

# Public-Key Infrastructures

## Cryptography is not enough

Cryptography is useful for authenticity and secrecy.

- A digital signature with a public key may be interpreted as an authenticity assertion.
- Encryption under a public key may offer some secrecy guarantees.
- Encryption under a shared key may offer both authenticity and secrecy.

But are you using the right key?

## Certification authorities (CAs)

If  $A$  sends its public key to  $B$ , anyone can replace it with their own public key.

A certification authority solves this problem by signing  $A$ 's public key.

The key may be

- a signature key,
- an encryption key, or
- both.

## Some questions

We have reduced a hard problem to lots of hard problems:

- How do you get the key from the CA?
- Who is the CA?
- Why do you trust them?
- Why do you trust them for this purpose?
- How is the key stored?
- Is the key still valid?

If the key is associated with a name,

- Is it the right name?
- Is it a meaningful name?
- Do you actually check the name?
- What if the check fails?

Other associations are important too:

- names / groups,
- names / powers.

## Public-key infrastructures (PKIs)

A variety of PKIs address the previous questions in different ways.

Basic tasks:

- creation of certificates,
- dissemination of certificates,
- revocation of certificates,
- renovation of certificates,
- (sometimes) key escrow,
- (sometimes) key archival.

## The phonebook CA

Early on, there was a hope that one could simply have a directory that associates public keys with names.

The directory could be actively updated and queried.

The directory can be implemented as a set of certificates, signed with a CA key.

(But then the issue of revocation comes up.)

## Obtaining a certificate (one method)

*A* generates a key pair.

*A* signs the public key and its identity information with the secret key.

CA does some verifications.

CA signs the public key and *A*'s identity information.

*A* checks CA's certificate.

## Certificate distribution

*A* may show (push) its certificate when it uses its keys.

Alternatively, other parties may request (pull) the certificate:

- from *CA*,
- from other directories,
- from *A*.

*CA* may be off-line (in a safe!) most of the time.

## What certificates say

A certificate typically binds a public key  $K$  and a name  $A$ .

- The corresponding secret key speaks for  $A$ :
  - If the secret key signs  $M$  then  $A$  supports  $M$ .
  - $A$  may or may not control the secret key.

A certificate may also say:

- A business is legitimate.
- A user has a clearance.
- A user is Italian.
- A user may modify a part of an X.500 database.

# Running a CA

## Issues:

- Business model.
- Limiting responsibilities, insurance.
- Bootstrapping.

## Policies:

- Obligations of users (truthfulness, secrecy).
- Information collection and checking.
- Renovation.
- Publishing certificates, revocations.
- Use of certificates.
- Logging, auditing.
- Confidentiality.
- Disaster recovery.
- Security: physical, personnel, computer.

# Expiration and revocation

Keys may become suspect or compromised.

Problems:

- How can you tell if a key is compromised?
- What do you do?

Approaches:

- Expiration times in certificates.
- Certificate renovation.
- Certificate revocation lists (CRLs)  
with many twists.
- On-line status checking  
with many protocols.
- Forward security  
for encryption but also for signatures.

## Shared-key vs. public-key

Similar considerations apply in shared-key systems (e.g., Passport).

Shared key is typically more efficient.

Shared key requires on-line trusted authorities.

Public key still requires on-line servers for revocation, if quick revocation is wanted.

## Closed infrastructures

Simple, fairly monolithic CAs make sense in closed systems:

- military agencies,
- banking service providers (SWIFT),
- some DRM systems,
- telco systems (?).

Then each entity may end up with many keys, which should all be used for appropriate purposes.

## Scaling issues: trust

Finding the CA's public key.

Trusting the CA.

- For all purposes?
- Even on matters close to you?

Why would the CA want to be trusted that much?

What is trust?

Creation and enforcement of standards.

## Scaling issues: names

Who is “John Smith”?

Names are subject to conventions.

The same entity may have many names and pseudonyms.

Names are not always useful and convenient.

Names are not stable.

