 CF 1F 20 B6 A5 A5 5E F0 9F 6D B0 40 F9 8D
        [ Another 12 bytes skipped ]
 }
```

continues

Problems with X.509 (ctd)

```
[3] {
 SEQUENCE {
  SEQUENCE {
   OBJECT IDENTIFIER netscape-cert-type (2 16 840 1 113730 1 1)
   OCTET STRING, encapsulates {
    BIT STRING 5 unused bits
     '101'B
    }
   }
  SEQUENCE {
   OBJECT IDENTIFIER netscape-comment (2 16 840 1 113730 1 13)
   OCTET STRING
    16 81 C6 54 68 69 73 20 63 65 72 74 69 66 69 63
    61 74 65 20 68 61 73 20 62 65 65 6E 20 69 73 73
    75 65 64 20 62 79 20 53 77 69 73 73 6B 65 79 20
    41 47 20 67 6F 76 65 72 6E 65 64 20 62 79 20 69
    74 73 20 43 65 72 74 69 66 69 63 61 74 65 20 50
    72 61 63 74 69 63 65 20 53 74 61 74 65 6D 65 6E
    74 20 28 43 50 53 29 2E 20 43 50 53 20 61 6E 64
    20 66 75 72 74 68 65 72 20 69 6E 66 6F 72 6D 61
          [ Another 73 bytes skipped ]
   }
  SEQUENCE {
   OBJECT IDENTIFIER keyUsage (2 5 29 15)
   OCTET STRING, encapsulates {
    BIT STRING 5 unused bits
     '101'B
    }
   }
  SEQUENCE {
   OBJECT IDENTIFIER privateKeyUsagePeriod (2 5 29 16)
   OCTET STRING, encapsulates {
    SEQUENCE {
     [0] '19980929093816Z'
     [1] '20000929093800Z'
     }
    }
   }
  }
 }
SEQUENCE {
 OBJECT IDENTIFIER md5withRSAEncryption (1 2 840 113549 1 1 4)
 NULL
 }
BIT STRING 0 unused bits
 2A 2A 40 C4 03 48 0B B9 7D 7F B6 85 FD CF A8 D7
 CF 96 D8 55 5D C0 87 4D BE E6 C1 0F 7A 0B 0F 17
 DF 7A 10 49 81 EB A1 6B 8C 16 93 FB 38 37 79 A0
 B6 1F B3 EA F0 AA D5 CA 0A 52 DA D3 19 3A 55 B6
 F6 7F 77 4E 30 15 D4 5C 8C 73 44 62 FF 15 9C 44
 C3 38 F0 D1 58 85 D0 C6 88 55 7C FF D0 67 14 4C
 DE D2 7F F8 00 A8 BC 6E A7 35 BD 51 DD CB 7D F2
 C8 E7 34 61 00 C2 25 51 F0 ED 0B B0 38 93 FC 30
}
SEQUENCE {
 SEQUENCE {
  [0] {
   INTEGER 2
   }
  INTEGER 5
  SEQUENCE {
   OBJECT IDENTIFIER md5withRSAEncryption (1 2 840 113549 1 1 4)
   NULL
   }
```

continues

Problems with X.509 (ctd)

```
SEQUENCE {
 SET {
  SEQUENCE {
   OBJECT IDENTIFIER countryName (2 5 4 6)
   PrintableString 'CH'
  }
 }
 SET {
  SEQUENCE {
   OBJECT IDENTIFIER organizationName (2 5 4 10)
   PrintableString 'Swisskey AG'
  }
 }
 SET {
  SEQUENCE {
   OBJECT IDENTIFIER organizationalUnitName (2 5 4 11)
   PrintableString 'Public CA Services'
  }
 }
 SET {
  SEQUENCE {
   OBJECT IDENTIFIER localityName (2 5 4 7)
   PrintableString 'Zuerich'
  }
 }
 SET {
  SEQUENCE {
   OBJECT IDENTIFIER commonName (2 5 4 3)
   PrintableString 'Swisskey Root CA'
  }
 }
}
SEQUENCE {
 UTCTime '9807061348492'
 UTCTime '0512312359002'
}
```

```
SEQUENCE {
 SET {
  SEQUENCE {
   OBJECT IDENTIFIER countryName (2 5 4 6)
   PrintableString 'CH'
  }
 }
 SET {
  SEQUENCE {
   OBJECT IDENTIFIER organizationName (2 5 4 10)
   PrintableString 'Swisskey AG'
  }
 }
 SET {
  SEQUENCE {
   OBJECT IDENTIFIER organizationalUnitName (2 5 4 11)
   PrintableString 'Public CA Services'
  }
 }
 SET {
  SEQUENCE {
   OBJECT IDENTIFIER localityName (2 5 4 7)
   PrintableString 'Zuerich'
  }
 }
 SET {
  SEQUENCE {
   OBJECT IDENTIFIER commonName (2 5 4 3)
   PrintableString 'Swisskey ID CA 1024'
  }
 }
}
SEQUENCE {
 SEQUENCE {
  OBJECT IDENTIFIER rsaEncryption (1 2 840 113549 1 1 1)
  NULL
 }
```

continues

Problems with X.509 (ctd)

```
 BIT STRING 0 unused bits
  30 81 89 02 81 81 00 AB E9 1F E9 AD FF 53 9F 71
  70 35 6D F8 F8 4C 76 B4 F7 43 E8 19 80 DD A9 0A
  D6 4E 60 C2 FD 48 7B 43 F6 6E BE 53 D0 0E 62 F0
  35 27 6F 2E 55 22 F2 82 40 2E 21 5B 5D 7E 18 16
  CA 87 31 2E 12 71 4C 5F 92 8A AB 36 61 9C 91 3B
  BC BD 95 88 BF 7E 0C 4A D7 A0 12 F9 FA FF 0F 84
  F8 57 6E DE AE B4 03 FC 77 CF 7C E5 B3 33 79 61
  31 4E CE 70 03 E7 73 D8 E8 1B D3 EB 15 FF 69 B3
        [ Another 12 bytes skipped ]
 }
[3] {
 SEQUENCE {
  SEQUENCE {
   OBJECT IDENTIFIER basicConstraints (2 5 29 19)
   BOOLEAN TRUE
   OCTET STRING, encapsulates {
    SEQUENCE {
     BOOLEAN TRUE
     INTEGER 0
    }
   }
  }
  SEQUENCE {
   OBJECT IDENTIFIER keyUsage (2 5 29 15)
   BOOLEAN TRUE
   OCTET STRING, encapsulates {
    BIT STRING 1 unused bits
     '1100000'B
    }
   }
  }
 }
}
```

```
SEQUENCE {
 OBJECT IDENTIFIER md5withRSAEncryption (1 2 840 113549 1 1 4)
 NULL
}
BIT STRING 0 unused bits
 0E 0F 67 22 AA D2 8A 7B BF 3D 47 AB 1F 5E 8C F3
 2C 32 3E AB D3 48 60 A1 BA 49 FD 81 28 6A 26 69
 83 97 29 1F E8 80 14 96 30 2B C3 18 97 3B 6C F3
 F0 A2 D6 E0 30 EF F6 2C 38 1F C0 37 7E 9E 45 ED
 62 38 67 07 27 BE 81 07 E9 12 60 E8 BE 6B ED 14
 8E 61 17 52 99 C2 FE 33 B7 21 CA 5E FE 6D B4 1E
 B9 8C 54 36 42 55 1E 73 D9 81 DE 5D 25 AD 72 39
 15 AF 68 E9 44 45 55 7F 2E 2E F9 6F EF 44 B0 E0
}
SEQUENCE {
 SEQUENCE {
  [0] {
   INTEGER 2
  }
  INTEGER 1
  SEQUENCE {
   OBJECT IDENTIFIER md5withRSAEncryption (1 2 840 113549 1 1 4)
   NULL
  }
  SEQUENCE {
   SET {
    SEQUENCE {
     OBJECT IDENTIFIER countryName (2 5 4 6)
     PrintableString 'CH'
    }
   }
   SET {
    SEQUENCE {
     OBJECT IDENTIFIER organizationName (2 5 4 10)
     PrintableString 'Swisskey AG'
    }
   }
```

continues

Problems with X.509 (ctd)

```
SET {
  SEQUENCE {
    OBJECT IDENTIFIER organizationalUnitName (2 5 4 11)
    PrintableString 'Public CA Services'
    }
  }
 SET {
  SEQUENCE {
    OBJECT IDENTIFIER localityName (2 5 4 7)
    PrintableString 'Zuerich'
    }
  }
 SET {
  SEQUENCE {
    OBJECT IDENTIFIER commonName (2 5 4 3)
    PrintableString 'Swisskey Root CA'
    }
  }
 }
SEQUENCE {
 UTCTime '980706120207Z'
 UTCTime '0512312359002'
 }
SEQUENCE {
 SET {
  SEQUENCE {
    OBJECT IDENTIFIER countryName (2 5 4 6)
    PrintableString 'CH'
    }
  }
 SET {
  SEQUENCE {
    OBJECT IDENTIFIER organizationName (2 5 4 10)
    PrintableString 'Swisskey AG'
    }
  }
 SET {
  SEQUENCE {
    OBJECT IDENTIFIER organizationalUnitName (2 5 4 11)
    PrintableString 'Public CA Services'
    }
  }
 SET {
  SEQUENCE {
    OBJECT IDENTIFIER localityName (2 5 4 7)
    PrintableString 'Zuerich'
    }
  }
 SET {
  SEQUENCE {
    OBJECT IDENTIFIER commonName (2 5 4 3)
    PrintableString 'Swisskey Root CA'
    }
  }
 }
SEQUENCE {
 SEQUENCE {
  OBJECT IDENTIFIER rsaEncryption (1 2 840 113549 1 1 1)
  NULL
  }
```

continues

Problems with X.509 (ctd)

```
 BIT STRING 0 unused bits
  30 81 89 02 81 81 00 AC AB 60 E0 C5 69 FD 07 4E
  97 9B AF 4A 1C 30 D7 68 26 D1 2C 3D 44 F0 D6 AB
  16 34 6F 00 D8 7F D6 3F B9 35 D6 83 28 77 A3 3E
  24 5D A4 D1 C2 FA 04 B3 DB 4D 38 91 23 70 6C 2B
  2D 48 69 D5 15 6F 4A 9F 91 BC E4 83 2F 35 A2 29
  DB 55 66 F8 90 C6 0E 0C 32 75 95 24 E0 8D B7 8E
  AB 13 70 61 1E 01 91 7D 9D 44 37 42 41 C9 C2 01
  DD 26 D8 B9 2C 29 57 A1 54 17 1E AC 1A DE 8C 6C
       [ Another 12 bytes skipped ]
 }
[3] {
 SEQUENCE {
  SEQUENCE {
    OBJECT IDENTIFIER basicConstraints (2 5 29 19)
    BOOLEAN TRUE
    OCTET STRING, encapsulates {
      SEQUENCE {
        BOOLEAN TRUE
        INTEGER 3
        }
      }
    }
  SEQUENCE {
    OBJECT IDENTIFIER keyUsage (2 5 29 15)
    BOOLEAN TRUE
    OCTET STRING, encapsulates {
      BIT STRING 1 unused bits
        '1100000'B
      }
    }
  }
 }
}
SEQUENCE {
 OBJECT IDENTIFIER md5withRSAEncryption (1 2 840 113549 1 1 4)
 NULL
 }
BIT STRING 0 unused bits
 72 A7 93 A3 CD D7 3A DB 79 50 DB 98 03 52 B0 CD
 AF 0C D2 A6 89 38 52 6C 5C E9 7C B3 37 3C 9E 94
 C4 74 57 D4 BB 78 05 5B B6 B9 31 04 FC 60 33 51
 5F CF 2C 44 55 85 EC 1F 0B CB 89 E7 F0 93 D4 CD
 85 D3 FF B6 B5 99 D3 7C 35 06 11 7B 0E 9F E6 BE
 99 B3 49 D0 5A 85 FA 7C BA 54 9B B9 AF F7 4B E3
 FF DC 83 4A 04 F8 F9 A5 1D EC 37 AE C6 23 4C 9D
 B2 01 1F D4 26 EA E4 4A 7E BE BE 1E 11 1E 27 D1
}
SET {
 SEQUENCE {
  INTEGER 1
  SEQUENCE {
   SEQUENCE {
    SET {
     SEQUENCE {
      OBJECT IDENTIFIER countryName (2 5 4 6)
      PrintableString 'CH'
      }
     }
    SET {
     SEQUENCE {
      OBJECT IDENTIFIER organizationName (2 5 4 10)
      PrintableString 'Swisskey AG'
      }
     }
    SET {
     SEQUENCE {
      OBJECT IDENTIFIER organizationalUnitName (2 5 4 11)
      PrintableString 'Public CA Services'
      }
     }
```

continues

Problems with X.509 (ctd)

