SPKI/SDSI 2.0
A Simple Distributed Security Infrastructure

by Ronald L. Rivest

MIT Lab for Computer Science

(Joint work with Butler Lampson and Carl Ellison)

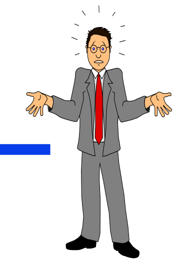
Outline

- context and history
- motivation and goals
- syntax
- public keys (principals)
- naming and certificates
- groups and access control

The Context

- Public-key cryptography invented in 1976 by Diffie, Hellman, and Merkle, enabling:
  - *Digital signatures:*
    private key signs, public key verifies.
  - *Privacy:*
    public key encyrpts, private key decrypts.
- But: *Are you using the "right" public key?* Public keys must be *authentic*, even though they need not be *secret*.

How to Obtain the “Right’’ PK?



- Directly from its owner
- Indirectly, in a signed message from a trusted *certification agent* (CA):
  - A *certificate* (Kohnfelder, 1978) is a digitally signed message from a CA binding a public key to a name:

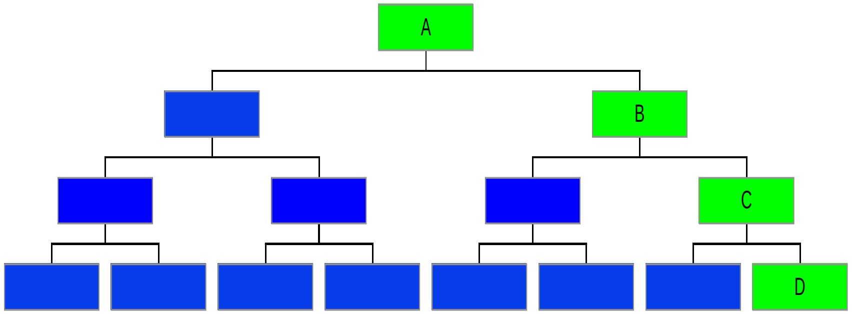
    “The public key of *Bob Smith* is *432102571376553422086*7 (signed: CA)’’
  - Certificates can be passed around, or managed in directories.

Scaling-Up Problems

- How do I find out the CA’s public-key (in an authentic manner)?
- How can everyone have a *unique name*?
- Will these unique names actually be *useful* to me in identifying the correct public key?
- Will these names be *easy to use*?

Hierarchical “Solution”

- (PEM, X.509): Use a global hierarchy with one (or few) top-level roots:



- Use *certificate chains* (root to leaf):

$$A \longrightarrow B \longrightarrow C \longrightarrow D$$

- Names are also hierarchical: *A/B/C/D*.

Scaling-Up Problems (continued)

- Global name spaces are politically and technically difficult to implement.
- Lawyers must get involved if one wants certificates to support commerce or binding contracts. Standards of due care for issuing certificates must be created.
- Nonetheless, a global hierarchical PK infrastructure is slowly beginning to appear (e.g. VeriSign).

PGP "Solution"

- User chooses name (userid) for his public key:

  Robert E. Smith <res@xyz.com>

- Bottom-up approach where anyone can "certify" a key (and its attached userid).
- "Web of trust" algorithm for determining when a key/userid is trusted.

Is There a Better Way?

- Reconsider goals...
- Standard problem is to *implement name ⟷ key maps*:
  - Given a public key, identify its owner by name
  - Find public key of a party with given name
- But often the "real'' problem is to *build secure distributed computing systems*:
  - *Access control* is paradigmatic application: should a digitally signed request (e.g. http request for a Web page) be honored?

SPKI/SDSI (“spooky”?/“sudsy”)

- **S**imple **P**ublic **K**ey **I**nfrastructure
- **S**imple **D**istributed **S**ecurity **I**nfrastructure
- **SDSI** is effort by Butler Lampson and myself to rethink what’s needed for distributed systems’ security. It attempts to be fresh design (start with a clean slate).
- **SPKI** is effort by Carl Ellison and others to design public-key infrastructure for IETF.
- **SPKI/SDSI** is a merger of these designs.