Early binding may feel safer but can be cumbersome.

Adding addresses, etc., only complicates matters.

Are UIDs an acceptable substitute for names? SSNs?

## Hierarchical CAs

Having a single CA is unrealistic:

- No single CA is trusted by everyone for everything.
- A single CA may be a bottleneck.

One solution is to have a hierarchy of CAs:

CA1 certifies A,

CA2 certifies CA1,

⋮

Root certifies CA<sub>n</sub>.

Certificates are chained.

## Hierarchical CAs (cont.)

The certification path may be determined by *A*'s name.

E.g., if *A* is `abadi@cs.ucsc.edu`

then CA1 may be `CA@cs.ucsc.edu`.

The hierarchy of names corresponds to the hierarchy of CAs. (See PEM.)

We can avoid going up to Root for “nearby” names.

⇒ Better performance.

⇒ More appropriate trust.

We may also avoid going up to Root through cross-certification.

(Cross-certification can be messy.)

## X.500

X.500 relies on the notion of distinguished names (DNs).  
Everything should have a DN.

A DN includes:

- country,
  - state or province,
  - locality,
  - organization,
  - organizational unit,
  - common name,
- and
- certificate type,
  - email address,
  - fields required by signature laws,
  - nonces.

## X.500 (cont.)

But:

- There is no agreement on what all these mean, e.g., localities?
- Some certificates contain meaningless or dummy values.

X.509 v3 added support for other forms of names, in particular, attribute / value pairs.

X.509 is vague in many areas.

## X.500 (cont.)

Implementations have had many problems, and do not always interoperate.

X.509 has not displaced passwords.

X.509 certificates are widely used nevertheless (e.g., with SSL).

## “Web of Trust”

“Web of Trust” is an alternative to hierarchical CAs introduced with PGP.

Each user gives a trust rating to each public key on the user’s ring:

- validity (of binding to some other user),
- trust (none, partial, complete).

If a certificate for a key is signed by two partially trusted users, then the key is deemed valid.

Signature keys are used to certify encryption keys.

“Web of Trust” avoids some scaling problems associated with global trust and global names.

## SPKI and the like

SPKI certificates deal with:

- identity,
- authorization of actions,
- delegation,
- membership.

SPKI relies on linked local name spaces.

SPKI supports some hierarchical and web-of-trust certification.

## Global names

Global identifiers are understood equally in all name spaces.

Global identifiers include:

- public keys,
- global names (like DNS!), which represents the root of the DNS hierarchy).

## Local names

Local names are auxiliary names that each principal can use arbitrarily.

For example, the name `Lampson` may be associated with a public key  $KL$ .

$\Rightarrow$  Statements whose signatures can be checked with  $KL$  will be viewed as coming from `Lampson`.

Each principal has a name space where local names are bound to values.

## Local names (cont.)

- Lampson may be used as a local name by anyone, and bound to any value.
- The value could be a global identifier (such as a public key), or another local name.
  - Two principals may disagree on the value of Lampson.
  - So two principals may disagree on whether a statement comes from Lampson.

## Linked local name spaces

The value of a name may be defined by reference to the name spaces of other principals.

Rivest may be associated with Lampson's Rivest.

⇒ If Lampson says that Rivest's public key is  $KR$ , then statements whose signatures can be checked with  $KR$  will be viewed as coming from Rivest.

Compound names like Lampson's Rivest allow one name space to import bindings from another.

## Principals

A compound name is an expression of the form

(ref:  $p_1 \dots p_n$ )

which is abbreviated to  $p_1$ 's ...'s  $p_n$ .

Each of  $p_1, \dots, p_n$  may be a global identifier, a local name, or a compound name.

A principal is

- a global identifier,
- a local name, or
- a compound name.

## Sources and further reading

Rants by Bruce Schneier.

Ross Anderson's book.

Peter Gutmann's tutorial on network security.

Fred Schneider's class material.

X.500 specifications.

SPKI specifications and presentations.

My own work on authentication in distributed systems.