On Public-Key Infrastructures

by Ronald L. Rivest
MIT Lab for Computer Science

(SDSI is joint work with Butler Lampson)

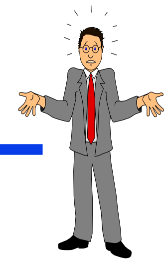
Outline

- Context and history
- Motivation and goals
- “SDSI” (Simple Distributed Security Infrastructure):
  – 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:

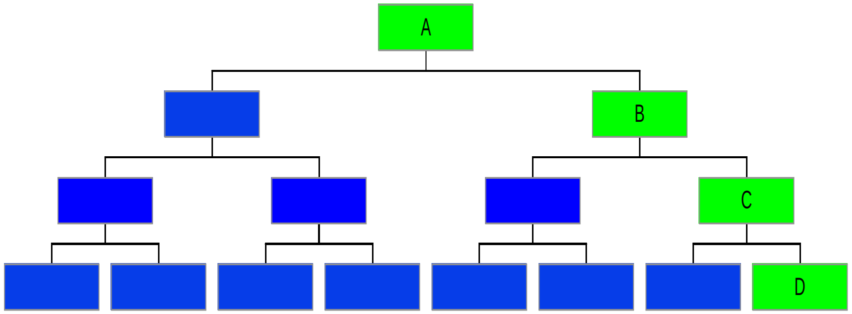
    “The public key of *Bob Smith* is *4321025713765534220867* (signed: CA)’’
  - Certificates can be passed around, or managed in directories.

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.
- Legal issues arise 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 <u>*implement name ⟷ key maps*</u>:
  - 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 <u>*build secure distributed computing systems*</u>:
  - *Access control* is paradigmatic application: should a digitally signed request (e.g. http request for a Web page) be honored?

SDSI (a.k.a. “sudsy”)

- <u>**S**</u>imple <u>**D**</u>istributed <u>**S**</u>ecurity <u>**I**</u>nfrastructure
- Effort by Butler Lampson and myself to rethink what’s needed for distributed systems’ security.
- Attempts to be fresh design (start with a clean slate).

Motivations for designing SDSI:

- 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

- IETF's "SPKI" (Simple Public Key Infrastructure) working group (esp. Carl Ellison's work)
- Blaze, Feigenbaum, and Lacy's work on "decentralized trust management"
- W3C (world wide web consortium) work on security and on PICS
- Evolution of X.509 standards

SDSI has Simple Syntax

A SDSI object (an *S-expression*) may be:

`abc`	(token)
`“Bob Dole”`	(quoted string)
`#4A5B70`	(hexadecimal)
`=TRa5`	(base-64)
`#03:def`	(length:verbatim)
`[unicode] #3415AB8C`	(with hint)
`( RSA-with-MD5:`	(list)

```
( RSA-with-MD5:
   ( E: #03 )
   ( N: #42379F3A0721BB17 ) )
```

Keys are ``Principals''

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

```
( Principal:
  ( Public-Key:
    ( RSA-with-MD5:
      ( E: #03 )
      ( N: #34FBA341FF73 ) ) )
  ( Principal-At: "http://abc.def.com/" ) )
```

All Keys are Equal*

- Each SDSI 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 SDSI.
- * Some SDSI principals may have a special syntax, e.g.: `VeriSign!!  USPS!!`

Signed Objects

- Signing adds a new signature element to end of list representing object being signed.
- A signature can be managed independently of the corresponding signed object.
- An object may be multiply-signed.
- A signature element may itself be signed (this is used to *reconfirm* a signature).

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.
- A principal can use *arbitrary* local names; there is no need for you and I to give the same name to a public key.
- Two important exceptions:
  - linking of name spaces (indirection)
  - special treatment for DNS names

Linking of name spaces

- A principal can *export* name/value bindings by issuing corresponding certificates.
- SDSI syntax supports linking (indirection);
  I can refer to keys (values) named:

  `bob`

  `bob’s alice`

  `bob’s alice’s mother`

  if I have defined `bob`, `bob` has defined `alice`, and `alice` has defined `mother`.
- (Sugar for `( ref: bob alice mother )`)

DNS names get special treatment

- A name of the form:
  `billg@microsoft.com`
  is equivalent to the indirect form:
  `DNS!!'s com's microsoft's billg`
- (This assumes that public keys for entities in the DNS have been created, which may happen in the not too distant future.)