```
    SET {
     SEQUENCE {
      OBJECT IDENTIFIER localityName (2 5 4 7)
      PrintableString 'Zuerich'
      }
     }
    SET {
     SEQUENCE {
      OBJECT IDENTIFIER commonName (2 5 4 3)
      PrintableString 'Swisskey ID CA 1024'
      }
     }
    }
   INTEGER 145
   }
  SEQUENCE {
   OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
   NULL
   }
  [0] {
   SEQUENCE {
    OBJECT IDENTIFIER contentType (1 2 840 113549 1 9 3)
    SET {
     OBJECT IDENTIFIER data (1 2 840 113549 1 7 1)
     }
    }
   SEQUENCE {
    OBJECT IDENTIFIER signingTime (1 2 840 113549 1 9 5)
    SET {
     UTCTime '981113072133Z'
     }
    }
```

```
   SEQUENCE {
    OBJECT IDENTIFIER messageDigest (1 2 840 113549 1 9 4)
    SET {
     OCTET STRING
      2F 7E 95 9F 34 AC 85 B8 1C 53 9E 5C F8 60 BE 3A
      AA D0 30 B5
     }
    }
   SEQUENCE {
    OBJECT IDENTIFIER sMIMECapabilities (1 2 840 113549 1 9 15)
    SET {
     SEQUENCE {
      SEQUENCE {
       OBJECT IDENTIFIER des-EDE3-CBC (1 2 840 113549 3 7)
       }
      SEQUENCE {
       OBJECT IDENTIFIER rc2CBC (1 2 840 113549 3 2)
       INTEGER 128
       }
      SEQUENCE {
       OBJECT IDENTIFIER desCBC (1 3 14 3 2 7)
       }
      SEQUENCE {
       OBJECT IDENTIFIER rc2CBC (1 2 840 113549 3 2)
       INTEGER 64
       }
      SEQUENCE {
       OBJECT IDENTIFIER rc2CBC (1 2 840 113549 3 2)
       INTEGER 40
       }
      }
     }
    }
   }
```

continues

Problems with X.509 (ctd)

```
  SEQUENCE {
   OBJECT IDENTIFIER rsaEncryption (1 2 840 113549 1 1 1)
   NULL
   }
  OCTET STRING
   9F EC C4 B4 B2 5A FE 87 EA 28 22 C2 6A 1F E3 2F
   16 8D 01 EA 2F 35 0E 13 D1 3E BE 1D 92 48 EF F0
   8E BB BC 98 3B 11 44 88 A8 20 AE AB 65 2D 98 E1
   3E 62 E1 47 5F FE 18 39 AF 97 29 7E D1 68 03 F1
   03 78 44 DB A1 BB 9F 3B C9 89 D5 0D 00 B3 0B FA
   98 F8 2E 58 4C E4 4F 73 02 D6 17 41 84 B6 50 A2
   94 F8 E2 6F C3 78 AF 4D 71 CF E7 FF 25 97 B9 00
   CC A5 BE A8 8C 3D 52 43 C9 BB 41 A9 87 5F 85 6F
  }
 }
}
```

All this from a standard S/MIME signature!

Problems with X.509 (ctd)

Hierarchical certification model doesn't fit typical business practices

- Businesses generally rely on bilateral trading arrangements or existing trust relationships
- Third-party certification is an unnecessary inconvenience when an existing relationship is present

X.509 PKI model entails building a parallel trust infrastructure alongside the existing, well-established one

- In the real world, trust and revocation is handled by closing the account, not with PKI's, CRL's, certificate status checks, and other paraphernalia

PGP Certificates

Certificates are key-based, not identity-based

- Keys can have one or more free-form names attached
- Key and name(s) are bound through (independent) signatures

Certification model can be hierarchical or based on existing trust relationships

- Parties with existing relationships can use self-signed certificates
  - Self-signed end entity certificates are a logical paradox in X.509v3

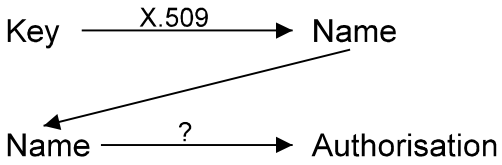
Authentication keys are used to certify confidentiality keys

- Confidentiality keys can be changed at any time, even on a per-message basis

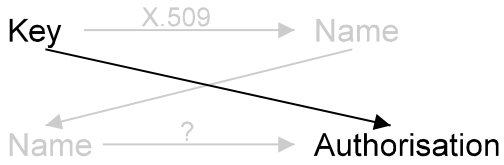
SPKI

Simple Public Key Infrastructure

Identity certificates bind a key to a name, but require a parallel infrastructure to make use of the result



SPKI certificates bind a key to an authorisation or capability



SPKI (ctd)

Certificates may be distributed by direct communications or via a directory

Each certificate contains the minimum information for the job (cf X.509 dossier certificates)

If names are used, they only have to be locally unique

- Global uniqueness is guaranteed by the use of the key as an identifier
- Certificates may be anonymous (eg for balloting)

Authorisation may require *m* of *n* consensus among signers (eg any 2 of 3 company directors may sign)

SPKI Certificate Uses

Typical SPKI uses

- Signing/purchasing authority
- Letter of introduction
- Security clearance
- Software licensing
- Voter registration
- Drug prescription
- Phone/fare card
- Baggage claim check
- Reputation certificate (eg Better Business Bureau rating)
- Access control (eg grant of administrator privileges under certain conditions)

Certificate Structure

SPKI certificates use collections of assertions expressed as LISP-like S-expressions of the form (*type value(s)*)

(name fred) ⇒ Owner name = fred

(name *CA_root CA1 CA2 ... CAn leaf_cert*) ⇒ X.500 DN

(name (hash sha1 |TLCgPLFlGTzyUbcaYLW8kGTEnUk=|) fred) ⇒ Globally unique name with key ID and locally unique name

(ftp (host ftp.warez.org)) ⇒ Keyholder is allowed FTP access to an entire site

(ftp (host ftp.warez.org) (dir /pub/warez)) ⇒ Keyholder is allowed FTP access to only one directory on the site

Certificate Structure (ctd)

