6.857 Computer and Network Security October 24, 2002

Lecture Notes 14 : Certificate chains

Lecturer: Ron Rivest *Scribe: Alwen/Burns/Ma*

1 Introduction

We begin with a quick overview of the structure of the Domain Name System, followed by an introduction to the X.509 public key infrastructure. After defining the goals for a PKI, SPKI/SDSI is introduced through a series (3) of attempts at designing a PKI. The first approach consists of ACLs which are essentially lists of PK's. Next, a first level of indirection is introduced, which leads to problems with PK's being maintained between domains. Finally, group structures are introduced, and the SPKI/SDSI PKI is presented. We end with a quick example of how such a PKI structure might look.

2 Outline

- DNS
- X.509
- SPKI/SDSI
  - Certificates
  - ACL's Naive Approach
  - ACL's with 1 Level of Indirection
  - ACL's with Groups (SPKI/SDSI)
  - Simple Model

3 Domain Name System (DNS)

DNS is the global system which maps names → IP addresses. (www.poledancing.com → 66.115.151.162 for example). DNS is structured as a tree where the root node points to servers which handle com, org, edu, ...

However such a structure is not ideal. For one thing, it has a single point of failure at the root node, as was demonstrated by the Denial of Service attack last Monday on the (13) root servers. About half of them went down. The DNS SEC project proposes to secure the DNS system.[1]

[0]May be freely reproduced for educational or personal use.

[1]see: http://www.nlnetlabs.nl/dnssec/

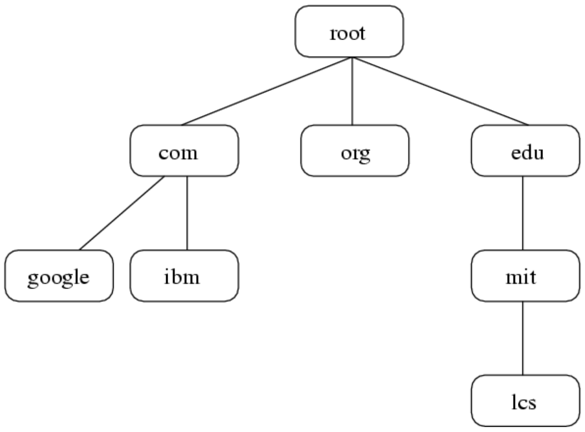

Figure 1: DNS-tree structure

4 Public Key Infrastructure and X.509

A PKI (such as X.509[2] for example) maps Names → public keys. X.509 is arranged as a strict tree structure. The first level is the most general, and each subsequent level provides greater detail. For example the first level (i.e. root) is an all encompassing node which points to the country nodes. Each country node then points to organization nodes, which in turn point to division nodes, which finally point to a specific name. The address of this leaf then consists of co=us/org=IBM/div=TJWatson/name=Don Coppersmith.

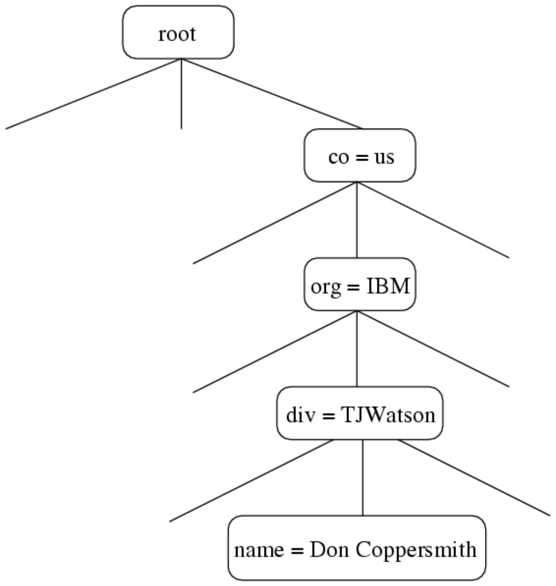

Figure 2: X.509-tree structure

[2]http://www.ietf.org/rfc/rfc2459.txt

5 Goals for a PKI

The following are goals for a PKI which were considered during the design of the SPKI/SDSI PKI.

- A PKI should not be Centralized, instead should be hierarchical with a top-down structure. $\rightarrow$ In other words a PKI should be distributed and decentralized, with a bottom-up design.
- A PKI should be flexible (should have groups).
- A PKI should be easy to use. (So it should use local names rather than global names.)