Certificates

- Certificates are signed statements (signed S-expressions).
- Certificates may bind names to values (e.g. to principals or group definitions), may describe the owner of public key, or serve other functions.
- A certificate has an issuer (signer) and an expiration date.

Sample Certificate

```
( Cert:
  ( Local-Name: “Bob Smith” )
  ( Value: ( Principal: ... ) )
  ( Signed:
    ( Object-Hash: ( SHA-1: #34FD4A ) )
    ( Date: 1996-03-19T07:00 )
    ( Expiration-Date: 2000-01-01T00:00 )
    ( Signer: ( Principal: ... ) )
    ( Signature: #57ACD1 ) ) )
```

Auto-Certificates describe signer

```
( Auto-Cert:
    ( Public-Key: ... )
    ( Principal-At: http://bu.edu )
    ( Server: http://aol.com )
    ( Name: “Robert E. Smith” )
    ( Postal-Address: ... )
    ( Phone: 617-555-1212 )
    ( Photo: [image/gif] ... )
    ( Email: alice@abc.com )
    ( Signed: ... ) )
```

On-line orientation

- SDSI assumes that each principal can provide on-line service, either directly or (more typically) indirectly through a server.
- A SDSI server provides:
  - access to certificates issued by the principal
  - access to other objects owned by principal
  - *reconfirmation* service for expired certificates (SDSI does not have CRL's !)

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."

Reconfirmation of Certificates

- SDSI certificates have an expiration date, and may have a *reconfirmation period*.
- A certificate is valid before expiration, as long as most recent signature is not older than reconfirmation period.
- A principal (or one of its authorized servers) may reconfirm certificate by supplying a fresh *reconfirmation signature*.

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 = ( Group: bob alice tom )`
  - using names of other groups, and algebra:
    `enemies = ( Group: ( OR: mgrs vps ) )`
- Defining principal can export group definitions, so you may say:

```
friends = ( Group: ron ron’s friends )
```

Sample ACL's

```
( ACL: ( read: friends ) )
( ACL: ( read: AOL's subscribers ) )
( ACL: ( read: VeriSign!!'s adults ) )
( ACL: ( read: microsoft's employees ) )
( ACL: ( write: ( OR: bob bob's asst )))
( ACL: ( read:
          ( OR: bob
                bob's friends
                mit's eecs's faculty ) )
       ( write: ron ) )
```

Querying for protected objects

- Can query server for any SDSI 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 ``membership certificates'' and including them as credentials in his re-attempted query.

Membership Certificates

- Issued by principal defining group, or his server, when requested.
- ```
  ( Membership.Cert:
      ( Member: <ron's principal> ) )
      ( Group: faculty )
      ( Signed:
          ( Signer: <mit's principal> )
          ... ) )
  ```
  ```

Encrypted Objects

- ```
  ( Encrypted:
    ( Key: ( Key-Hash:
               ( SHA-1 #DA3710 ) ) )
    ( Ciphertext:
      =AZrGT57+30vB1QsMPuI5Ol79 ) )
  ```
- One can indicate the key:
  - by its hash value
  - in encrypted form
  - through its name
  ```

Other issues and topics

- Generalized queries and templates
- Delegation and delegation certificates
- Multiply-signed requests
- Algorithm for evaluating names
- Algorithm for determining group membership
- Data compression

Implementations

- Microsoft (Wei Dai, in C++)
- MIT (Matt Fredette, in C)
- We expect working code by end of 1996.

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
- Reconfirmation instead of CRLs
- Signing authority can be delegated
- 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`
- Comments appreciated!