6.857 Computer and Network Security	Fall Term, 1997
Lecture 21 : November 18, 1997	
Lecturer: Ron Rivest	*Scribe: James Megquier*

1 Topics

- SDSI/SPKI (*sudsy/spooky*)
  - Without naming
    - authorization/delegation
    - keys as principals
    - authorization certs, requests & tags, propagation control, acl's, threshold
  - Naming
    - local name spaces
    - name certs (defs)

SDSI/SPKI is "research in progress" by Ron Rivest, Butler Lampson, and Carl Ellison. It provides a framework for general client/server requests such as web page access. Canonical example: to control who accesses a page, a server makes decisions as to whether to honor a client request.

2 Main Focus: Authorization

- Client proves to server that he is authorized to make request. (shouldn't be too much work for sesrver; he handles a lot of requests) (see figure 1).
- Server maintains some objects. (has some Access Control List or database).
- Client proves he'ss a party in the ACL. E.g., by giving a digital signature
- Perhaps some dialogue: (see figure 2)

  Delegation:
- A party may delegate authority to someone else. e.g., hierarchy; organization $\rightarrow$ member. might have chain of certificate

2.1 Keys and Principles

It was decided *not* to go through classical, hierarchical global name structure. It seems to get in the way.

- Keys are the thing: they are the principles (not people, etc.). Keys are the "actors" in cyberspace.
- May have keyholders, but we don't talk about who those are.
- Key can make requests, have authority, etc. (assume some mechanism for figuring out who goes with what key). This goes with "no global naming."

3 Without Naming

3.1 Authentication

What is a request?

request: form filled in parameters digitally signed

We ignore what language these are in, and think of them as being *s-exprs*.

Examples of requests:

```
(read-file
    (file-name ``foo.bar''))

(open-dam
    (from 12:00)
    (to    1:00))
```

What's an ACL? acl: who are the parties (list of public keys) authorized to do what

To specify classes of s-exprs, we use a generalized s-expr:

```
(read-file (file-name (* prefix ``/www/bob'')))
```

```
(transfer
    (from-account (* set 123 456))
    (to-account 15)
    (amount (* range 1 1000)))
```

The wildcard operators (* prefix, * range, and * set) allow sets of s-exprs to be specified.

3.2 Delegation

acl is a kind of *delegation* from object owner to keyholder. We'd like recipient to be able to pass that on.

(e.g., k_0 gives something to k_1)

```
delegation-certificate:
    issuer: (k0)
    subject (recipient): (k1)
    tag: generalized s-expr
(signed by k0)
```

Bob might delegate less authority than he was given. (see figure 3)

Alice now wants to read the file. She has to convince the server to let her do it. Alice needs to present all certificates (chain) to server. She needs the complete chain, with effective tag

$$tag_0 \cap tag_1 \cap \ldots \cap tag_n$$

3.3 Propagation Control

Subscriptions to the Wall Street Journal, for example, don't want recipients to be able to keep delegating ability to read it.

Add field to delegation cert:

```
delegation-certificate:
    issuer:                 (k0)
```

```
    subject (recipient):    (k1)
    tag: generalized s-expr
    propagation: allowed/disallowed
(signed by k0)
```

The first acl is effectively isssued by the server to public key. (recip).

3.4 Thresholds

Some debate as to whether to do this; it's in the proposal.

The subject header above might read:

```
    subject: (threshold 2/3 k1, k2, k3)
```

Instead of a chain of certificates, you present a tree of certificates (see figure 4):

Alice shows how the keys fit in the tree to satisfy the threshold.

4 Naming

Up to this point, we have the complete picture, without naming. Not very auditable or human-friendly.

Back to names.

Historical sidebar: That stuff was SPKI (Simple Public Key Infrastructure — Carl Ellison) mostly. At the same time, Rivest and Lampson were working on the SDSI (Simple Distributed Security Infrastructure). Eventually, they started to combine the two.

Where does a user see the names?

