

Mallory's attack on public keys

1. Alice → Bob: Hey Bob, I've got a really juicy secret to tell you, what's your public key?
2. Bob → Alice (Mallory): Hi Alice, Its K_B , Regards, Bob
3. Mallory (Bob) → Alice: Hi Alice, Its K_M , Regards, Bob
4. Alice → Bob (Mallory): {Secret} K_M
5. Mallory decrypts using K_M^{-1} , reads Secret
6. Mallory (Bob) → Alice: {Secret} K_B

Diffe/Hellman's solution::

A secure online directory D serving Public key requests.

1. each user trusts D
2. each user has a shared key with D
3. n users → use n secure channels to get n^2 secure channels

Kohnefelder's solution: Signed certificates for offline name-key binding validation

Public Key Infrastructure

Active attackers

Eve, the eavesdropper could listen in on the communications on a channel

Mallicious *Mallory*, type attacker can

1. listen in on transmissions,
2. block messages from reaching the intended recipient
3. modify plaintext parts of messages
4. send fake messages on the channel

Certificate Distribution Methods

Certificate gets integrity and verifiability from the signature, so does not need secure storage/transmission.

Can be distributed

1. Along with the signed document
2. As part of a protocol (e.g. SSL)
3. using directory Services (e.g. X.500, LDAP)
4. on web-pages
5. using finger
6. by email ...

Public Key Certificate Standards

Aspects to be standardised:

1. Certificate Syntax
2. Certificate Semantics
3. Rules for Operation of certificate infrastructure
4. Legal Issues, Liability

Real World Certificates

“A document containing a certified statement, especially as to the truth of something”

1. Birth certificates
2. Marriage certificates
3. Degree certificates
4. Doctors certificates

(Basic) Public Key Certificates

The information:

1. **Subject:** name of person/entity holding the key,
2. **Public Key:** key value
3. **Certificate Authority Name:** a name N
4. Signed using N's signature key

X.500 names

X.500: an ITU/ISO standard (1984-1988) for directory services
 assigned a distinguished name for use in directories
 X.500 names constructed from Attributes

E.g. {Country=Australia, Organisation=University of New South
 Wales, Department=Computer Science and Engineering,
 Name=John Smith }

X.509 (version 3) certificate structure

1. Certificate Version
2. Certificate Serial Number
3. CA's signature algorithm ID
4. CA's X.500 name
5. Validity period
6. Subjects X.500 name
7. Subjects Public Key information (Algorithm Identifier, Public
 Key value)
8. (optional) Issuer unique identifier
9. (optional) subject unique identifier
10. (optional) Extension fields

Extension fields

Structure: (Extension Type, Critical/Non-critical, value)

1. Extension type defined by application community
2. Extension type application (user) definable
3. Some extension types standardised
4. *Critical* means "an application should not use this certificate
 unless it understands this field".

Optional Fields

Motivations:

1. X.500 name may not be enough to identify subject to a
 certificate user
2. Application specific naming systems needed (e.g. email address)
3. Certificates can be issued under different policies – application
 may need to know which one
4. Constraints on cross certification (delegation)

Some Standard Extension Types

1. PrivateKeyUsagePeriod
2. CertificatePolicies (val = policy object identifier)
3. PolicyMappings (pairings between similar policy OID's)
4. SubjectAltName (e.g. DNS name)
5. BasicConstraints
- (a) Subject allowed to be a CA?
- (b) Length of chain
6. NameConstraints (when subject a CA).
7. FreshestCRL (how to get it)

Single Certification Authority

Fine within a single, centralised company, but

1. Who can everybody trust?
2. Who is qualified/capable of verifying everyone's identity?
3. How to do identity verification over a distance?
4. Monopoly (you pay to get a certificate)
5. Single point of vulnerability for all applications

Sub Certification Authorities

A root CA, delegating its rights to sign certificates to others (which may delegate further, to some depth)
Certificate says "Key *k* belongs to RA, and RA is authorized by me as a sub-certification authority."
E.g. National Postmaster → State Postmaster → Local PO
In X.509, use *BasicConstraints* extension.

Certificate Chain Logic

If

1. I believe **k0** belongs to CA, and I trust CA as a certification authority
 2. "**k1** belongs to RA1, which is my subCA", signed **k0**
 3. "**k2** belongs to RA2, which is my subCA", signed **k1**
 4. "**k3** belongs to Bill Clinton, which is my subCA", signed **k2**
- I believe **k3** belongs to Bill Clinton.