```
( cert
    ( issuer ( hash sha1 |TLCgPLFlGTzyUbcaYLW8kGTEnUk=|
      ) )
    ( subject ( hash sha1 |Ve1L/7MqiJcj+LSa/l10fl3tuTQ=l| ) )
    ...
    ( not-before “1998-03-01_12:42:17” )
    ( not-after “2012-01-01_00:00:00” )
  ) ⇒ X.509 certificate
```

Internally, SPKI certificates are represented as 5-tuples <Issuer, Subject, Delegation, Authority, Validity>

- Delegation = Subject has permission to delegate authority
- Authority = Authority granted to certificate subject
- Validity = Validity period and/or online validation test information

Trust Evaluation

5-tuples can be automatically processed using a general-purpose tuple reduction mechanism

<I1, S1, D1, A1, V1> + <I2, S2, D2, A2, V2>
⇒ <I1, S2, D2, intersection(A1, A2), intersection(V1, V2)
if S1 = I2 and D1 = true

Eventually some chains of authorisation statements will reduce to <Trusted Issuer, *x*, D, A, V>

- All others are discarded
- Trust management decisions can be justified/explained/verified
  - “How was this decision reached?”
  - “What happens if I change this bit?”
- X.509 has nothing even remotely like this

Digital Signature Legislation

A signature establishes validity and authentication of a document to allow the reader to act on it as a statement of the signers intent

Signatures represent a physical manifestation of consent

A digital signature must provide a similar degree of security

Digital Signature Legislation (ctd)

Typical signature functions are

- Identification
- Prove involvement in the act of signing
- Associate the signer with a document
- Provide proof of the signers involvement with the content of the signed document
- Provide endorsement of authorship
- Provide endorsement of the contents of a document authored by someone else
- Prove a person was at a given place at a given time
- Meet a statutory requirement that a document be signed to make it valid

General Requirements for Digital Signatures

The signing key must be controlled entirely by the signer for non-repudiation to function

The act of signing must be conscious

- "Grandma clicks the wrong button and loses her house"
- "You are about to enter into a legally binding agreement which stipulates that ..."

May require a traditional written document to back up the use of electronic signatures

- "With the key identified by ... I agree to ... under the terms ..."

Cross-jurisdictional signatures are a problem

Utah Digital Signature Act

First digital signature act, passed in 1995

The Law of X.509

- Requires public-key encryption based signatures, licensed CA's, CRL's, etc etc.

Duly authorised digital signatures may be used to meet statutory requirements for written signatures

Liability of CA's is limited, signers and relying parties assume the risk

Signature carries evidentiary weight of notarised document

- If your key is compromised, you're in serious trouble
- If you hand over your key to a third party, you're in serious trouble

California Digital Signature Law

Very broad, allows any agreed-upon mark to be used as a digital signature

- Western culture has no real analog for this
- Asia has chop-marks, a general-purpose mark used to authenticate and authorise

One-sentence digital signature law:

"You can't refuse a signature just because it's digital"

Massachusetts Electronic Records and Signatures Bill

"A signature may not be denied legal effect, validity, or enforceability because it is in the form of an electronic signature. If a rule of law requires a signature [...] an electronic signature satisfies that rule of law"

"A contract between business entities shall not be unenforceable, nor inadmissible in evidence, on the sole ground that the contract is evidenced by an electronic record or that it has been signed by an electronic signature"

The Massachusetts law doesn't legislate forms of signatures or the use of CA's, or allocate liability

- "Attorneys Full Employment Act of 1997"

German Digital Signature Law

Like the Utah act, based on public-key technology

Requirements

- Licensed CA's which meet certain requirements
  - CA's must provide a phone hotline for revocation
- Identification is based on the German ID card
  - This type of identification isn't possible in most countries
  - Allows pseudonyms in certificates
- Key and storage media must be controlled only by the key owner
- Provisions for timestamping and countersigning

Signatures from other EU countries are recognised provided an equivalent level of security is employed

German Digital Signature Law (ctd)

Details are set out in the implementation guidelines

- Extremely detailed (over 300 pages)
- Specifies things like
  - Hash and signature algorithms
  - Random number generation for keys
  - Personnel security
  - Directory and timestamping services
- Criticised as being too detailed and complex to follow

UNCITRAL Model Law on Electronic Commerce

UN Comission on International Trade (UNCITRAL) model e-commerce law

- Many acts and laws legislate a particular technology to provide reliance for digital signatures
- The model law provides a general framework for electronic signatures without defining their exact form

Later revisions may nail down precise forms for electronic signatures

EU Directive on Electronic Signatures

Defines an electronic signature as linking signer and data, created by a means solely controlled by the signer (not necessarily a cryptographic signature)

Precedes the directive itself with the intended aims of the directive

Makes accreditation and licensing voluntary and non-discriminatory

- No-one can be prevented from being a CA
- Intent is to encourage best practices while letting the market decide

EU Directive on Electronic Signatures (ctd)

Electronic signature products must be made freely available within the EU

Electronic signatures can't be denied recognition just because they're electronic

Absolves CA's of certain types of liability

- Provides for reliance limits in certificates

Recognises certificates from non-EU states issued under equivalent terms

Allows for pseudonyms in certificates

EU Directive on Electronic Signatures (ctd)

Recognises that a regulatory framework isn't needed for signatures used in closed systems

- Trust is handled via existing commercial relationships
- Parties may agree among themselves on terms and conditions for electronic signatures
- Keys may be identified by a key fingerprint on a business card or in a letterhead

Session-Level Security

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

— Don Kitchen

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

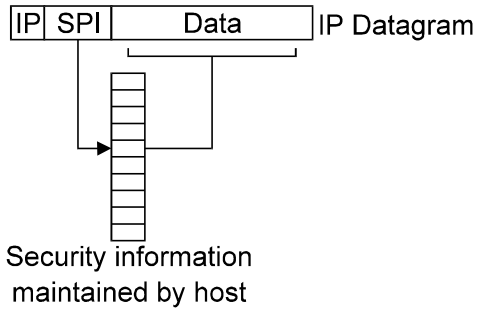
Comprised of two parts:

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

IPSEC Architecture

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

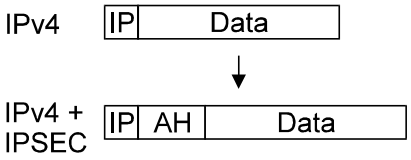
- SA used to provide authentication/confidentiality for that session
- SA is referenced via a security parameter index (SPI) in each IP datagram header



AH

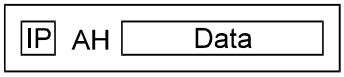
Authentication header — integrity protection only

Inserted into IP datagram:



AH (ctd)

Authenticates entire datagram:

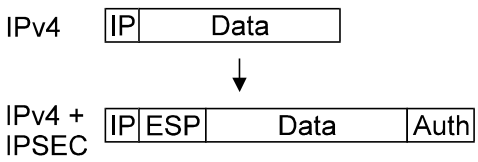


Mutable fields (time-to-live, IP checksums) are zeroed before AH is added

ESP

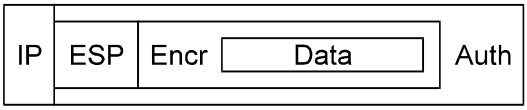
Encapsulating security protocol — authentication and confidentiality

Inserted into IP datagram:



ESP (ctd)

Secures data payload in datagram:



Encryption protects payload

- Authentication protects header and encryption

IPSEC Algorithms

DES for encryption

HMAC/MD5 and SHA for authentication

Others optional

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)
- Tunneling mode (secure IP inside standard IP)

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

Cookie based on IP address and port, stops flooding attacks

- Later adopted by other IPSEC key management protocols

Photuris (ctd)

Client		Server
Client cookie	→	
	←	Server cookie Offered schemes
Chosen scheme	→	
DH keygen	↔	DH keygen
Client identity	→	
Authentication for previous data		
	←	Server identity Authentication for previous data

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)

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

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

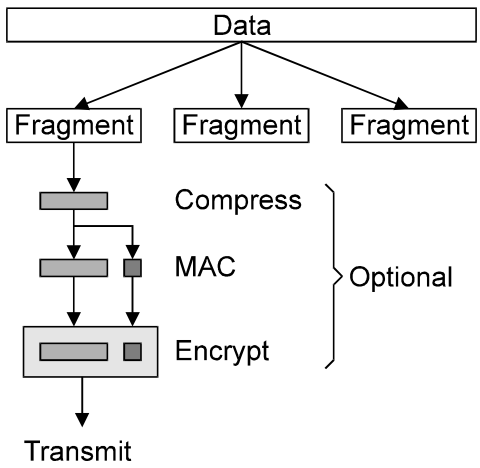
- MAC of previously exchanged parameters (authenticates data from Hello and other exchanges)

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

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:                 Export 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

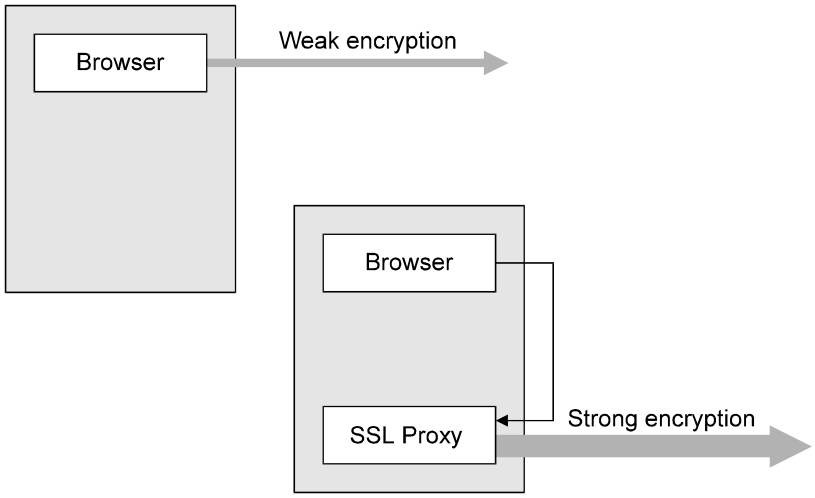
Mostly Unix-only, some NT ports in progress

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

Available to “qualified financial institutions”

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

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)

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

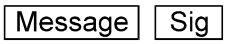
Other requirements

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

Many of these are difficult or impossible to achieve

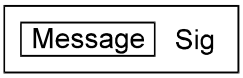
Security Mechanisms

Detached signature:



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

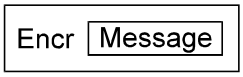
Signed message



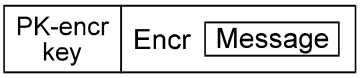
- Signature is always included with the data

Security Mechanisms (ctd)

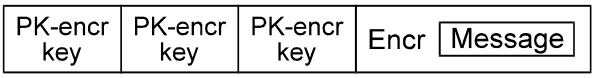
Encrypted message



Usually implemented using public-key encryption



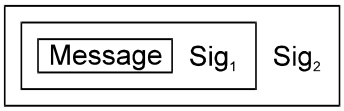
Mailing lists use one public-key encrypted header per recipient



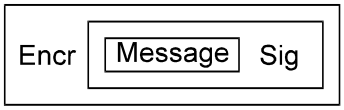
- Any of the corresponding private keys can decrypt the session key and therefore the message

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

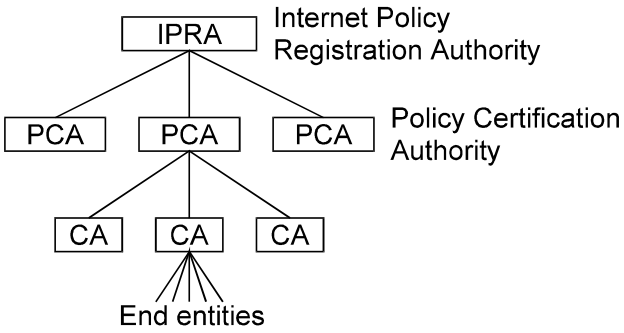
PEM Protection Types (ctd)

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

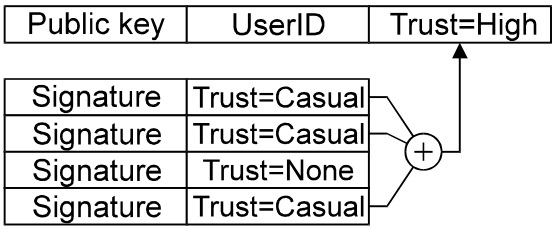
- UserID trust = High
- Key trust = Low

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

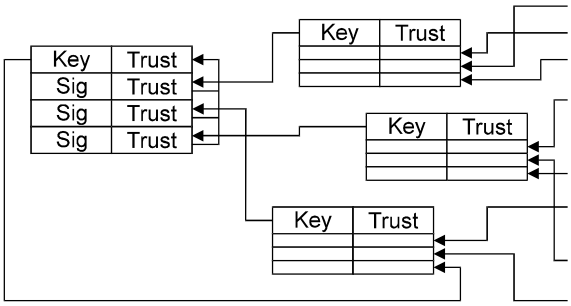
Keys are revoked with a signed revocation which PGP adds to the key

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

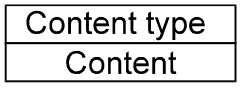
- PKCS, Public Key Cryptography Standards
  - RC2, RC4 for data encryption
  - PKCS #1, RSA encryption, for key exchange
  - PKCS #7, cryptographic message syntax, for message formatting

Newer versions added non-proprietary and non-patented ciphers

CMS

Cryptographic Message Syntax

- Type-and-value format



Data content types

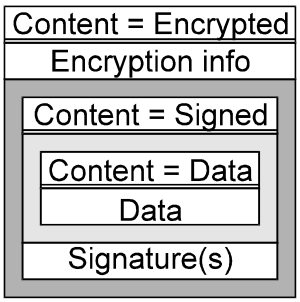
- Data
- Signed data
- Encrypted data (conventional encryption)
- Enveloped data (PKC-encrypted)
- Digested (hashed) data
- Authenticated (MAC'd) data

CMS (ctd)

Other content types possible

- Private keys
- Key management messages

Content can be arbitrarily nested



Signed Data Format

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

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

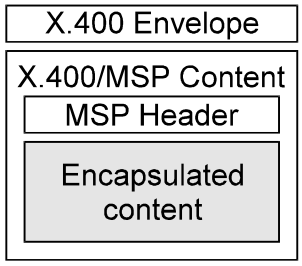
Original S/MIME based on patented RSA and proprietary RC2, rejected by IETF as a standard

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

Authentication

"What was your username again?" *clickety clickety*

— The BOFH

User Authentication

Basic system uses passwords

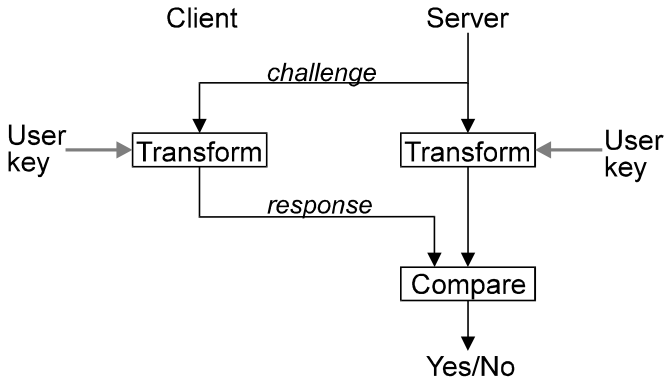
- Can be easily intercepted

Encrypt/hash the password

- Can still intercept the encrypted/hashed form

Modify the encryption/hashing so the encrypted/hashed value changes each time (challenge/response mechanism)

User Authentication (ctd)



Vulnerable to offline password guessing (attacker knows challenge and encrypted challenge, can try to guess the password used to process it)

User Authentication (ctd)

There are many variations of this mechanism but it's *very* hard to get right

- Impersonation attacks (pretend to be client or server)
- Reflection attacks (bounce the authentication messages elsewhere)
- Steal client/server authentication database
- Modify messages between client and server
- Chess grandmaster attack

Simple Client/Server Authentication

Client and server share a key K

- Server sends challenge encrypted with K
  - Challenge should generally include extra information like the server ID and timestamp
- Client decrypts challenge, transforms it (eg adds one, flips the bits), re-encrypts it with K, and sends it to the server
- Server does the same and compares the two

Properties

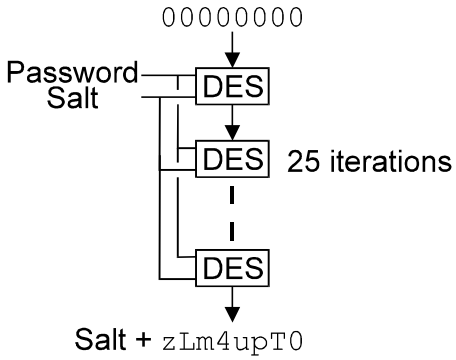
- Both sides are authenticated
- Observer can't see the (unencrypted) challenge, can't perform a password-guessing attack
- Requires reversible encryption

Something similar is used by protocols like Kerberos

Unix Password Encryption

Designed to resist mid-70's level attacks

Uses 25 iterations of modified DES



Salt prevents identical passwords from producing the same output

crypt16

Handles 16 characters instead of 8

- 20 DES crypts for the first 8
- 5 DES crypts for the second 8

Result was weaker than the original crypt

- Search for passwords by suffix
- Suffix search requires only 5 DES crypts

LMHASH

From MS LAN Manager

Like crypt16 but without the salt

- If password < 7 chars, second half is 0xAAD3B435B51404EE
- Result is zero-padded(!) and used as 3 independant DES keys
- 8-byte challenge from server is encrypted once with each key
- $3 \times$ 8-byte value returned to server

Newer versions of NT added NTHASH (MD4 of data), but LMHASH data is still sent alongside NTHASH data

Subject to rollback attacks (“I can’t handle this, give me LMHASH instead”)

LMHASH (ctd)

l0phtcrack, `http://www.l0pht.com/l0phtcrack/`

- Collects hashed passwords via network sniffing, from the registry, or from SAM file on disk
- Two attack types
  - Dictionary search: 100,000 words in a few minutes on a PPro 200
  - Brute force: All alphabetic characters in 6 hours, all alphanumerics in 62 hours on quad PPro 200
- Parallel attacks on multiple users
- Runs as a background process

NT Domain Authentication

Joining

- Client and server exchange challenges CC and SC
- Session key = CC + CS encrypted with the machine password (NTHASH of LMHASH of machine name)
- Result is used as RC4 key

Anyone on the network can intercept this and recover the initial key

NT Domain Authentication (ctd)

User logon

- Client sends RC4-encrypted LMHASH and NTHASH of user password
- Server decrypts and verifies LMHASH and NTHASH of password

RC4 key is reused, can be recoverd in the standard manner for a stream cipher

Auxiliary data (logon script, profile, SID) aren't authenticated