1. acl: em who are parties
2. naming-cert: to figure out what names mean

Global names are hard. We'll use an engineering solution:

```
delegation-cert:
  issuer:
  subject: (could be name)
  ...
```

Names have contexts. Each key has an associated namespace, all potentially unrelated. (see figure 5)

Definitions of names need to be exportable.

```
definition-certificate:
    issuer: k0 Alice (Alice is the name being defined)
    subject (value): k1
(signed by k0)
```

This is a lot like the conventional hierarchicical name-certs. But they also allow: (see figure 6)

```
subject: k1 Mary
    that is, whatever k1's ``Mary'' means.

or

subject: k1 Mary mother
```

Can expire, can issue new ones. Can have several active at the same time, etc. (see figure 7)

k_0 = MIT
k_1 = eecs
k_2 = rivest
k_5 = rivest's secy

- acl
- k_0 mit-faculty secy
  (has multiple definitions, one of which is...)
- k_0 eecs-faculty secy

- k_1 rivest secy
- k_2 secy
- k_5

Names delegate ability to name parties.

groups = multiple definitions; use a theorem prover to derive them.

- Delegation, etc. through public key space
- Names to handle delegation in a user-friendly way, using definition certs.

Other details, issues:

- Revokation
- Query a principle to double-check if a cert is valid (mark cert as verifiable) ⋮

Note that the naming is a superset of global naming; for large-scale systems, you can implement hierarchies as trees.

Building an infrastructure is still open and interesting; need to worry about:

- Easy to use
- Convenient naming

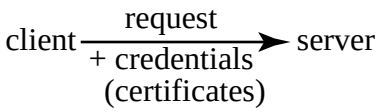

Figure 1: client request

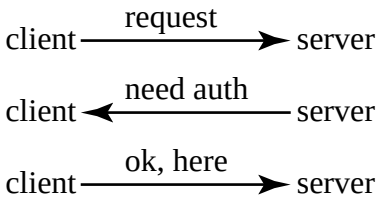

Figure 2: conversation

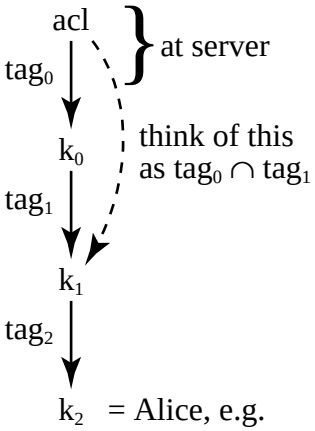

Figure 3: delegation

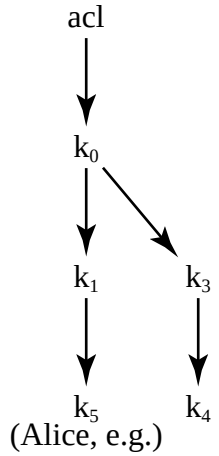

Figure 4: certificate tree

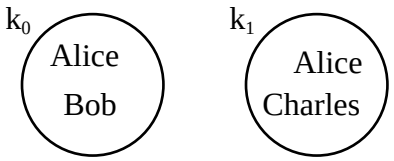

Figure 5: name context

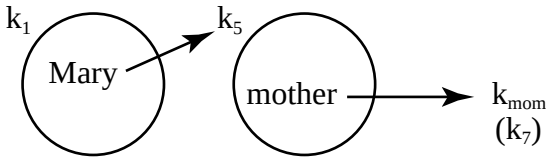

Figure 6: naming context

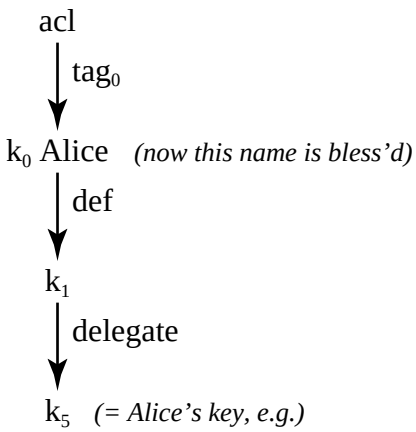

Figure 7: delegation