Multiple CA's

Since in practice, there is nobody that everybody trusts, we have multiple organisations setting themselves up as CA's.

They compete/pay browser/OS vendors to include their keys as trusted keys in their software....

Certificate Chains with Constraints

Name Constraints: what names an RA is allowed to certify

E.g. UNSW may certify names of the form

1. {Country=Australia, Organisation = UNSW, * }
2. *@*.unsw.edu.au

Processing basic and name constraints and validity periods uses a little logic (see SDSL/SPKI discussion later for an example)

PKI, should we outsource?

Commercial trust providers are good for individuals, small co's (Momm & Pop's Corner store).

But there are advantages for large organisations to run their own CA's

1. Can set own rules
2. Maintain desired levels of security
3. Prevent information flow to external bodies
4. can afford the cost/complexity of running a CA

so IBMs, Telstra's, Westpacs etc will tend to have their own root CA.

some of the default Signers (in my Netscape)

ABacom (sub. American Bankers Association) Root CA

American Express CA

American Express Global CA

Belisign Object Publishing CA

Belisign Secure Server CA

Deutsche Telekom AG Root CA

Digital Signature Trust Co. Global CA

Entrust.net Secure Personal CA

Equifax Premium CA

GTE CyberTrust Global Root

Thawte Premium Server CA

Verisign Class 1-4 Public Primary Certification Authority

Revocation

Keys do get compromised, smartcards lost, employees leave the company, etc.

So we need to be able to undo the effects of a certificate.

Approaches:

1. Certificate Revocation Lists
2. Online revvalidation

Certificate Revocation Lists

A signed statement with contents (X.509):

1. CRL version no
2. Issuers Signature algorithm ID
3. Issuers X.500 name
4. time this CRL issued
5. time next CRL issued
6. List of (Revoked Cert ID No., Revocation date)

Cross Certification of CA's

To interact, organisational CA's can cross certify when desired, using E.g. "CA_Telstra is trusted for domain .telstra.com", signed **kLBM**"

Rules for certificate chaining get more complicated.

E.g. "accept certificate chains going up to a root CA, across to another CA, then down to a user in the other organisation"

PKI Anarchy

PGP (Pretty Good Privacy, encryption package for email/file encryption) trust model

1. "web of trust" (introduction via friends)
2. *everyone* can act as a CA, and certify keys
3. (vague) user set levels of trust: full, marginal, untrustworthy; don't know (full = this public-key is fully trusted to introduce another public-key)
4. user can set parameters describing number of certificates at full/marginal levels required before a key is trusted. e.g. trust a public key/name binding if you have certificates from 3 fully trusted sources or 7 marginally trusted sources (problem: how do you know the sources are independent?)

Certificate Policies

A set of rules governing the running of a CA:

1. To whom will this CA issue certificates? (e.g. employees of UNSW)
2. What ID do these people need to present to get a certificate?
3. How does the CA physically protect its signature keys?
4. How often are CRL's issued?
5. What checks are done before a Sub-CA is certified?
6. What liability does the CA accept wrt its certificates?
7. Who is responsible for maintaining this policy, & how is it published?

Where are keys generated?

Answer1: In the users system:

1. Good for signature keys (I don't want anyone else to know mine!)
2. User needs to prove possession of the private key (protocol for this)
3. User needs to transmit public key to CA securely

Issues

1. How are CRL's distributed?
2. How large do they grow? (need Delta-CRLs)
3. Cost of running revocation service.
4. How long between updates?
5. Cost/Risk of key compromises occurring between updates?
6. Need suspensions as well as revocations?

Online Status Checks

Maintain a server to answer queries "is this certificate still valid?"

Main issue is cost

1. of keeping the server reliably online.
2. of security controls on server, server data.
3. Query processing requires crypto computation (signature) slow/expensive

How best to manage risk/cost tradeoff: ongoing research

Gatekeeper PKI

<http://www.govonline.gov.au/projects/confidence/Securing/gatekeeper.htm>

1. CA Standard developed by Australian National Office of Information Environment (NOIE)
2. for business/govt interactions
3. X.509 based, with extension for ABN
4. Sets out policy, legal issues
5. NOIE accredits organisations and govt agencies as CA's
6. Identrus certificates accepted for ABN 2001

Where are keys generated?