6 SPKI/SDSI

6.1 Certificates

The main cryptographic primitive employed by SPKI/SDSI is certificates. There are two kinds.

- "name certs" which define a local name.
- "authorization certs" which grant authorization.

Public keys *are* the principals, because only things signed by them can be recognized. Compare to X.509 for example, where users are the main objects. Names (also called "identifiers") are for the interface (so that people can remember something to refer to the keys), and can be chosen arbitrarily. Each PK has its own namespace, in which it can certify the validity of local names and bind them to a principal by producing a certificate to bind the name to a public key. Such namespaces are not related to each other. So if K_1 signs (A, PK_1) and K_2 signs (A, PK_2), then A has different definitions in the namespaces of K_1 and K_2. When we say K signs (A, PK) what is meant is that the corresponding secret key to K is used to produce a name cert. A name cert is a tuple (K, A, S, V) which is issued by the owner of K (i.e. has SK for K) where:

- `K A` is the local name being validated.
- S is the subject = definition of name. This may be a public key or another identifier.
- V is the validity period.
- `K A` $\rightarrow S$ is then signed by K.
- K_{LCS} can sign (`K rivest`, K_{rivest}) which would say "in K_{LCS}'s name space, `K rivest` is defined by the PK K_{rivest}".

6.2 Access Control – Naive Approach

In the naive approach, the Access Control List (ACL) = $\{PK_0, PK_1, \text{etc}\}$. In other words, the ACL is simply a list of public keys. A typical request for access might look something like this:

1. PK identifies itself (without proof) and requests access to a resource.
2. Guardian checks if PK is in the ACL.
3. If not it returns an error message, otherwise it returns the file encrypted with PK.

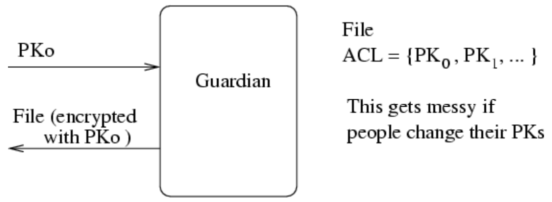
PKo
Guardian
File (encrypted with PKo)
File
ACL = {PK0 , PK1, ... }
This gets messy if people change their PKs

Figure 3: Access Control – Naive Approach

However this requires the guardian to maintain potentially unreasonably long lists of public keys for each resource and things become very messy quickly when users change their public keys.

6.3 Access control with 1 level of indirection

So to avoid the problems of needing to modify the ACL each time a user changes their public key, a level of indirection is introduced. This means that keys can now be assigned identifiers, so the ACL's are of the form $\{K_{LCS}$ `rivest`, K_{LCS} `kaashoek`, K_{IBM} `dom`, etc$\}$. Therefore, if a key changes, all that need be done is to issue a new certificate binding the new key to the same identifier, instead of changing the entire ACL. With this new scheme a typical request for a resource might look something like this:

1. K_{rivest} (= Rivest's public key) requests access to a resource.
2. Gaurdien responds "sorry, resource protected by ACL = ...".
3. K_{rivest} resends request (this time signed) along with proof that K_{rivest} is authorized, (K_{LCS} `rivest`, K_{rivest}) signed by K_{LCS}.

However, the guardian has to maintain other parties' PKs, e.g., LCS has to maintain PKs issued by IBM, MIT, etc. So to avoid this, an extra level of indirection is introduced. To include K_{MIT} `ron_rivest` in an ACL, the definition K_{LCS} `rivest` $\rightarrow$ K_{MIT} `ron_rivest` is created (by K_{LCS}). Now for K_{rivest} to get accepted by a guardian, a certificate chain of the form (K_{LCS} `rivest` $\rightarrow$ K_{MIT} `ron_rivest`) and (K_{MIT} `ron_rivest` $\rightarrow$ K_{rivest}) is needed. In other words, a chain beginning at the public key and ending in a valid identifier listed in LCS's ACL is required. A name can have multiple values (K_{LCS} `rivest` $\rightarrow PK_1$) and (K_{LCS} `rivest` $\rightarrow PK_2$)