Motivations:

- Incredibly slow development of PK infrastructure
- Sense that existing PK infrastructure proposals are:
  - too complex (e.g. ASN.1 encodings)
  - an inadequate foundation for developing secure distributed systems
- A sensed need within W3C security working group for a better PK infrastructure

Related Work

- Blaze, Feigenbaum, and Lacy's work on "decentralized trust management" (Policy-Maker)
- W3C (world wide web consortium) work on security and on PICS
- Evolution of X.509 standards

Simple Syntax (S-expressions)

Byte-strings:

`abc`	(token)
`“Bob Dole”`	(quoted string)
`&4A5B70`	(hexadecimal)
`=TRa5`	(base-64)
`#3:def`	(length:verbatim)
`[unicode] &3415AB8C`	(display hint)
`abc~ def = abcdef`	(fragmentation)

Lists:

```
(certificate (issuer bob)
             (subject alice))
```

Principals are Keys

- Our active agents (principals) are *keys*: specifically, the private keys that sign statements. We identify a principal with the corresponding verification (public) key:

```
(public-key
  (rsa-md5-verify
    object
    signature
    (const &03)
    (const &435affd1…)))
```

- In practice, keys are often represented by their hash values.

Keys may be simple programs

- ```
  (public-key
    (let object-hash (md5 object))
    (equal object-hash
           (rsa signature
              (const &03)
              (const &435affd1…))))
  ```
- Programming language has only two statement types:
  - assignment statements
  - equality tests.
  ```

All Keys are Equal

- Each principal can make signed statements, just like any other principal.
- These signed statements may be certificates, requests, or arbitrary S-expressions.
- This egalitarian design facilitates rapid “bottom-up” deployment of SPKI/SDSI.

Signed Objects

- Signing creates a separate object, containing the hash of object being signed.
-
```
(signed
  (object-hash (hash sha1 &84…))
  (signer (public-key …))
  (signature &5632…))
```

Encrypted Objects

- ```
  (encrypted
     (key (hash sha1 &DA…))
     (ciphertext =AZrG…))
  ```
- One can indicate the key:
  - by its hash value
  - in encrypted form
  - using its name
  ```

Users Deal with *Names*, not Keys

- The point of having names is to allow a convenient understandable user interface.
- To make it workable, the *user* must be allowed to choose names for keys he refers to in ACL's.
- The binding between names and keys is necessarily a careful manual process. (The evidence used may include credentials such as VeriSign or PGP certificates...)

Names in SDSI are *local*

- All names are *local* to some principal; there is no global name space. Each principal has its own local name space.
- Syntax: `(ref <key> name)`
  (or just `(ref name)` if key is understood)
- A principal can use *arbitrary* local names; two principals might use the same name differently, or name another key differently.
- Linking of name spaces allows principals to use definitions another principal has made.

Linking of name spaces

- A principal can *export* name/value bindings by issuing corresponding certificates.
- Name spaces are *linked;* I can refer to keys named: `(ref bob)`
  `(ref bob alice)`
  `(ref bob alice mother)`
  if I have defined `bob`,
  `bob` has defined `alice`, and
  `alice` has defined `mother`.

Certificates in SPKI/SDSI 2.0

- These take a single unified form, but are used for many purposes:
  - binding a local name to a value
  - defining membership in a group
  - delegating rights to others
  - specifying attributes of documents and of key-holders

Certificate Parts

- *issuer:* <key> or (ref <key> name)
- *subject:* <key> or
  (ref <key> name$_1$ … name$_k$)
  or a document (or its hash)
- *validity period*
  (not-before …) (not-after …)
  Note: no revocation of certificates!
- *tag:* specifying rights or attributes
- *propagation-control:* a boolean flag

Sample Certificate

```
(certificate
  (issuer (ref <my-key> “Bob Smith”))
  (subject <bob’s-key>)
  (not-after 1996-03-19_07:00 )
  (tag (*)))
```

This defines <bob’s-key> as the value of the name “Bob Smith” in my key’s name space . The tag (*) means that <bob’s-key> inherits all the rights of my name “Bob Smith”.