Answer2: By the CA

1. Bad for signature keys (I don't want anyone else to know mine!)
2. CA has assurance of key quality.
3. Allows escrow of private encryption key (e.g. if employee leaves company/dies, co. can recover encrypted documents)
4. CA needs to transmit private key to user securely

PKI Radicals

Summary:

- problems with X509
- SPSI: local vs global names
- SPKI: authorization rather than identity certificates
- Trust Management systems: Policymaker, Keynote

Arguments that global root CA is unworkable:

1. Politically: where do you find a global root that everybody trusts?
2. Legally: global root CA would need to take into account relevant law of all countries.
3. on liability grounds: certificates issued by current CA's restricted to particular applications: e.g. government communications, financial institutions

Result:

1. cross-certification of CA's
2. PGP model: globally unique names, but "web of trust": "everyone is a CA"

PKI Radicals

Summary:

- problems with X509
- SPSI: local vs global names
- SPKI: authorization rather than identity certificates
- Trust Management systems: Policymaker, Keynote

Examples where certificates listing the key holders name is undesirable:

1. voting over the internet
2. intelligence agency employees, undercover police
3. sensitive medical information services, e.g. getting results of medical tests

Blaze, Feigenbaum and Lacy, Distributed Trust Management (1996) argue that instead of **(key, name)** certificates + **(name, right)** access control list, **(key, right)** certificates should use

Arguments against globally unique names:

1. privacy invasion: allow a profile to be built up
2. requires a naming infrastructure with a global root
3. tendency to try to use common names, which are not unique even in small communities
4. "knowing who someone is" useful in a small community, but in a global community, names don't automatically give useful information
5. (with overwhelming probability) keys already uniquely associated with a key holder

Are names even relevant?

Decisions about keyholders are typically based on the fact that the keyholder

1. regularly pays their bills
2. is an employee of company X
3. has enough money in their account
4. has been granted authority to perform the requested action, etc

rather than the fact that they are called "John Smith"

Linking local name spaces

principals can make use of names from another principal's name space
 my name "Joe's Fred" means
 "the principal(s) that the principal(s) I call Joe call Fred"

A SDSI *name* is an expression (name N1 N2 ...Nk) where N1 is a string or a key and the remaining Ni are strings, also written "N1's N2's ... Nk"

If N1 is a key then the name is *fully qualified*
 The meaning of a name is relative to the name space
 The meaning of a fully qualified name is independent of the name space

Local names: SDSI

Rivest and Lampson: Simple Distributed Security Infrastructure
 – identify principals with keys
 – each principal is associated with a *local name space*
 – a name is bound to a (possibly many) keys in each local name space
 – the same name can have different meanings in different name spaces

$K_R: \text{Fred} \mapsto K_1$
 $K_J: \text{Fred} \mapsto K_2$

SPKI: Simple Public Key Infrastructure

– working group starting around 96,
 – focus on *authorization certificates* and delegation
 – motivated by growing complexity of X.509
 – X.509 certificates too large to be used in smart cards/current mobile devices
 – now merged with SDSI
 – current IETF RFC: (Internet Engineering Taskforce Request for Comment), dormant since 99
 – used mainly within closed systems rather than for interoperability

Interpreting linked local names

if

$K_R: \text{Joe} \mapsto K_J$
 $K_J: \text{Fred} \mapsto K_F$

then

$K_R: \text{Joe's Fred} \mapsto K_F$

SPKI certificate format

Naming certificates

(cert (issuer KEY STRING) (subject SDSI-NAME) (validity V))

says STRING is bound to SDSI-NAME in KEY's namespace during period V, which is either

1. an interval of time during which the certificate is valid

2. (online URL) - stating that validity of the certificate should be checked at verification time by a query to URL

3. (url URL) - stating that validity can be checked by the certificate revocation list available at the given URL

- A is an expression denoting a set of actions
- the (propagate) field is optional

where

(validity V))

(cert (issuer KEY) (subject SDSI-NAME) (propagate)? (tag A))

Authorization certificates

Two modes of operation of SPKI

1. **Proof Presentation:** prover presents a sequence of certificates. Verifier checks that when the *tuple reduction rules* are applied using the certificates in the order presented, the result is a tuple stating that the request is authorized
2. **Certificate Discovery:** prover presents an unordered set of certificates. Verifier tries to find a subset from which there exists a derivation of a tuple stating that the request is authorized