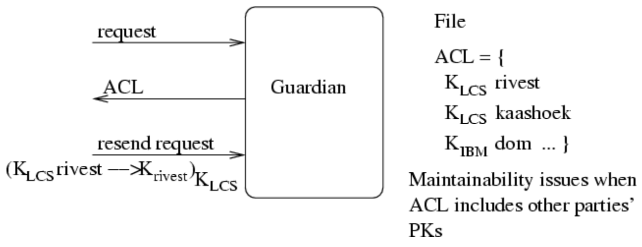
request
ACL
resend request
$(K_{LCS}$ rivest $\longrightarrow K_{rivest})_{K_{LCS}}$
Guardian
File
ACL = {
K_{LCS} rivest
K_{LCS} kaashoek
K_{IBM} dom ... }
Maintainability issues when ACL includes other parties' PKs

Figure 4: Access Control with 1 Level of Indirection

for example. Similarly, K_{LCS} `faculty` $\rightarrow$ K_{LCS} `rivest` means K_{LCS} `rivest` is a subset of K_{LCS} faculty.

This could lead to the problem of ACLs quickly becoming unreasonably long.

6.4 Access Control with Groups (SPKI/SDSI)

To solve this problem, the concept of groups is introduced. Using the fact that names can have multiple definitions, abstract names can be introduced forming groups (such as $K_{faculty}$). Thus a typical ACL may be as simple as ACL = {`faculty`}, and a typical request for access would look something like this:

1. PK sends a request.
2. Guardian responds by sending the ACL
3. PK sends the request again, this time signed with a SK along with a proof that PK is in fact included through some level of indirection in the ACL, in the form of a certificate chain beginning at PK and ending in a name in the ACL.

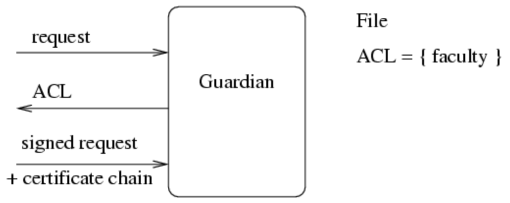

Figure 5: Access Control with Groups (SPKI/SDSI)

6.5 Simple Model

ACL = K_{LCS} faculty uses a 3-certificate chain:

1. K_{LCS} `faculty` $\rightarrow$ K_{LCS} `rivest`
2. K_{LCS} `rivest` $\rightarrow$ K_{MIT} `ron_rivest`
3. K_{MIT} `ron_rivest` $\rightarrow$ PK_0

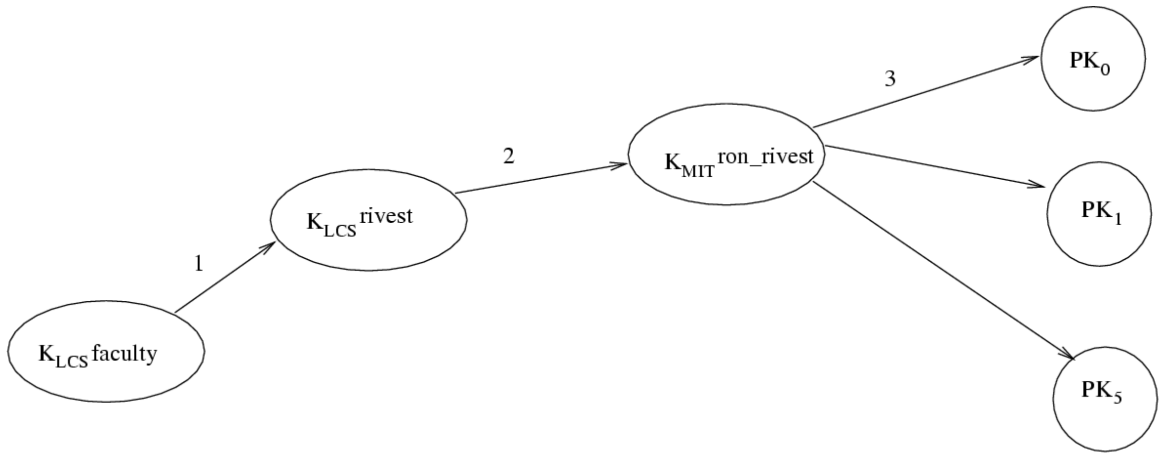

Figure 6: Simple Certificate Chain