This is why NT 5 will use Kerberos

Attacking Domain Authentication over the Net

Create a web page with an embedded (auto-loading) image

- Image is held on an SMB Lanman server instead of an HTTP server
- NT connects to server
- Server sends fixed all-zero challenge
- NT responds with the username and hashed password

All-zero challenge allows the use of a precomputed dictionary

Attacking Domain Authentication over the Net (ctd)

Reflection attack

- NT connects to the server as before
- Server connects back to NT
- NT sends challenge to server, server bounces it back to NT
- NT responds with the username and hash, server bounces it back to NT

Server has now connected to the NT machine without knowing the password

Netware Authentication

Netware 3 used challenge-response method

- Server sends challenge
- Client responds with MD4(MD4(serverID/salt, password), challenge)
- Server stores (server-dependant) inner hash
  - Not vulnerable to server compromise because of serverID/salt

Netware 4 added public-key encryption managed via the Netware Directory Services (NDS)

Netware Authentication (ctd)

Users public/private keys are stored by NDS, accessed with a modification of the V3 protocol

- Server sends challenge
- Client responds with server-public-key encrypted V3 hash and session key
- Server returns users RSA key encrypted with the session key

Once the user has their RSA key pair on the workstation, they can use it for further authentication

- RSA key is converted to short-term Gillou-Quisquater (GQ) key
- RSA key is deleted
- GQ key is used for authentication

Netware Authentication (ctd)

Compromise of GQ key doesn’t compromise the RSA key

GQ is much faster than RSA for both key generation and authentication

GQ is used to authenticate the user

- Sign request with short-term GQ key
- Authenticate GQ key with long-term RSA key (from NDS)

All valuable information is deleted as quickly as possible

- Only the short-term GQ key remains active

Kerberos

Designed at MIT based on late-70's work by Needham and Schroeder

Relies on key distribution centre (KDC) to perform mediated authentication

KDC shares a key with each client and server

Kerberos (ctd)

When a client connects to a server

- KDC sends to client
  - Session key encrypted with clients key
  - Session key + client ID encrypted with servers key
- User forwards the latter (a ticket) to the server
- User decrypts session key, server decrypts ticket to recover client ID and session key
  - Only the client can recover the client-encrypted session key
  - Only the server can recover the server-encrypted session key

Kerberos (ctd)

Ticket identifies the client and clients network address (to stop it being used elsewhere)

To avoid long-term password storage, the users password is converted to a short-term client key via the KDC

- KDC sends a short-term client key encrypted with the users password to the client
- User decrypts the short-term client key, decrypts the password
- Future KDC $\leftrightarrow$ client communications use the short-term client key

Kerberos (ctd)

KDC also sends out a ticket-granting ticket (TGT)

- TGT contains the client short-term key encrypted with the KDC key
- Based on a theoretical Kerberos model which separates the authentication server and ticket-granting server
  - KDC/AS issues the TGT to talk to the KDC/TGS

Mutual Authentication

Parties exchange session-key encrypted timestamps

- Only holders of the shared session key can get the encryption right
- Replays detected by checking the timestamp on each exchange
- Messages older than a certain time are automatically rejected

KDC's can be replicated to increase availability

- Kerberos is designed so that most database accesses are read-only, making replication easier

Kerberos Realms

Problems with a single KDC database

- Compromising a KDC can compromise many users
- Big KDC databases are difficult to manage
- Big databases lead to name clashes

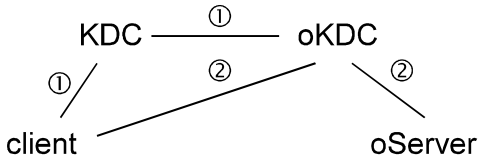
Solution is to use multiple realms

Interrealm authentication is just an extension of the standard Kerberos authentication mechanism

Kerberos Realms (ctd)

When a client connects to a server in another realm

- KDC authenticates client to other realm's KDC
- Other realm's KDC authenticates client to other realm's server



Multi-KDC chaining is disallowed for security reasons (a rogue KDC in the chain could allow anyone in)

Kerberos V5

Extended V4 in various ways

- Extended ticket lifetimes (V4 max = 21 hours)
- Allowed delegation of rights
- Allowed heirarchical realms
- Added algorithms other than DES
- V4 used ad hoc encoding, V5 used ASN.1

Ticket Lifetimes

V4 allowed maximum 21 hour lifetime, V5 allows specification of

- Start time
- End time
- Renewal time (long-term tickets must be renewed periodically)

This added flexibility by providing features like postdated tickets

Delegation

Request a TGT for different machine(s) and/or times

TGT can be set up to allow forwarding (converted for use by a third party) and proxying (use on a machine other than the one in the TGT)

Delegation is a security hole, Kerberos makes it optional

TGT's are marked as forwarded or proxied to allow them to be rejected

Realms

V4 required a KDC to be registered in every other realm

V5 (tries to) fix the problem with rogue KDC chaining by including in the ticket all transited realms

(client →) foo.com → bar.com (→ server)

vs

(client →) foo.com → hacker.com → bar.com (→ server)

Realms (ctd)

Requires trusting KDC's

- Trusted chain only goes back one level
- Current KDC can alter ID of previous KDC's

Kerberos abdicates responbility for trust in chaining to application developers (who will probably get it wrong)

When chaining, use the shortest path possible to limit the number of KDC's in the path which must be trusted

Other Changes in V5

V4 used DES, V5 added MD4 and MD5

V4 allowed offline password-guessing attacks on TGT's

- Request a TGT for the victim
- Try passwords on the TGT

V5 adds pre-authentication data to the TGT request

Kerberos-like Systems

KryptoKnight (IBM)

- Closer to V4 than V5
- Can use random challenges instead of synchronised clocks
- Either party can contact the KDC (in Kerberos, only the initiator can do this)
- Encryption is CDMF (40-bit DES)

Kerberos-like Systems (ctd)

SESAME (EU)

- European Kerberos clone mostly done by ICL, Bull, and Siemens
  - Uses XOR instead of DES
  - XOR in CBC mode cancels out 50% of the “encryption”
  - Keys are generated from the current system time
  - Only the first 8 bytes of data are authenticated
- Apparently users were expected to find all the holes and plug in their own secure code
- Later versions added public-key encryption support
- Vendor-specific versions provided enhanced security services

Kerberos-like Systems (ctd)

DCE (OSF)

- Distributed Computing Environment uses Kerberos V5 as a security component
- DCE adds privilege and registration servers to the Kerberos KDC
  - Privilege server provides a universal unique user ID and group ID (Kerberos uses system-specific names and ID’s)
  - Registration server provides the database for the KDC and privilege server
- DCE security is based on ACL’s (access control lists) for users and groups
- Data exchanges are protected via native DCE RPC security

Authentication Tokens

Physical device used to authenticate owner to server

Two main types

- Challenge-response calculators
- One-way authentication data generators
  - Non-challenge-response nature fits the “enter name and password” authentication model

Authentication Tokens (ctd)

SecurID

- Uses clock synchronised with server
- Token encrypts the time, sent to server in place of password
  - 64-bit key (seed), 64-bit time, produces 6-8 digit output (cardcode)
  - Card can be protected by 4-8 digit PIN which is added to the cardcode
- Server does the same and compares the result
- Timestamp provides automatic replay protection, but needs to be compensated for clock drift
- Proprietary ACE server protocol will be replaced by RSA-based SecurSight in early ‘99

Authentication Tokens (ctd)

Challenge-response calculators

- Encrypt a challenge from the server and return result to server
- Server does the same and compares the result
- Encryption is usually DES
- Encryption key is random (rather than a fixed password) which makes offline password guessing much harder

Authentication Tokens (ctd)

Possible attacks

- Wait for all but the last character to be entered, then sieze the link
- Hijack the session after the user is authenticated

However, any of these are still vastly more secure than a straight password

- It’s very difficult to choose an easy-to-guess password when it’s generated by crypto hardware
- It’s very difficult to leak a password when it’s sealed inside a hardware token

S/Key

One of a class of software tokens/one-time-password (OTP) systems

Freely available for many OS's,
`http://www.yak.net/skey/`

Uses a one-way hash function to create one-time passwords

- pass3 = hash(hash(hash(password)))
- pass2 = hash(hash(password))
- pass1 = hash(password)
  - Actual hash includes a server-specific salt to tie it to a server

Each hash value is used only once

- Server stores value *n*, verifies that hash(*n*-1) = *n*

S/Key (ctd)

Knowing hash(hash(password)) doesn't reveal hash(password)

Values are transmitted as 16 hex digits or 6-word phrases

Later refinements added new algorithms, more rigorous definitions of the protocol

OPIE

One-time Passwords in Everything, developed by (US) Naval Research Laboratory

Freely available for many OS's, `ftp://ftp.nrl.navy.mil/pub/security/opie/`

Enhancement of S/Key with name change to avoid trademark problems

PPP PAP/CHAP

Simplest PPP authentication is PAP, password authentication protocol

- Plaintext user name + password

Challenge handshake protocol was created to fix PAP

- Standard challenge/response protocol using a hash of challenge and shared secret

Other PAP Variants

SPAP

- Shiva {Propietary|Password} Authentication Protocol
- PAP with a few added bells and whistles

ARAP

- Appletalk Remote Access Protocol
- Bidirectional challenge/response using DES
  - Authenticates client to server, server to client

MSCHAP

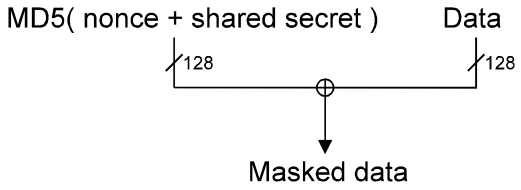
- Microsoft CHAP
- DES-encrypts 8-byte challenge using LMHASH/NTHASH
- Server stores the hash rather than the plaintext password
- Subject to the usual LMHASH attacks

RADIUS

Remote authentication for dial-in user service

Provides an authentication server for one or more clients (dial-in hosts)

Client communicates with RADIUS server via encrypted communications using a shared secret key



RADIUS (ctd)

RADIUS protocol:

- Client forwards user access request to RADIUS server
- Server replies with
  - Reject access
  - Allow access (based on password)
  - Challenge (for challenge-response protocol, eg CHAP)
- If challenge-response is used, client forwards challenge to user, user sends response to client, which forwards it to server

One RADIUS server may consult another (acting as a client)

TACACS/XTACACS/TACACS+

Based on obscure ARPANET access control system for terminal servers, later documented and extended by Cisco

- Forwards username and password to TACACS server, returns authorisation response

XTACACS, Extended TACACS

- Adds support for multiple TACACS servers, logging, extended authorisation
- Can independantly authorise access via PPP, SLIP, telnet

TACACS/XTACACS/TACACS+ (ctd)

TACACS+

- Separation of authentication, authorisation, and accounting functions with extended functionality
- Password information is encrypted using RADIUS-style encryption
- Password forwarding allows use of one password for multiple protocols (PAP, CHAP, telnet)
- Extensive accounting support (connect time, location, duration, protocol, bytes sent and received, connect status updates, etc)
- Control over user attributes (assigned IP addresse(s), connection timeout, etc)

Sorting out the *xxxxx*S's

RADIUS, TACACS = Combined authentication and authorisation process

XTACACS = Authentication, authorisation, and accounting separated

TACACS+ = XTACACS with extra attribute control and accounting

ANSI X9.26

Sign-on authentication standard ("Financial institution sign-on authentication for wholesale financial transmission")

DES-based challenge-response protocol

- Server sends challenge (TVP = time variant parameter)
- Client responds with encrypted authentication information (PAI = personal authentication information) XOR'd with TVP

Offline attacks prevented using secret PAI

Variants include

- Two-way authentication of client and server
- Authentication using per-user or per-node keys but no PAI

Public-key-based Authentication

Simple PKC-based challenge/response protocol

- Server sends challenge
- Client signs challenge and returns it
- Server verifies clients signature on the challenge

Vulnerable to chosen-protocol attacks

- Server can have client sign anything
- Algorithm-specific attacks (eg RSA signature/encryption duality)

FIPS 196

Entity authentication using public key cryptography

Extends and clarifies ISO 9798 entity authentication standard

Signed challenge/response protocol:

- Server sends server nonce SN
- Client generates client nonce CN
- Client signs SN and CN and returns to server
- Server verifies signature on the data

FIPS 196 (ctd)

Mutual authentication uses a three-pass protocol

- Server sends client signed SC as final step

Inclusion of CN prevents the previous chosen-protocol attacks

- Vulnerable to other attacks unless special precautions are taken

Biometrics

Capture data via cameras or microphones

Verification is relatively easy, identification is very hard

Fingerprints

- Small and inexpensive
- ~10% of people are difficult or impossible to scan
- Associated with criminal identification

Voice authentication

- Upset by background noise, illness, stress, intoxication
- Can be used over phone lines

Eye scans

- Intrusive (scan blood vessels in retina/patterns in iris)

Biometrics (ctd)

Advantages

- Everyone carries their ID on them
- Very hard to forge
- Easy to use

Disadvantages

- You can’t change your password
- Expensive
- No real standards (half a dozen conflicting ones as well as vendor-specific formats)
- User acceptance problems

PAM

Pluggable Authentication Modules

OSF-designed interface for authentication/identification plugins

Administrator can configure authentication for each application

Service	Module
`login`	`pam_unix.so`
`ftp`	`pam_skey.so`
`telnet`	`pam_smartcard.so`
`su`	`pam_securid.so`

PAM (ctd)

Modules are accessed using a standardised interface