Authorization certificates say that according to the holder of KEY, any principal K denoted by SDSI-NAME has the right to perform any action denoted by A while during the validity V period denoted by V. Moreover, if (propagate) is present, K has the right to grant that right to any other principal

authorization certifies

(cert (issuer K N) (subject P) (validity V))

are translated to *4-tuples*

$\langle K, N, P, V \rangle$

4-tuple reduction rule

$\langle K_1, N, K_2, S, M, P, V_1 \rangle + \langle K_2, M, Q, V_2 \rangle \longrightarrow \langle K_1, N, Q, S, P, V_1 \cup V_2 \rangle$

where

D is true if the propagate field is present and false otherwise.

$\langle K, K', S, D, A, V \cup V' \rangle$

SPKI tuple reduction rules

naming certifies

(cert (issuer K N) (subject P) (validity V))

are translated to *4-tuples*

$\langle K, N, P, V \rangle$

4-tuple reduction rule

$\langle K_1, N, K_2, S, M, P, V_1 \rangle + \langle K_2, M, Q, V_2 \rangle \longrightarrow \langle K_1, N, Q, S, P, V_1 \cup V_2 \rangle$

Access control policies

Examples:

- access control lists (e.g. UNIX file permissions)
- Chinese wall: e.g. in a legal firm, a lawyer working on the Pepsi account may not access files on the Coca-Cola account on conflict of interest grounds
- Authorization required from two independent sources (Clark/Wilson models)

(e.g. processing of foreign currency transactions by bank tellers)

5-tuple reduction rules:

$\langle K_1, S_1, \text{true}, A_1, V_1 \rangle + \langle S_1, S_2, D_2, A_2, V_2 \rangle \longrightarrow \langle K_1, S_2, D_2, A_1 \cup A_2, V_1 \cup V_2 \rangle$

Trust Management Approach

Blaze et al propose that the verification of policy compliance should be performed

- *not* in each application
- *but* in a *trust management engine* that acts as a server to all applications

Input to trust management engine:

1. requested action
2. certificates
3. description of local policy

Output: a recommendation about whether the action is approved

Advantages

1. Expressiveness: can go beyond ACL based authorization (and beyond SPKI)
2. Extensibility of policy
3. Ease of modification of policy (don't need to hack application code)
4. Coherency/Integration of policy across applications
5. Transparency of policy: not hidden in application code, can be made readable by native users by appropriate choice of policy representation language

Role based access control/authorization

Here rights attach not to people, but to *roles* in an organisation, and people have the rights by virtue of the fact that they *fill* those roles. E.g.,

- *Project leaders* may read the working files of any members of their teams.
- The *Head of School* has authority to authorize expenditure greater than \$50,000.
- During periods of absence, the Head of School may *delegate* his right to authorize expenditures of up to \$25,000 to an *acting Head of School*.

To apply PKI to enforce policies, *each application* of interest must be modified to do the following before taking an action:

1. obtain certificates, verify their signatures, determine public key of originator
2. verify that certificates are unrevoked
3. attempt to find "trust path" from trusted root to certificate of public key of interest
4. (X509 model) extract names from certificates, lookup names in a databases
5. determine whether the requested action is in compliance with local policy and whether the CA's have the right to to issue the relevant certificates under local policy
6. if all OK, proceed

REFERENCE (AT&T Labs, MIT 97)

- designed for trust management in web browsing
- “yes/no/don't know” output
- policies may call other policies
- trust engine may fetch missing credential from the net

Prototype Systems

Policy-Maker (AT&T Labs 96):

- policies can be expressed in any language that can be interpreted
- “safety” (with termination guarantees)
- safe version of AWK developed
- calling application responsible for gathering and verifying all certificate signatures/revocation

Keynote (UPenn-AT&T Labs 98)

- crypto verification done in trust engine
- simple human readable condition language for policies, e.g.


```
$file = “/etc/passwd” && $access=”read” – > {return “ok”}
```
- no loops

Ongoing research:

- what level of expressiveness do practical policies need?
- what are appropriate languages for policies?
- how can we make them human-comprehensible?
- what algorithms are required for policy processing?