Key Distribution - Secret Keys

- What if there are millions of users and thousands of servers?
- Could configure $n^2$ keys
- Better is to use a Key Distribution Center
  - Everyone has one key
  - The KDC knows them all
  - The KDC assigns a key to any pair who need to talk

Alice        KDC        Bob
A wants to talk to B
Randomly choose $K_{ab}$

\{"B", $K_{ab}$\}_{Ka}
\{"A", $K_{ab}$\}_{Kb}

\{Message\}_{K_{ab}}
A Common Variant

Alice          KDC          Bob
A wants to talk to B

Randomly choose $K_{ab}$

\[ \{ "B", K_{ab} \} \oplus \{ "A", K_{ab} \} \oplus \{ \text{Message} \} \]

KDC Realms

- KDCs scale up to hundreds of clients, but not millions
- There’s no one who everyone in the world is willing to trust with their secrets
- KDCs can be arranged in a hierarchy so that trust is more local

KDC Realms

Interorganizational KDC

Lotus KDC

SUN KDC

MIT KDC

A

B

C

D

E

F

G
KDC Hierarchies

- In hierarchy, what can each compromised KDC do?
- What would happen if root was compromised?
- If it's not a name-based hierarchy, how do you find a path?

Key Distribution - Public Keys

- Certification Authority (CA) signs “Certificates”
- Certificate = a signed message saying “I, the CA, vouch that 489024729 is Radia’s public key”
- If everyone has a certificate, a private key, and the CA’s public key, they can authenticate

KDC vs CA Tradeoffs

- Impact of theft of KDC database vs CA private key
- What needs to be done if CA compromised vs. if KDC compromised?
- What if KDC vs CA down temporarily?
- What’s more likely to work behind firewalls?
**Network Security**

**PKI**

### Strategies for CA Hierarchies

- One universally trusted organization
  - Top-Down, starting from a universally trusted organization's well-known key
  - No rules (PGP, SDSI, SPKI). Anyone signs anything. End users decide who to trust
  - Many independent CA's. Configure which ones to trust

### One CA

- Choose one universally trusted organization
  - Embed their public key in everything
  - Give them universal monopoly to issue certificates
  - Make everyone get certificates from them
  - Simple to understand and implement

### One CA: What’s wrong with this model?

- Monopoly pricing
  - Getting certificate from remote organization will be insecure or expensive (or both)
  - That key can never be changed
  - Security of the world depends on honesty and competence of the one organization, forever
One CA Plus RAs

- RA (registration authority), is someone trusted by the CA, but unknown to the rest of the world (verifiers).
- You can request a certificate from the RA
- It asks the CA to issue you a certificate
- The CA will issue a certificate if an RA it trusts requests it
- Advantage: RA can be conveniently located

What’s wrong with one CA plus RAs?

- Still monopoly pricing
- Still can’t ever change CA key
- Still world’s security depends on that one CA key never being compromised (or dishonest employee at that organization granting bogus certificates)

Oligarchy of CAs

- Come configured with 50 or so trusted CA public keys
- Usually, can add or delete from that set
- Eliminates monopoly pricing
Default Trusted Roots in IE

What’s wrong with oligarchy?

- Less secure!
  - Security depends on ALL configured keys
  - Naïve users can be tricked into using platform with bogus keys, or adding bogus ones (easier to do this than install malicious software)
- Although not monopoly, still favor certain organizations

CA Chains

- Allow configured CAs to issue certs for other public keys to be trusted CAs
- Similar to CAs plus RAs, but
  - Less efficient than RAs for verifier (multiple certs to verify)
  - Less delay than RA for getting usable cert
Anarchy
- Anyone signs certificate for anyone else
  - Like configured+delegated, but users consciously configure starting keys
  - Problems
    - Does not scale (too many certs, computationally too difficult to find path)
    - No practical way to tell if a path should be trusted
    - Too much work and too many decisions for user

Name Constraints
- Trustworthiness of a CA is not binary
  - Complete trust or no trust
  - CA should be trusted for certifying a subset of the users
  - Example:
    - Northeastern University CCS should (only) be trusted to certify users with name x@y.ccs.neu.edu
  - If users have multiple names, each name should be trusted by the "name authority"

Top Down with Name Subordination
- Assumes hierarchical names
  - Similar to monopoly: everyone configured with root key
  - Each CA only trusted for the part of the namespace rooted at its name
  - Can apply to delegated CAs or RAs
  - Easier to find appropriate chain
  - More secure in practice
    - This is a sensible policy that users don’t have to think about)
**Bottom-Up Model**

- Each arc in name tree has parent certificate (up) and child certificate (down)
- Name space has CA for each node in the tree
  - E.g., a certificate for .edu, neu.edu, and ccs.neu.edu
- "Name Subordination" means CA trusted only for a portion of the namespace
- Cross Links to connect Intranets, or to increase security
- Start with your public key, navigate up, cross, and down

---

**Intranet**

```
abc.com
  nj.abc.com
    alice@nj.abc.com
  ma.abc.com
    bob@nj.abc.com
carol@ma.abc.com
```

---

**Extranets: Crosslinks**

```
abc.com
  ---------------------
  xyz.com
```
Extranets: Adding Roots

Advantages of Bottom-Up
- For intranet, no need for outside organization
- Security within your organization is controlled by your organization
- No single compromised key requires massive reconfiguration
- Easy configuration: you start with is your own public key