```
pam_start( &handle );
pam_authenticate( handle );
pam_end( handle );
```

Modules can be stacked to provide single sign-on

- User is authenticated by multiple modules at sign-on
  - Avoids the need to manually invoke each sign-on service (`kinit`, `dce_login`, `dtlogin`, etc)
- Password mapping allows a single master password to encrypt per-module passwords

PAM in Practice

A typical implementation is Linux-PAM

- Extended standard login (checks time, source of login, etc)
- `.rhosts`, `/etc/shells`
- cracklib (for password checking)
- DES challenge/response
- Kerberos
- S/Key, OPIE
- RADIUS
- SecurID

Electronic Commerce

SET is the answer, but you have to phrase the question very carefully

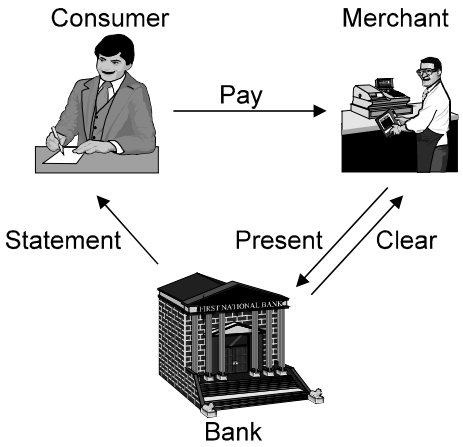
Electronic Payments

An electronic payment system needs to be

- Widely recognised
- Hard to fake
- Hold its value
- Convenient to use
- Anonymous/not anonymous

Convenience is the most important point

Cheques



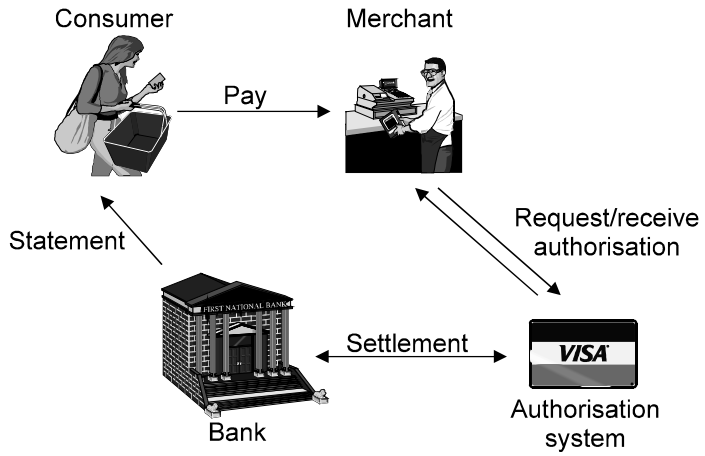
Merchant doesn't know whether the cheque is valid until it's cleared

Cheques (ctd)

Consumer can't detect fraud until the statement arrives

Cost of processing errors vastly outweighs the cost of normal actions

Credit Cards



Authentication is online

Settlement is usually offline (batch processed at end of day)

Credit Cards (ctd)

Consumer can't detect fraud until the statement arrives

Cost of processing errors vastly outweighs the cost of normal actions

Merchant carries the risk of fraud in card not present transactions

Consumer liability is limited to $50

Far more merchant fraud than consumer fraud

Credit card companies assume liability for their merchants; banks with cheques don't

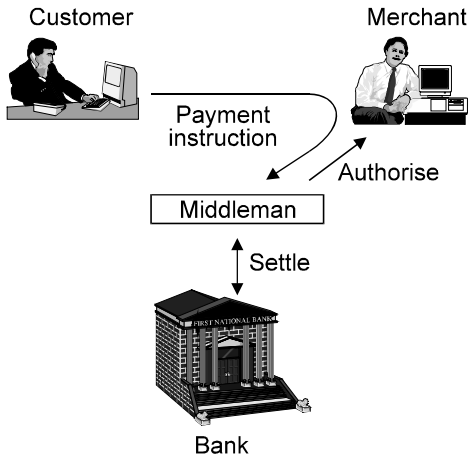
Transactions on the Internet

Transactions are fairly conventional card not present transactions and follow the precedent set by phone orders

Online nature provides instant verification

Biggest problems are authentication and confidentiality

General Model of Internet Transactions



Virtually all net payment systems consist of some variant of this

Everyone wants to be the middleman

Retail vs Business-to-business Commerce

Retail commerce

- Small dollar amounts
- Stranger-to-stranger transactions

Business-to-business commerce

- Large dollar amounts
- Based on trust relationships
- Banks play a direct role — they guarantee the transaction
  - You can't disintermediate the banks

Business-to-business commerce is where the money is

- For retail transactions, you can't beat a credit card over SSL

Business customers will buy to reduce current costs

Payment Systems

Book entry systems

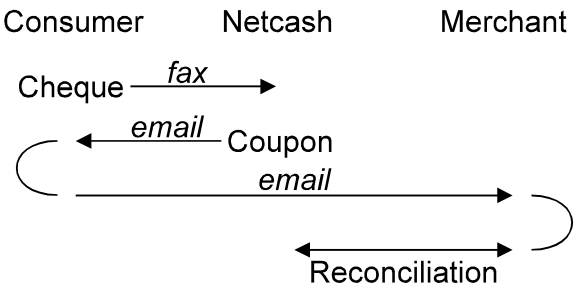
- Credit cards over SSL
- Encrypted credit cards (Cybercash)
- Virtual credit cards (First Virtual)
- e-cheques (Netcash)
- Mondex/SET
- Many, many others

Bearer certificate systems

- Scrip (Millicent)
- True digital cash (Digicash)

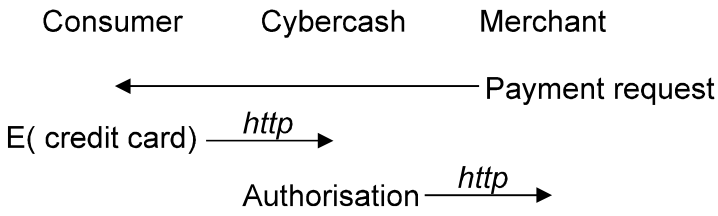
Netcash

e-cheques, `http://www.teleport.com/~netcash`



Cybercash

Encrypted credit cards, `http://www.cybercash.com`



Book Entry System Variations

Some systems (eg GlobeID) have the consumer (instead of the merchant) do the messaging

Credit cards don't handle small transactions very well. Some options are

- Don't handle micropayments at all
- Middleman has to act as a bank
- Use a betting protocol: 10 cent transaction = 1% chance of a $10 transaction

Digicash

Digicash issuing protocol

User		Bank (mint)
blind(note)	$\rightarrow$	
	$\leftarrow$	sign(blind(note))
unblind(sign(blind(note))) = sign(note)		

User ends up with a note signed by the bank

- Note is not tied to user
- Implemented as an electronic purse which holds arbitrary denominations

Digicash (ctd)

Using e-cash

- Send note to merchant
- Merchant redeems note at bank
- Double spending is avoided by having the user ID revealed if the note is banked twice (ZKP)

Problems

- Banks don't like it (anyone can be a bank)
- Governments don't like it
- Not used much (awkward/fluctuating licensing requirements)
  - Licensed as if it were an RSA-style monopoly patent

By the time they figure it out, the patent will expire (2007)

Making e-cash work

Best e-cash business model is to earn seignorage by selling it

- Bank earns interest on real cash corresponding to digital bits held by consumer
- US Federal Reserve earns $20B/year in interest on outstanding dollar bills
- Phone cards and gift vouchers are a small-scale example of this

Consumers may demand interest on e-cash

e-cash is useful for small transactions (micropayments) which other systems can't handle

- But what do you buy over the net for 10 cents?

SET

Secure Electronic Transactions

Based on two earlier protocols, STT (VISA/Microsoft) and SEPP (MasterCard/IBM)

STT

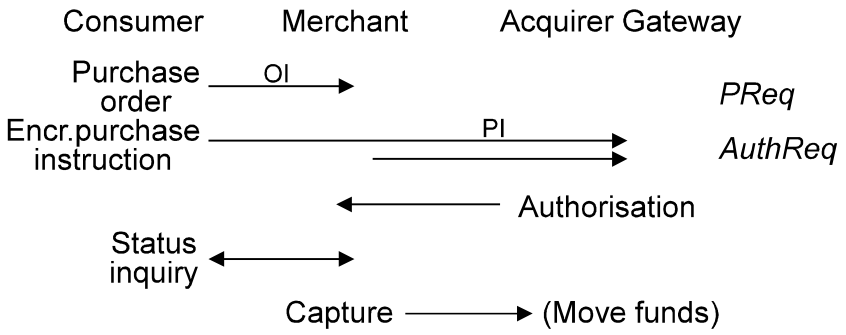
- One component of a larger architecture
- Provision for strong encryption
- Completely new system
- More carefully thought out from a security standpoint

SET (ctd)

SEPP

- General architectural design rather than a precise specification
- Lowest-common-denominator crypto
- Fits in with existing infrastructure
- More politically and commercially astute

SET (ctd)



Acquirer gateway is an Internet interface to the established credit card authorisation system and cardholder/merchant banks

SET Features

Card details are never disclosed to merchant

- Encrypted purchase instruction (PI) can only be decrypted by the acquirer
- PI is cryptographically tied to the order instruction (OI) processed by the merchant
- Clients digital signature protects the merchant from client repudiation

Authorisation request includes the consumer PI and merchant equivalent of the PI

- Acquirer can confirm that the cardholder and merchant agree on the purchase details

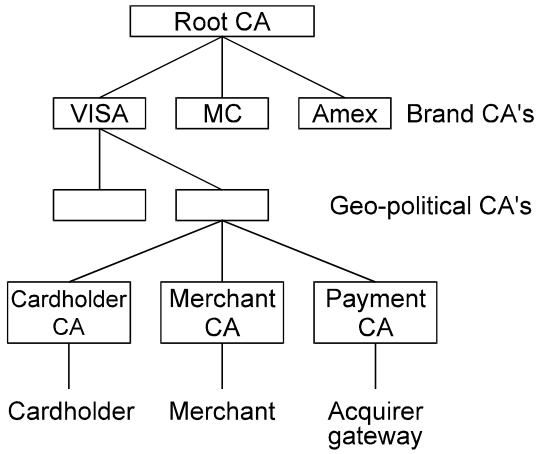
SET Features (ctd)

Capture can take place later (eg when the goods are shipped)

- User can perform an inquiry transaction to check the status

The whole SET protocol is vastly more complex than this

SET Certification



SET root CA and brand CA’s are rarely utilised and have very high security

SET Certification (ctd)

SET includes a complete PKI using customised X.509

- Online certificate requests
- Certificate distribution
- Certificate revocation

SET certificates are implemented as an X.509 profile with SET-specific extensions

SET Certification (ctd)

Card-based infrastructure makes certificate management (relatively) easy

- Users are identified by their cards
- Certificates are revoked by cancelling the card
- Because everything is done online, “certificate management” is easy
- Acquirer gateways have long-term signature keys and short-term encryption keys
  - Encryption keys can be revoked by letting them expire

SET in Practice: Advantages

SET will enable e-commerce, eliminate world hunger, and close the ozone hole

- SET prevents fraud in card not present transactions

SET eliminates the need for a middleman (the banks love this)

SET leverages the existing infrastructure

SET in Practice: Problems

SET is the most complex (published) crypto protocol ever designed

- > 3000 lines of ASN.1 specification
- 28-stage (!) transaction process
  - "The SET reference implementation will be available by mid 1996"
  - "SET 1.0 " " " mid 1997"
  - "SET 2.0 " " " mid 1998"
- Interoperability across different implementations is a problem

SET is awfully slow (6 RSA operations per transaction)

- Great for crypto hardware accelerator manufacturers
- For comparison, VISA interchange gateway currently has to handle 2000 pure DES-based transactions/second

SET in Practice: Problems (ctd)

Although SET was specifically designed for exportability, you still can't export the reference implementation

SET requires

- Custom wallet software on the cardholders PC
- Custom merchant software
- Special transaction processing software (and hardware) at the acquirer gateway.

Practical Issues

> Of course my password is the same as my pet's name
>
> My macaw's name was Q47pY!3 and I change it every 90 days
>
> — Trevor Linton

Practical Issues

Strong, effectively unbreakable crypto is universally available (despite US government efforts)

- Don't attack the crypto, attack the infrastructure in which it's used
- " " " " implementation
- " " " " users

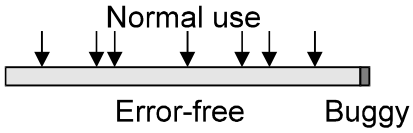
Many infrastructure/implementation details are treated as black boxes by developers

- Storage protection/sanitisation
- Long-term secret storage
- Key generation

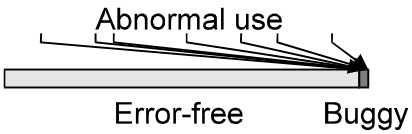
Why Security is Harder than it Looks

All software has bugs

Under normal usage conditions, a 99.99% bug-free program will rarely cause problems



A 99.99% security-bug-free program can be exploited by ensuring the 0.01% instance is always encountered



This converts the 0.01% failure to 100% failure

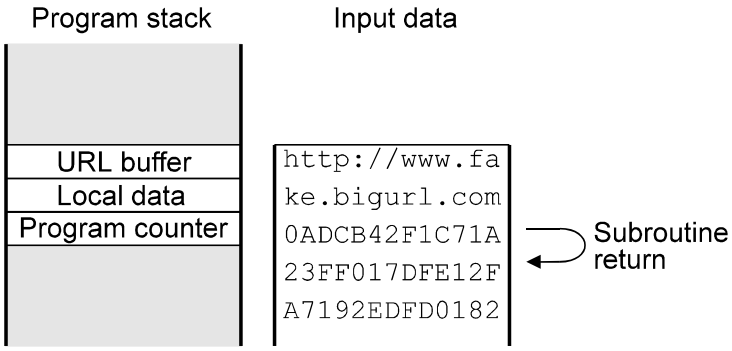
Buffer Overflows

In the last year or two these have appeared in

splitvt, syslog, mount/umount, sendmail, lpr, bind, gethostbyname(), modstat, cron, login, sendmail again, the query CGI script, newgrp, AutoSofts RTS inventory control system, host, talkd, getopt(), sendmail yet again, FreeBSD's crt0.c, WebSite 1.1, rlogin, term, ffbconfig, libX11, passwd/yppasswd/nispasswd, imapd, ipop3d, SuperProbe, lpd, xterm, eject, lpd again, host, mount, the NLS library, xlock, libXt and further X11R6 libraries, talkd, fdformat, eject, elm, cxterm, ps, fbconfig, metamail, dtterm, df, an entire range of SGI programs, ps again, chkey, libX11, suidperl, libXt again, lquerylv, getopt() again, dtaction, at, libDtSvc, eeprom, lpr yet again, smbmount, xlock yet again, MH-6.83, NIS+, ordist, xlock again, ps again, bash, rdist, login/scheme, libX11 again, sendmail for Windows NT, wm, wwwcount, tgetent(), xdat, termcap, portmir, writesrv, rcp, opengroup, telnetd, rlogin, MSIE, eject, df, statd, at again, rlogin again, rsh, ping, traceroute, Cisco 7xx routers, xscreensaver, passwd, deliver, cidentd, Xserver, the Yapp conferencing server, multiple problems in the Windows95/NT NTFTP client, the Windows War and Serv-U FTP daemon, the Linux dynamic linker, filter (part of elm-2.4), the IMail POP3 server for NT, pset, rpc.nisd, Samba server, ufsrestore, DCE secd, pine, dslip, Real Player, SLMail, socks5, CSM Proxy, imapd (again), Outlook Express, Netscape Mail, mutt, MSIE, Lotus Notes, MSIE again, libauth, login, iwsh, permissions, unfsd, Minicom, nslookup, zpop, dig, WebCam32, smbclient, compress, elvis, lha, bash, jidentd, Tooltalk, ttdbserver, dbadmin, zgv, mountd, pcnfs, Novell Groupwise, mscreen, xterm, Xaw library, Cisco IOS, mutt again, ospf_monitor, sdtcm_convert, Netscape (all versions), mpg123, Xprt, klogd, catdoc, junkbuster, SerialPOP, and rdist

Buffer Overflows (ctd)

Typical case: Long URL's



- Data at the end of the URL overwrites the program counter/return address
- When the subroutines returns, it jumps to the attackers code

Fixing Overflow Problems

More careful programming

- Isolate security functionality into carefully-checked code

Make the stack non-executable

Compiler-based solutions

- Build bounds checking into the code (very slow)
- Build stack checking into the code (slight slowdown)
- Rearrange stack variables (no slowdown)

Storage Protection

Sensitive data is routinely stored in RAM, but

- RAM can be swapped to disk at any moment
  - Users of one commercial product found multiple copies of their encryption password in the Windows swap file
  - "Suspend to disk" feature in laptops is particularly troublesome
- Other processes may be able to read it from memory
- Data can be recovered from RAM after power is removed

Protecting Memory

Locking sensitive data into memory isn't easy

- Unix: `mlock()` usable by superuser only
- Win16: No security
- Win95/98: `VirtualLock()` does nothing
- WinNT: `VirtualLock()` doesn't work as advertised (data is still swapped)
- Macintosh: `HoldMemory()`

Scan memory for data:

