**CS489/589: Access Control & System Security**

**Lecture 5: Distributed Access Control and Trust Management**

---

**Distributed Access Control**

- Flexible and scalable access control in large-scale, open, distributed, decentralized systems
  - Electronic commerce
    - Transaction authorization
    - Application-level / business-policy authorization
  - Resource sharing in decentralized systems
    - Coalitions, multi-centric collaborative systems
    - Grid computing
  - Health care
  - and so on

**Characteristics**

- No central administration, each service makes its own decision
- No relationship between a service and a user prior to a request
  - Knowing a user’s name may not help
  - Must rely on information from third-party to make authorization decision (delegation)
- Authorization information is distributed
- Communication channels may be insecure

**Trust Management Approach**

- Multi-centric access control using delegation
  - Access control decisions are based on distributed policy statements issued by multiple principals
  - Policy statements contain
    - Attributes of principals such as permissions, roles, qualifications, characteristics
    - Trust relationships

**An Example**

- Bob, I delegate a permission to access File A, since I trust you
- Charles, I delegate a permission to access File A, since I trust you
- Alice, can I access to File A? - (O)

**Another Example**
Common Characteristics

- Use public-key certificates for non-local statements
- Treat public keys as principals to be authorized
  - Authentication consists of verifying signatures
- Adopt a peer model
  - An entity can be an authorizer, a requester, or a credential providers (trusted 3rd party)
- Treat the authorization decision problem as an application-independent proof-of-compliance problem

Public Key Certificates

- A certificate is a data record together with a digital signature
  - A certificate binds some information to public key
- A certificate is signed using $K^{-1}$
  - We say that it is issued by a public key $K$
- Can be verified by anyone who knows issuer’s public key
  - Can one trust the issuer’s public key?

Trust management

- TM components for proper functionality
  - Credentials
  - Compliance checker (TM engine)
  - Local security policy
- Access is granted if a chain of credentials proves that a requested action complies with local policies
  - Proposed TM systems and languages
    - PolicyMaker, Keynote, Referee, SPKI/SDSI, TPL, PMI, RT

Overview

- Context and history
- Motivation and goals
- Syntax
- Public keys (principals)
- Naming and certificates
- Groups and access control

Context

- Public-key cryptography invented in 1976 by Diffie and Hellman, enabling:
  - Digital signatures: private key signs, public key verifies.
  - Privacy: public key encrypts, 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 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
- Related to CA’s public key
  - How do I find out the CA’s public key (in an authentic manner)?
- Related to naming schemes
  - 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
- (X.509): Use a global hierarchy with one (or few) top-level roots:
- Use certificate chains (root to leaf):
- Names are also hierarchical: A/B/C/D

Scaling-up Problems, cont
- Global name spaces are politically and technically difficult to implement
- Nonetheless, a global hierarchical PK infrastructure began 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 THE application: should a digitally signed request (e.g. http request for a Web page) be honored?
SPKI/SDSI

- Simple Public Key Infrastructure
- Simple Distributed Security Infrastructure
- SDSI is effort by Butler Lampson and Ron Rivest 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

Simple Syntax

Byte-strings:
- abc (token)
- "Bob Dole" (quoted string)
- &4A5B70 (hexadecimal)
- 7Ra5 (base-64)
- #3:def (length:verbatim)
- abc- def = abdef (fragmentation)

Lists:
- (certificate (issuer bob) (subject alice))

Principals are Keys

- 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-verif object signature (const &03 (const &435affd1…)))))
- In practice, keys are often represented by their hash values

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 484..)) {signer (public-key …)} {signature 45632..})
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

- 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 Namespace

- 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

- 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> name1 ... namek)
  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 Chain**

- 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

---

**Online 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: named groups of principals

---

**Group Define Sets of Principal**

- 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 ACL 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 Object

- 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

Formal Framework to Understand TM

Traditional Access Control

Reference monitor mediates access.

Access matrix determines what is allowed.

TM Engine

TIME : Principal x Audit x P(Cert) --> Bool

(p:principal) TME p:principal (p:principal)
Principal, Auth, and AuthMap

- **Principal** = \{Bob, Charlie, ...\}
- **Auth** = \{RW, R, W, N\}, RW stands for an authorization under which Alice can read and write a file. Construct a lattice
- **AuthMap** = Principal → Auth, may represents that Bob authorize Alice to read/write the file, and Charlie to read the file

Example

```
Principal      Auth
  B             RW
  C             W
  M             N
```

License and Assertion

- **License** = AuthMap → Auth
- **License** function should be monotonic
- **Assertion** = Principal × License
- **Assertion** is the framework’s abstraction of digital certificate

Example

```
License      Assertion
  m(B)        Principal = {Bob, Charlie, ...}
  m(C)        Auth = {RW, R, W, N}, RW stands for an authorization under which Alice can read and write a file. Construct a lattice
```

Examples of Some Licenses

<table>
<thead>
<tr>
<th>License</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>λB.rw</td>
<td>Alice may write the file.</td>
</tr>
<tr>
<td>λC.r</td>
<td>Alice may read the file.</td>
</tr>
<tr>
<td>λD.r</td>
<td>Alice may read the file.</td>
</tr>
</tbody>
</table>

Framework

- **Components** →
  - Principal, Auth, AuthMap
  - License = AuthMap → Auth
  - Assertion = Principal × License
  - License function should be monotonic

Semantics

- **Semantics** →
  - $M_{\text{Assertion}} : \mathbb{P}(\text{Assertion}) \rightarrow \text{AuthMap}$
  - $M_{\text{Assertion}}(A) = \bigcup \{ l(m) \mid p, l \in A \}$

TM Engine

- **TM Engine** →
  - $M_{\text{Engine}} : \text{Principal} \times \text{Auth} \times \mathbb{P}(\text{Assertion}) \rightarrow \text{Bool}$
  - $M_{\text{Engine}}(p, u, A) = u \subseteq M_{\text{Assertion}}(A)(p)$

Semantics of Assertions

- **Semantics of Assertions** →
  - $M_{\text{Assertion}} : \mathbb{P}(\text{Assertion}) \rightarrow \text{AuthMap}$
  - $M_{\text{Assertion}}(A) = \bigcup \{ l(m) \mid p, l \in A \}$

Trust Management Engine

- **Trust Management Engine** →
  - $M_{\text{Engine}} : \text{Principal} \times \text{Auth} \times \mathbb{P}(\text{Assertion}) \rightarrow \text{Bool}$
  - $M_{\text{Engine}}(p, u, A) = u \subseteq M_{\text{Assertion}}(A)(p)$

If Alice would like to write the file under Bob’s control and the assertions $a_1, a_2, a_3$ are available, the trust engine would compute,

$$M_{\text{Engine}}(\text{Bob}, W, \{a_1, a_2, a_3\}) = W \subseteq M_{\text{Assertion}}(\{a_1, a_2, a_3\})(\text{Bob}) \rightarrow \text{true or false}$$
Instantiation of Framework

- Essentials
  - Auth lattice
  - Language for expressing licenses (monotone)
  - TM Engine