Bridge CA Model
- Similar to bottom-up, in that each organization controls its destiny, but top-down within organization
- Trust anchor is the root CA for your org
- Your org’s root points to the bridge CA, which points to other orgs’ roots
Chain Building

- Call building from target "forward", and from trust anchor "reverse"
  - With the reverse approach it can be easier to find a path from the anchor to A by looking at the path
  - With the forward approach "going up" we don't know if a link/path starting at A leads to a trust anchor known by B

- Where should cert be stored?
  - With subject: harder to build chains from trust anchors
  - With issuer: it may become impractical if large fanout at root

X.509

- An authentication framework defined by ITU
- A clumsy syntax for certificates
  - No rules specified for hierarchies
  - X.509 v1 and v2 allowed only X.500 names and public keys in a certificate
  - X.509 v3 allows arbitrary extensions
- A dominant standard
  - Because it is flexible, everyone willing to use it
  - Because it is flexible, all hard questions remain
- C: country, CN: common name, O: organization, etc.

X.509 Certificate Contents

- version # (1, 2, or 3)
- Serial Number
- Effective Date
- Expiration Date
- Issuer Name
- Issuer UID (not in V1)
  - Unique ID
  - Subject Name
- Subject UID (not in V1)
- Subject Public Key Algorithm
- Subject Public Key
- Signature Algorithm
- Signature
- Extensions (V3 only)
### Some X.509 V3 Extensions

- **Public Key Usage**
  - Encryption
  - Signing
  - Key Exchange
  - Non-repudiation
- **Subject Alternate Names**
- **Issuer Alternate Names**
- **Key Identifiers**
- **Where to find CRL information**
- **Certificate Policies**
- **"Is a CA" flag**
  - path length constraints
  - name constraints
- **Subject Alternate Names**
- **Issuer Alternate Names**
- **Key Identifiers**
- **Where to find CRL information**
- **Certificate Policies**
- **"Is a CA" flag**
  - path length constraints
  - name constraints
- **Extended key usage**
  - specific applications

### Policies

- **A policy is an OID:**
  - Code Signing (1.3.6.1.5.5.7.3.3),
  - Windows Hardware Driver Verification (1.3.6.1.4.1.311.10.3.5)
- **Verifier specifies required OIDs**

### Policies (as envisioned by X.509/PKIX)

- **Policy is an OID (Object Identifier) e.g., top-secret, or secret**
- **Verifier says what policy OID(s) it wants**
- **Every link must have same policy in chain, so if verifier wants A or B or C, and chain has A, AC, ABC; B: not OK**
- **Policy mapping:** A=X; "want A" AB, A=X, X, X...
- **"Policy constraints" things like:**
  - policies must appear, but it doesn't matter what they are
  - "any policy" policy not allowed
  - any of these, but specified as taking effect n hops down chain
Other Certificate Standards

  - An IETF effort to standardize extensions to X.509 certificates
  - PKIX is a profile of X.509
  - Still avoids hard decisions, anything possible with PKIX

  - Simple Public Key Infrastructure
  - A competing IETF effort rejecting X.509 syntax

- SDSI: http://theory.lcs.mit.edu/~cis/sdsi.html
  - Simple Distributed Security Infrastructure
  - A proposal within SPKI for certificates with relative names only

Revocation Problem

- Suppose a bad guy learns your password or steals your smart card...
- Notify your KDC and it will stop issuing "tickets"
- Notify your CA and it will give you a new certificate
- How do you revoke your old certificate?

- Tickets can have short lifetimes; they can even be "one-use" with nonces
- Certificates have expiration dates, but it is inconvenient to renew them frequently
  - If sufficiently frequent and automated, CA can no longer be off-line
- Supplement certificate expirations with Certificate Revocation Lists (CRLs) or a blacklist server (On-Line Revocation Server: OLRS)
Why not put CA on-line?

- On-line revocation server is less security sensitive than an on-line CA
- The worst it can do is fail to report a revoked certificate
- Damage is more contained
- Requires a double failure
- With CRLs, limits OLRS damage

Revocation Ideas

- Incremental (delta) CRLs
- Micali’s hashing scheme
- Kaufman-Perlman “first valid cert”
- Good lists vs bad lists

Micali’s Hashing

- Components:
  - CA: generates/revokes certificates
  - Directory: gets daily updates from the CA and gets requests from users
  - It is not trusted
- Users
- Technique for efficient revocation:
  - CA generates:
    - Computes: \( Y_i = \text{Hash}(Y_{i-1}) \) and \( N_i = \text{Hash}(N_{i-1}) \), where \( Y_0 \) and \( N_0 \) are secret values
    - Certificate = signature of traditional info (e.g., public key, issue date, etc.) and \( Y_i \) and \( N_i \), 100 bit messages unique to the certificate, \( i \) is the certificate lifetime
  - Every day, the CA sends the directory:
    - \( Y_i \) or \( N_i \), depending on if the certificate is revoked or not
Authorization

- Access Control Lists (ACL) capabilities:
  - Makes a difference whether you can answer "who has access to that" or "what can he do"
- Groups, nesting, roles
- On-line group servers
- Anonymous groups

Suppose want to move subtrees?

- How would you design certificates if you want to be able to move an entire subtree, for example, com.sun.east.labs.radia becomes com.sun.labs.radia.
- What would up, down, and cross certs look like? How design cross link if want things not to change if both points move together?