```
VirtualQueryEx()
VirtualUnprotectEx()
ReadProcessMemory()
```

Protecting Memory (ctd)

Create DIY swapfile using memory-mapped files

- Memory is swapped to a known file rather than system swapfile
- File is wiped after use

Problems:

- Truly erasing disk data is impossible
- Data isn’t wiped on system crash/power loss

Protecting Memory (ctd)

Force memory to remain in use at all times

- Background thread touches memory periodically

Allocate non-pageable memory

- Requires a kernel driver
- Mapping memory from kernel to user address space is difficult

Storage Sanitisation

Problems in erasing disk data

- Defect management systems move/remap data, making it inaccesible through normal means
- Journaling filesystems retain older data over long periods of time
- Online compression schemes compress fixed overwrite patterns to nothing, leaving the target data intact
- Disk cacheing will discard overwrites if the file is unlinked immediately afterwards (Win95/98, WinNT)
  - Many Windows file-wipers are caught by this

Recovering Data

One or two passes can be easily recovered by “error cancelling”

- Read actual (digital) data
- Read raw analog signal
- Subtract expected signal due to data from actual analog signal
- Result is previous (overwritten) data

US government standard (DoD 5200.28) with fixed patterns (all 0’s, all 1’s, alternating 0’s and 1’s) is particularly bad

Design overwrite patterns to match HD encoding methods

Advanced Data Recovery

Ferrofluid + optical microscopes

- Defeated by modern high-density storage systems

Scanning probe microscopes overcame this problem

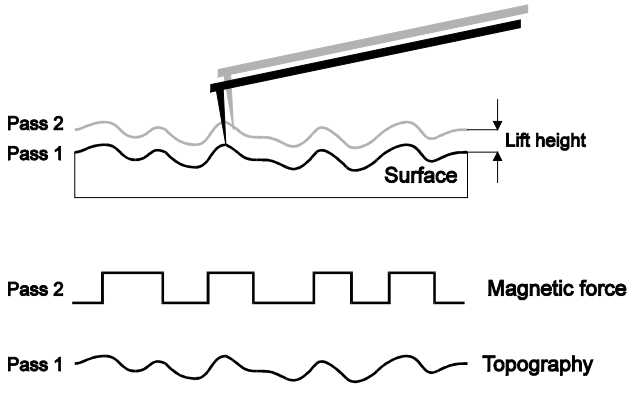
- Oscillating probe is scanned across the surface of the object
- Change in probe movement measured by laser interferometer

Can be built for a few thousand dollars

Commercial ones specifically set up for disk platter analysis are available

Advanced Data Recovery (ctd)

Magnetic force microscope (MFM)



1. Read disk topography
2. Read magnetic force (adjusted for topography)

Advanced Data Recovery (ctd)

MFM's can be used as expensive read channels, but can do far more

- Erase bands (partially-overwritten data at the edges) retain previous track images
- Overwriting one set of data with another causes track width modulation
- Erased/degaussed drives can often still be read with an MFM
  - Modern high-density media can't be effectively degaussed with commercial tools

Advanced Data Recovery (ctd)

Recommendations

- Use the smallest, highest-density drives possible
- If data is sensitive, destroy the media
  - Where does your returned-under-warranty drive end up?
  - For file servers, business data, always destroy the media (there's always something sensitive on there)

Recovering Memory Data

Electrical stress causes ion migration in DRAM cells

Data can be recovered using special (undocumented) test modes which measure changes in cell thresholds

- At room temperature, decay can take minutes or hours
- At cryogenic temperatures, decay can take weeks? months?

A quick overwrite doesn't help much

Solution is to only store data for short periods

- Relocate data periodically
- Toggle bits in memory

Random Number Generation

Key generation requires large quantities of unpredictable random numbers

- Very difficult to produce on a PC
- Most behaviour is predictable
- User input can be unpredictable, but isn't available on a standalone server

Many implementations leave it to application developers (who invariably get it wrong)

Bad RNG's

Netscape