Certificate Chains

- A sequence of certificates can form a *chain*, where definitions cascade and rights flow.
- {K1} ==> {K1 mit rivest} (tag (read foo))
  {K1 mit} ==> {K2} (tag (read (*)))
  {K2 rivest} ==> {K3} (tag (read (*)))
  is equivalent to:
  {K1} ==> {K3} (tag (read foo))
- Validity periods and tags intersect.
- A request may be accompanied by a chain.

Generalized tags and *-forms

- There are a set of "*-forms" for writing tags that represent a *set* of *-free tags. The system can automatically intersect these sets, even though tag semantics is application-dependent.
-
```
(tag
 (spend-money
  (account (* set 1234 5678))
  (date (* range date 1997 1998))
  (amount
    (* range numeric 1 1000))))
```

Propagation Control

- A certificate may turn on *propagation control*, in which case rewriting of issuer's name in a certificate chain can not proceed past the point where it is rewritten to be a single key.
- Examples:
  – Subscribers to on-line journal
  – Group of individuals who are "adults".

Cert can also describe keyholder

```
(certificate
  (issuer <rons-key>)
  (subject (keyholder <rons-key>))
  (not-after 1998-01-01_00:00)
  (tag  (name “Ronald L. Rivest”)
        (postal-Address ... )
        (phone 617-555-1212)
        (photo [image/gif] ... )
        (email rivest@mit.edu )
        (server “http://aol.com/~rlr” )))
```

On-line orientation

- We assume that each principal can provide on-line service directly, or indirectly through a server.
- A server provides:
  - access to certificates issued by the principal
  - access to other objects owned by principal

A Simple Query to Server

- A server can be queried:
  "What is the current definition your principal gives to the local name `bob' ?"
- Server replies with:
  – Most recent certificate defining that name,
  – a signed reply: "no such definition", or
  – a signed reply: "access denied."

Access Control for Web Pages

- Motivating application for design of SDSI.
- Discretionary access control: server maintains an access-control list (ACL) for each object (e.g. web page) managed.
- A central question: how to make ACL's easy to create, understand, and maintain? (If it's not easy, it won't happen.)
- Solution: <u>named groups of principals</u>

Groups define sets of principals

- Distributed version of UNIX "user groups"
- A principal may define a local name to refer to a *group* of principals:
  - using names of other principals:
    `friends include bob alice tom`
  - using names of other groups:
    `enemies include mgrs vps`
- Defining principal can export group definitions, so you may say:

```
friends include ron (ref ron friends)
```

"Membership Certificates"

- Just like name/value certificate, where name is "group name"; subject is member or subgroup. (Group is "multivalued name".)
- ```
  (certificate
     (issuer (ref <mitkey> faculty))
     (subject <bob's-key> )
     (tag (*))
     (not-after 1997-07-01))
  ```
- Subject could also be another group, whose members are included in issuer group.
  ```

Sample ACLs

```
(acl (subject friends) (tag read))

(acl (subject(ref AOL subscribers))
     (tag read))

(acl (subject (ref VeriSign adults))
     (tag (http “http://abc.com/adult”)))

(acl (subject (ref ibm employees)
              (ref mit faculty))
     (tag read write))
```

Querying for protected objects

- Can query server for any object it has.
- If access is denied, server's reply may give the (relevant part of) the ACL.
- If ACL depends upon remotely-defined groups, *requestor* is responsible for obtaining appropriate certificates and including them as credentials (certificate chain) in a re-attempted query.

Implementations of SDSI 1.0

- Microsoft (Wei Dai, in C++)
- MIT (Matt Fredette, in C)
- Both implementations up and running now. (No compatibility testing yet…)
- Gillian Elcock is completing a web-based certificate-manager support system.

Recap of major design principles

- ACLs must be easy to write & understand
- Principals are public keys
- Linked local name spaces (one per key)
- Groups provide clarity for ACLs
- On-line client/server orientation
- Client does work of proving authorization
- Certificates support flexible naming and authorization patterns.
- Simple syntax

Conclusions

- We have presented a simple yet powerful framework for managing security in a distributed environment.
- Draft of our paper available at:

  `http://theory.lcs.mit.edu/~rivest`

  (Currently just SDSI 1.0; SPKI/SDSI 2.0 coming soon. These slides will be posted.)
- Comments appreciated!