```
a = mixbits( time.tv_usec );
b = mixbits( getpid() + time.tv_sec + ( getppid() <<
  12 );
seed = MD5( a, b );

nonce = MD5( seed++ );
key = MD5( seed++ );
```

Kerberos V4

```
srandom( time.tv_usec ^ time.tv_sec ^ getpid() ^
  gethostid() ^ counter++ );
key = random();
```

Bad RNG's (ctd)

MIT_MAGIC_COOKIE

```
key = rand() % 256;
```

SESAME

```
key = rand();
```

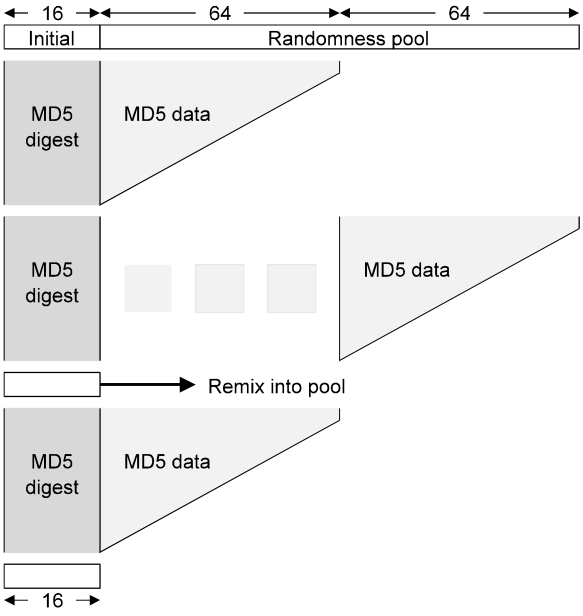
Types of Generator

Generator consists of two parts

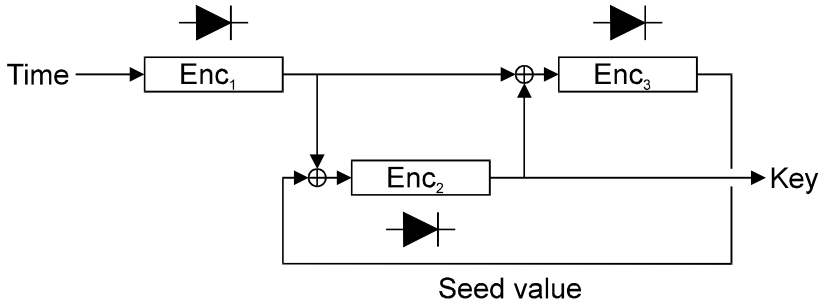
- Polling mechanism to gather random data
- Pseudo-random number generator (PRNG) to "stretch" the output

Physical source	Various hardware generators, Hotbits (radioactive decay), Lavarand
Physical source + postprocessing	SG100
Multi-source polling	SKIP, cryptlib
Single-source polling	PGP 2.x, PGP 5.x, `/dev/random`
Secret nonce + PRNG	Applied Cryptography, BSAFE
Secret fixed value + PRNG	ANSI X9.17
Known value + PRNG	Netscape, Kerberos V4, Sesame, and many more

Example: Unix `/dev/random`



Example: ANSI X9.17



Relies on strength of triple DES encryption and supplied encryption key

Randomness Sources

- Process and thread information
- Mouse and keyboard activity
- Memory and disk usage statistics
- System timers
- Network statistics
- GUI-related information

Run periodic background polls of sources

Try and estimate the randomness available, if insufficient

- Perform further polling
- Inform the user

Effectiveness of the Randomness Source

Effects of configuation

- Minimal PC hardware (one HD, one CD) produces half the randomness of maximum PC hardware (multiple HD's, CD, network card, SCSI HD and CD)

Effects of system load and usage

- Statistics change little over time on an unloaded machine
- A reboot drastically affects the system state
  - Reboot the machine after generating a high-value key

TEMPEST

Sometimes claimed to stand for Transient Electromagnetic Pulse Emission Standard

Known since the 1950's, but first publicised by van Eck in 1985

- Provided details on remote viewing of computer monitors
- Required about $15 worth of parts (for sync recovery)
- The spooks were not happy

TEMPEST Principles

Fast-rise pulses lead to harmonics radiated from semiconductor junctions

- Used to detect bugs
  - – Flood the room with microwaves
  - – Watch for radiated responses

Anything which carries a current acts as an antenna

TEMPEST monitoring gear receives and interprets this information

TEMPEST Sources

Computer monitor/laptop screen

- Generally radiates huge amounts of signal (range of hundreds of metres)
- Most signal is radiated to the sides, little to the front and back
- Requires external horizontal/vertical sync insertion, since sync frequencies are too low to be radiated
- Individual monitors can be picked out even when other similar monitors are in use
- Jamming is often ineffective for protection
  - – Eavesdroppers can still zero in on a particular monitor

TEMPEST Sources (ctd)

Keyboard

- Some keyboards produce distinct RF signatures for each key pressed
- Active monitoring
  - Beam RF energy at the keyboard cable
  - Reflected signal is modulated by abscence/presence of electrical current

Ethernet

- UTP can be intercepted over some distance

TEMPEST Sources (ctd)

Printer and serial cables

Leakage into power lines

Coupling into power lines, phone lines, metal pipes

- Further radiation from there

Surface waves on coax lines

TEMPEST Protection

Extremely difficult to protect against

Stopping it entirely

- Extreme amounts of shielding on all equipment
- Run the equipment inside a Faraday cage

Stopping it partially

- FCC Class B computers and equipment
- RF filters on power lines, phone lines
- Shielded cables
- Ferrite toroids around cables to attenuate surface waves
- Radio hams have information on safely operating computers near sensitive comms gear

Use a portable radio as a simple radiation tester

Snake Oil Cryptography

Named after magic cure-all elixirs sold by travelling medicine salesmen

Many crypto products are sold using similar techniques

- The crypto has similar effectiveness
- This is so common that there's a special term, "snake oil crypto", to describe it

Snake Oil Warning Signs

Security through obscurity

- "Trust me, I know what I'm doing"
  - They usually don't
- Most security through obscurity schemes are eventually broken
  - Once someone finds out what your secret security system is, it's no longera secret and no longer secure
  - It's very hard to keep a secret on the net

Proprietary algorithms and revolutionary breakthroughs

- "I know more about algorithm design than the entire world's cryptographers"
- Common snake oil warning signs are use of cellular automata, neural nets, genetic algorithms, and chaos theory
- See "security through obscurity"

Snake Oil Warning Signs (ctd)

Unbreakability

- Usually claimed by equating the product to a one-time-pad
- Product isn't a one-time-pad, and therefore not unbreakable

"Military-grade crypto"

- Completely meaningless term (cf "military-grade spreadsheet")
  - Military tends to use hardware, civilians use software
  - Prefer shift-register based stream ciphers, everyone else uses block ciphers
  - Keys are generally symmetric and centrally managed, everyone else uses distributed PKC keys
- Products should therefore be advertised as "nothing like military-grade crypto"

Snake Oil Warning Signs (ctd)

Technobabble

- Use of terms unknown to anyone else in the industry

Used by *xyz*

- Every product, no matter how bad, will gain at least one big-name reference customer

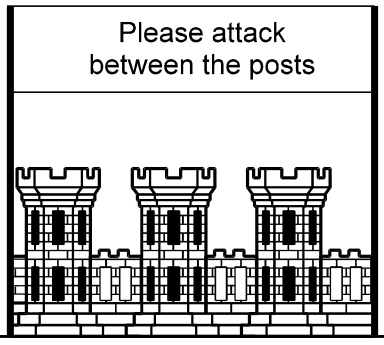
Exportable from the US

- Except for special-purpose cases (eg SGC), the US government will not allow the export of anything which provides real security
- If it's freely exportable, it's broken

Snake Oil Warning Signs (ctd)

Security challenges

- Generally set up to make it impossible to succeed



- These things always get the media's attention, especially if the reward is huge (chance of press coverage = 20% per zero after the first digit)

Snake Oil Warning Signs (ctd)

Would you buy this product?

- "Our unbreakable military-grade bi-gaussian cryptography, using a proprietary one-time-pad algorithm, has recently been adopted by a Fortune 500 customer and is available for use inside and outside the US"

Badly marketed good crypto is indistinguishable from snake oil

- If you're selling a crypto product, be careful how your marketing people handle it
  - If left to their own devices, they'll probably sell it as snake oil

Snake Oil Case Study

Meganet Virtual Matrix Encryption

- "A new kind of encryption and a new algorithm, different from any existing method"
- "By copying the data to a random built-in Virtual Matrix, a system of pointers is being created. ..."
- "The worlds first and only unbreakable crypto"
- "We challenged the top 250 companies in the US to break our products. None succeeded"
  - They don't even know Meganet exists
- "55,000 people tried to break our product"
  - 55,000 visited their web page
- "Working on standardising VME with the different standards committees"

Snake Oil Case Study (ctd)

Challenged large companies to break their unbreakable crypto

- Enumerate each company in the PR to ensure that their name is associated with large, publicly held stocks

Used accounts at organisations like BusinessWire and PRNewswire to inject bogus press releases into newswires

- Run anything at $500 for 400 words
- Claimed IBM was so impressed with their product that they were recommending it for the AES
  - IBM had never heard of them

Snake Oil (ctd)

Big-name companies sell snake oil too

Tools exist to recover passwords for

- Adobe Acrobat
- ACIUS 4th Dimension
- Arj archives
- Claris Filemaker Pro
- CompuServe WinCim
- dBASE
- Diet compressed files
- Eudora
- Lotus Ami-Pro
- Lotus 1-2-3
- Lotus Organiser
- Lotus Symphony

Continues

Snake Oil (ctd)

Continued

- LZEXE compressed files
- MS Access
- MS Excel
- MS Mail
- MS Money
- MS Scheduler
- MS Word
- MYOB
- Paradox
- Pegasus
- Pklite compressed files
- Pkzip archives
- Q&A Database

Continues

Snake Oil (ctd)

Continued

- Quattro Pro
- QuickBooks
- Quicken
- Stacker
- Symantec Act
- Trumpet Winsock
- VBA projects
- WinCrypt
- Windows 3.1/95 password
- Windows NT passwords
- WordPerfect
- WS FTP

... and many, many more

Selling Security

Security doesn't sell well to management

Many security systems are designed to show due diligence or to shift blame

- Crypto/security evidence from these systems is very easy to challenge in court

You get no credit if it works, and all the blame if it doesn't

To ensure good security, insurance firms should tie premiums to security measures

- Unfortunately, there's no way to financially measure the effectiveness of a security system

Selling Security to Management

Regulatory issues

- Liability for negligence (poor security/weak crypto)
- Shareholders could sue the company if share price drops due to security breach
- US companies spend more on security due to litigation threats

Privacy/data protection requirements

Media stories of hacker/criminal attacks on systems

The best security customers

- Have just been publicly embarassed
- Are facing an audit

Miscellaneous Topics

Buy a rifle, encrypt your data, and wait for the revolution

Smart Cards

Invented in the early 1970's

Technology became viable in early 1980's

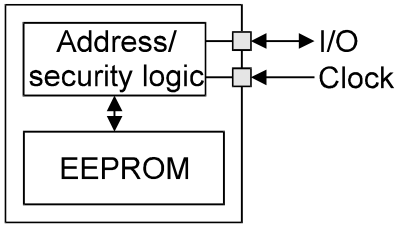
Major use is prepaid telephone cards (hundreds of millions)

- Use a one-way (down) counter to store card balance

Other uses

- Student ID/library cards
- Patient data
- Micropayments (bus fares, photocopying, snack food)

Memory Cards



Usually based on I^2C (serial memory) bus

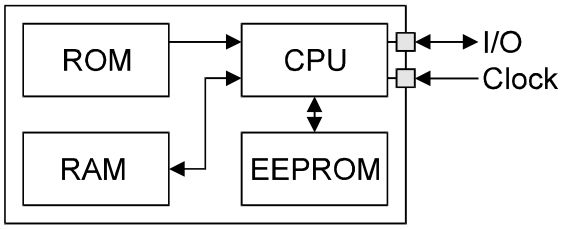
Typical capacity: 256 bytes

EEPROM capabilities

- Nonvolatile storage
- 10,000 write/erase cycles
- 10ms to write a cell or group of cells

Cost: $5

Microprocessor Cards



ROM/RAM contains card operating system and working storage

EEPROM used for data storage

Microprocessor Cards (ctd)

Typical specifications

- 8-bit CPU
- 16K ROM
- 256 bytes RAM
- 4K EEPROM

Size ratio of memory cells:

RAM = 4× EEPROM size
= 16× ROM size

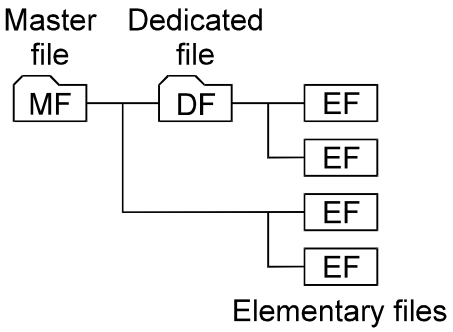
Cost: $5-50 (with crypto accelerator)

Smart Card Technology

Based on ISO 7816 standard, which defines

- Card size, contact layout, electrical characteristics
- I/O protocols
  - Byte-based
  - Block-based
- File structures

File Structures



Files addressed by 16-bit file ID (FID)

- FID is often broken into DF:EF parts (MF is always 0x3F00)

Files are generally fixed-length and fixed-format

File Types

Transparent

- Binary blob

Linear fixed

- $n \times$ fixed-length records

Linear variable

- n records of fixed (but different) lengths

Cyclic

- Linear fixed, oldest record gets overwritten

Execute

- Special case of transparent file

File Attributes

EEPROM has special requirements (slow write, limited number of write cycles) which are supported by card attributes

- WORM, only written once
- Multiple write, uses redundant cells to recover when some cells die
- Error detection/correction capabilities for high-value data
- Error recovery, ensures atomic file writes
  - Power can be removed at any point
  - Requires complex buffering and state handling

Card Commands

Typical commands are

- CREATE/SELECT/DELETE FILE
- READ/WRITE/UPDATE BINARY
  - Write can only change bits from 1 to 0, update is a genuine write
- ERASE BINARY
- READ/WRITE/UPDATE RECORD
- APPEND RECORD
- INCREASE/DECREASE
  - Changes cyclic file position

Card Commands (ctd)

Access control

- Based on PIN of chip holder verification (CHV)
- VERIFY CHV
- CHANGE CHV
- UNBLOCK CHV
- ENABLE/DISABLE CHV

Authentication

- Simple challenge/response authentication protocol
- INTERNAL AUTHENTICATE
  - Authenticate card to terminal
- EXTERNAL AUTHENTICATE
  - Authenticate terminal to card

Card Commands (ctd)

Encryption: Various functions, typically

- ENCRYPT/DECRYPT
- SIGN DATA/VERIFY SIGNATURE

Electronic purse instructions

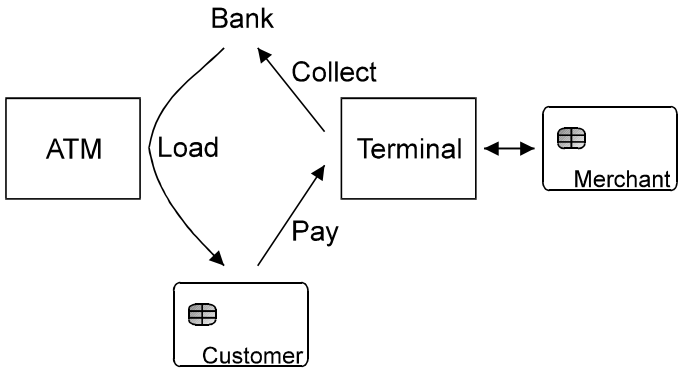
- INITIALISE/CREDIT/DEBIT

Application-specific instructions

- RUN GSM ALGORITHM

prEN 1546

Inter-sector electronic purse (IEP) standard, 1995



Both customer and merchant use smart-card based electronic purses to handle payment transactions

prEN 1546 (ctd)

Defines the overall framework in some detail, but leaves algorithms, payment types and parameters, and other details to implementors

- Specifies the file layout and data elements for the IEP
- Defines commands INITIALISE IEP, CREDIT IEP, DEBIT IEP, CONVERT IEP CURRENCY, and UPDATE IEP PARAMETER
- Specifies exact payment routines in a BASIC-like pseudolanguage
- All messages are “signed” (typically with a 4-byte DES MAC)
- Handles everything but purse-to-purse transactions

Includes many variants including a cut-down version for phonecards and extra acknowledgements for transactions

Credit IEP Transaction

IEP		Bank
INITIALISE bank for load (amount and currency)	→	Verify currency and balance
Verify details	←	INITIALISE IEP for load
Sign(DEBIT account)	→	Verify details Debit account
Verify details Update card state	←	Sign(CREDIT IEP)
Sign(Load acknowledgement)	→	Verify acknowledgement

Credit Merchant Transaction

IEP		Merchant
	←	INITIALISE IEP for purchase
Sign(INITIALISE merchant for purchase)	→	Verify details
Verify details Update card state	←	Sign(DEBIT IEP)
Sign(CREDIT Merchant)	→	Verify details Record transaction for transmission to bank
	←	Sign(Purchase acknowledgement)

Working with Cards

ISO 7816 provides only a standardised command set, implementation details are left to vendors

- Everyone does it differently

Standardised API's are slow to appear

PKCS #11 (crypto token interface) is the most common API

- Functionality is constantly changing to handle different card/vendor features
- Vendors typically only implement the portions which correspond to their products
- For any nontrivial application, custom handling is required for each card type

Working with Cards (ctd)

JavaCard

- Standard smart card with an interpreter for a Java-like language in ROM
- Card runs Java with most features (multiple data types, memory management, most class libraries, and all security (via the bytecode verifier)) stripped out
  - Can run up to 200 times slower than card native code

Provides the ability to mention both "Java" and "smart cards" in the same sales literature

Attacks on Smart Cards

Use doctored terminal/card reader

- Reuse and/or replay authentication to card
- Display $*x* transaction but debit $*y*
- Debit account multiple times

Protocol attacks

- Card security protocols are often simple and not terribly secure

Fool CPU into reading from external instead of internal ROM

Manipulating supply voltages can affect security mechanisms

- Picbuster
- Clock/power glitches can affect execution of instructions

Attacks on Smart Cards (ctd)

Erasing an EEPROM cell requires a high voltage (12 vs 5V) charge

- Don’t provide the power to erase cells
- Most cards now generate the voltage internally
  - Destroy the (usually large) on-chip voltage generator to ensure the memory is never erased

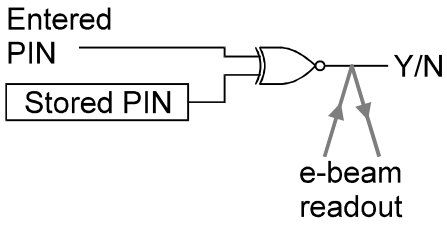
Physical Attacks

Erase onboard EPROM with UV spot beam

Remove chip from encapsulation with nitric acid

- Use microprobing to access internal circuit sections
- Use electron-beam tester to read signals from the operational circuit

Example: PIN recovery with an e-beam tester



Physical Attacks (ctd)

Modify the circuit using a focused ion beam (FIB) workstation

- Disable/bypass security circuitry (Mondex)
- Disconnect all but EEPROM and CPU read circuitry

Attacking the Random Number Generator

Generating good random data (for encryption keys) on a card is exceedingly difficult

- Self-contained, sealed environment contains very little unpredictable state

Possible attacks

- Cycle the RNG until the EEPROM locks up
- Drop the operating voltage to upset analogue-circuit RNG's
- French government attack: Force manufacturers to disable key generation
  - This was probably a blessing in disguise, since externally generated keys may be much safer to use

Timing/Power Analysis

Crypto operations in cards

- Take variable amounts of time depending on key and data bits
- Use variable amounts of power depending on key and data bits
  - Transistors are voltage-controlled switches which consume power and produce electromagnetic radiation
  - Power analysis can provide a picture of DES or RSA en/decrypt operations
  - Recovers 512-bit RSA key at ~3 bits/min on a PPro 200

Differential power analysis is even more powerful

- Many card challenge/response protocols are DES-based → apply many challenge/response operations and observe power signature

Voice Encryption

Built from three components

Hardware-based

- DSP with GSM or CELP speech compression
- DSP modem

Software-based

- GSM or CELP in software
- External modem or TCP/IP network connection

Mostly built from off-the-shelf parts (GSM DSP, modem DSP, software building blocks)

Typical Voice Encryption System

Speech compression

- GSM compression (high-bandwidth)
- CELP compression (low-bandwidth)

Security

- DH key exchange
- DES (larger manufacturers)
- 3DES, IDEA, Blowfish (smaller manufacturers, software)
- Password/PIN authentication

Typical Voice Encryption System (ctd)

Communications

- Built-in modem (hardware)
- Internet communications (software)

Speak Freely,
`http://www.fourmilab.ch/netfone/windows/speak_freely.html`

- Typical software implementation
- Uses standard software components
- Portable across several operating systems

Problems

Latency issues (dropped packets)

Authentication/MITM attacks

No standardisation

GSM

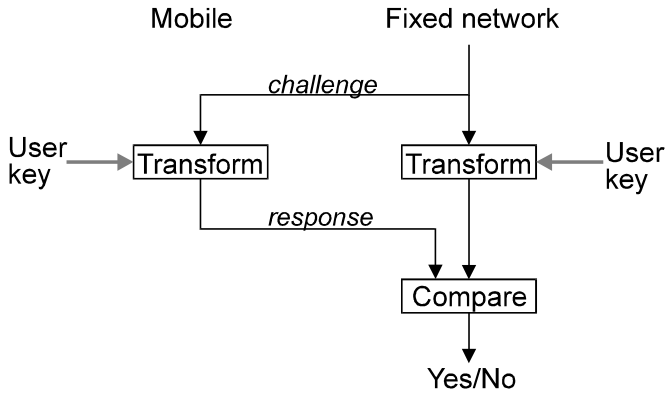
GSM subscriber identity module (SIM) contains

- International Mobile Subscriber Identity (IMSI)
- Subscriber identification key K_i

Used for authentication and encryption via simple challenge/response protocol

- A3 and A8 algorithms provide authentication (usually combined as COMP128)
- A5 provides encryption

GSM (ctd)

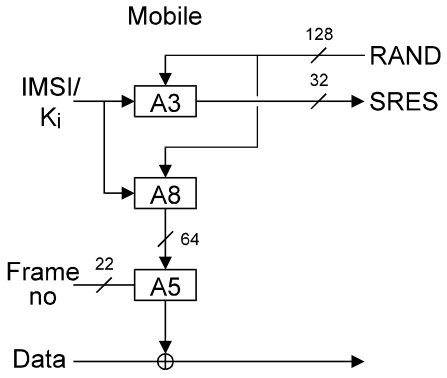


Authentication is simple challenge/response using A3 and IMSI/K_i

GSM Security

A3 used to generate response

A8 used to generate A5 key



GSM Security (ctd)

1. Base station transmits 128-bit challenge RAND
2. Mobile unit returns 32-bit signed response SRES via A3
3. RAND and K_i are combined via A8 to give a 64-bit A5 key
4. 114-bit frames are encrypted using the key and frame number as input to A5

GSM Security (ctd)

GSM security was broken in April 1998

- COMP128 is weak, allows IMSI and K_i to be extracted
  - Direct access to SIM (cellphone cloning)
  - Over-the-air queries to phone
- A5 was deliberately weakened by zeroing 10 key bits
- Claimed GSM fraud detection system doesn't seem to exist
- Affects 80 million GSM phones

GSM Security (ctd)

Key weakening was confirmed by logs from GSM base stations

```
BSSMAP GSM 08.08 Rev 3.9.2  (BSSM) HaNDover REQuest (HOREQ)
-------0 Discrimination bit D BSSMAP
0000000- Filler
00101011 Message Length    43
00010000 Message Type      0x10
Channel Type
00001011 IE Name           Channel type
00000011 IE Length         3
00000001 Speech/Data Indicator    Speech
00001000 Channel Rate/Type Full rate TCH channel Bm
00000001 Speech encoding algorithm      GSM speech algorithm
Encryption Information
00001010 IE Name           Encryption information
00001001 IE Length         9
00000010 Algorithm ID      GSM user data encryption V.1
******** Encryption  Key   C9 7F 45 7E 29 8E 08 00
Classmark Information Type 2
```

GSM Security (ctd)

Many countries were sold a weakened A5 called A5/2

- Workfactor to break A5 is ~2^{40}
- Workfactor to break A5/2 is ~2^{16}
- Much easier attack is to bypass GSM entirely and attack the base station or land lines/microwave links

Most other cellphone security systems have been broken too

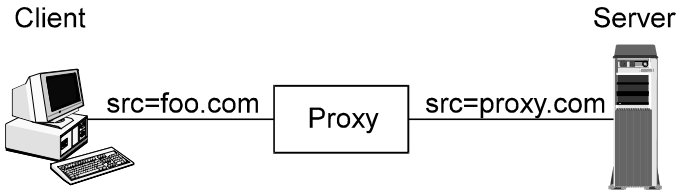
- Secret design process with no public scrutiny or external review
- Government interference to ensure poor security

Traffic Analysis

Monitors presence of communications and source/destination

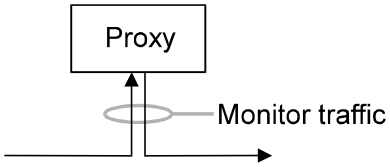
- Most common is analysis of web server logs
- Search engines reveal information on popularity of pages
- The mere presence of communications can reveal information

Simple Anonymiser Proxy



HTTP version at `http://www.anonymizer.com`

Fairly easy to defeat:

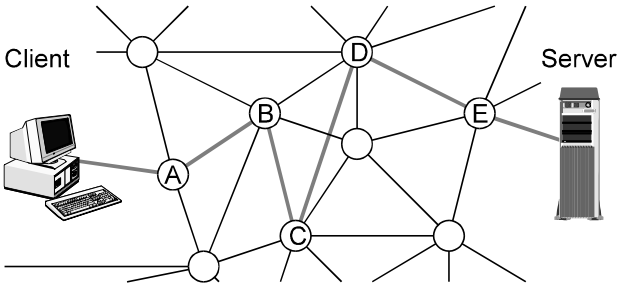


Mixes

Encrypted messages sent over user-selected route through a network

- Packet = A(B(C(D(E(data)))))
- Each server peels off a layer and forwards the data

Servers can only see one hop



Sender and receiver can’t be (easily) linked

Attacks on Mixes

Incoming messages result in outgoing messages

- Reorder messages
- Delay messages

Message sizes change in a predictable manner

Replay message (spam attack)

- Many identical messages will emerge at some point

Onion Routing

Message routing using mixes,
`http://www.itd.nrl.navy.mil/ITD/5540/projects/onion-routing`

Routers have permanent socket connections

Data is sent over short-term connections tunnelled over permanent connections

- 5-layer onions
- 48-byte datagrams
- CREATE/DESTROY for connection control
- DATA/PADDING to move datagrams
- Limited form of datagram reordering
- Onions are padded to compensate for removed layers

Mixmaster

Uses message ID's to stop replay attacks

Message sizes never change

- 'Used' headers are moved to the end, remaining headers are moved up one
- Payload is padded to a fixed size
- Large payloads are broken up into multiple messages
- All parts of the message are encrypted

Encryption is 1024 bit RSA with triple DES

Message has 20 headers of 512 bytes and a 10K body

Crowds

Mixes have two main problems

- Routers are a vulnerable attack point
- Requires static routing

Router vulnerability solved via jondo (anonymous persona)

Messages are forwarded to a random jondo

- Can't tell whether a message originates at a given jondo
- Message and reply follow the same path

Steganography

From the Greek for “hidden writing”, secures data by hiding rather than encryption

- Encryption is usually used as a first step before steganography

Encrypted data looks like white noise

Steganography hides this noise in other data

- By replacing existing noise
- By using it as a model to generate innocuous-looking data

Hiding Information in Noise

All data from analogue sources contains noise

- Background noise
- Sampling/quantisation error
- Equipment/switching noise

Extract the natural noise and replace it with synthetic noise

- Replace least significant bit(s)
- Spread-spectrum coding
- Various other modulation techniques

Examples of channels

- Digital images (PhotoCD, GIF, BMP, PNG)
- Sound (WAV files)
- ISDN voice data

Generating Synthetic Data

Usually only has to fool automated scanners

- Needs to be good enough to get past their detection threshold

Two variants

- Use a statistical model of the target language to generate plausible-looking data
  - "Wants to apply more or right is better than this mechanism. Our only way is surrounded by radio station. When leaving. This mechanism is later years".
  - Works like a text compressor in reverse
  - Can be made arbrtrarily close to real text

Generating Synthetic Data (ctd)

- Use a grammatical model of actual text to build plausible-sounding data
  - "{Steganography|Stego} provides a {means|mechanism} for {hiding|encoding} {hidden|secret} {data|information} in {plain|open} {view|sight}".
  - More work than the statistical model method, but can provide a virtually undetectable channel

Problems with steganography

- The better the steganography, the lower the bandwidth

Main use is as an argument against crypto restrictions

Watermarking

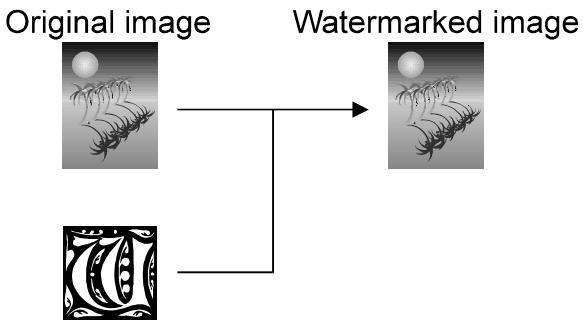
Uses redundancy in image/sound to encode information

Requirements

- Invisibility
- Little effect on compressability
- Robustness
- High detection reliability
- Security
- Inexpensive

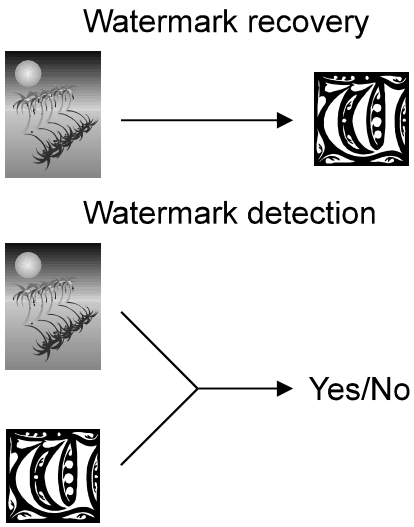
Watermarking (ctd)

Watermark insertion



Watermarking (ctd)

Watermark detection/checking



Watermarking (ctd)

Public watermarking

- Anyone can detect/view the watermark (and try to remove it)

Private watermarking

- Creator can demonstrate ownership using a secret key

Copy Protection Working Group (CPTWG) looking at standardisation, `http://www.dvcc.com/dhsg`

Defeating Watermarking

Lossy compression (JPEG)

Resizing

Noise insertion (print+scan)

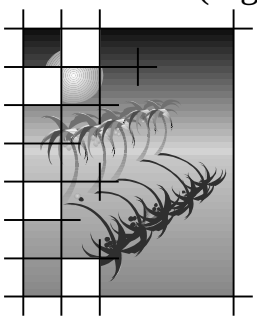
Cropping

Interpretation attacks (neutralise ownership evidence)

Automated anti-watermarking software available (eg UnZign)

Defeating Watermarking (ctd)

Presentation attacks (segmented images)



Watermarking is still in its infancy

- No watermarking standards
- No indication of security/benchmarks
- No legal recognition

Other Crypto Applications

Hashcash

- Requires finding a collision for *n* bits of a hash function
  - "Find a message for which the last 16 bits of the SHA-1 hash are 1F23"
- Forces a program to expend a (configurable) amount of effort before access is granted to a system or service
- Useful for stopping denial-of-service attacks
  - *n* varies as the system load goes up or down
  - Can be used as a spam-blocker

Other Crypto Applications (ctd)

PGP Moose

- Signs all postings to moderated newsgroups
  - Signature is added to the message as an `X-Auth` header
- Unsigned messages (spam, forgeries) are automatically cancelled
- Has so far proven 100% effective in stopping newsgroup spam/forgeries