# Key Management and Certificates

By the power vested in me I now declare this text and this bit string 'name' and 'key'. What RSA has joined, let no man put asunder

— Bob Blakley

# Key Management

Key management is the hardest part of cryptography

Actually, key management is the Achilles heel of cryptography

- None of the solutions that we have today work very well

Ignore the crypto and attack the key management

- Phishing is the best-known example of this

## Key Management Problems

Distributing keys

- How do I get Bob's key?
- How do I get my key to Bob?

Verifying the shared key

- Is this really Bob's key?

Key storage

- Where do I safely store Bob's key?

Validity checking

- Is the key that I have for Bob still valid?
- My key's been compromised, how do I inform Bob?

## Key Lifetimes

Two classes of keys

- Short-term session keys (sometimes called ephemeral keys)
  - Generated automatically and invisibly
  - Used for one message or session and discarded
- Long-term keys
  - Generated explicitly by the user

Long-term keys are used for two purposes

- Authentication (including access control, integrity, and non-repudiation)
- Confidentiality (encryption)
  - Establish session keys
  - Protect stored data

## Key Lifetimes (ctd)

Authentication keys

- Public keys may have an extremely long lifetime (decades)
- Private keys/conventional keys have shorter lifetimes (a year or two)

Confidentiality keys

- Should have as short a lifetime as possible

Effects of a key compromise

- Authentication: Signed documents are rendered invalid unless countersigned/timestamped
- Confidentiality: All data encrypted with it is compromised

## Key Management Models

Offline mechanisms

- Hierarchical certificates (X.509)
- Distributed certificates (PGP)

Online mechanisms

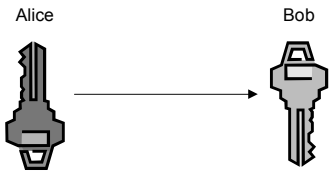
- On-demand distributions (SSL/TLS, S/MIME)
- Key continuity (SSH)

Ad-hoc ("fit for a purpose") methods

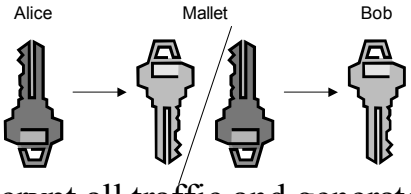
- Too many to list
- Standard mechanisms don't work very well, so people create their own ad-hoc ones

# Key Distribution

Alice retains the private key and sends the public key to Bob




Mallet (man-in-the-middle, MITM) intercepts the key and substitutes his own key




Mallet can decrypt all traffic and generate fake signed message

# Key Distribution (ctd)

For some mechanisms, almost all of the effort has been focused on addressing the man-in-the-middle (MITM) problem

- Others address further aspects of the key managt.life cycle

Committee design

- Address the MITM problem
- No specific goal, so add more and more complexity to the middle

Application-specific design

- Address the actual needs of the application
- Need to solve a particular problem, so design a workable domain-specific solution

## Key Use Controls

Very complex topic

- The term "policy" features prominently

"Use a smart card/HSM" is not the answer

- "Application" = "code that does anything an untrusted PC tells it to"
- "Smart card" = "device that does anything an untrusted PC tells it to"

Need to actually think about what you're trying to protect, and protect against

## Key Use Controls (ctd)

Example: Low-value key

- Standard software protection measures are about the best you can do
- Storage on a USB key creates a ritual in which the user is required to consciously apply the key
  - Rituals are very important in real life, e.g. notarised signatures (see Digital Signature Legislation slides)
  - In practice, it'll stay permanently plugged in, but it's the thought that counts

## Key Use Controls (ctd)

Example: High-value (infrequently-used) key

- Lock a laptop in a bank vault
- Bank is contractually bound to enforce access controls, two-person control, access auditing, etc
- Laptop is configured to only perform the authorised operation, e.g. CA certificate signing
  - Insert USB key, enter access code(s), new certificate is deposited on key, remove key
- Goes beyond straight technical security to create a complete ritual commensurate with the seriousness of the key use

## Key Backup/Archival

Need to very carefully balance security vs. backup requirements

- Every extra copy of your key is one more failure point
- Communications and signature keys never need to be recovered — generating a new key only takes a minute or so
- Long-term data storage keys should be backed up

*Never* give the entire key to someone else

- By extension, never use a key given to you by someone else (e.g. generated for you by a third party)

## Key Backup/Archival (ctd)

Use a threshold scheme to handle key backup

- Break the key into *n* shares
- Any *m* of *n* shares can recover the original
- Shares can be reconstructed under certain conditions (e.g. the death of the owner)

Defeating this setup requires subverting multiple shareholders

Key shares are virtually never used in practice

- Impossible to render comprehensible to users
- Huge amounts of effort
- Easier to just generate a new key

## Key Destruction

Ensure that all copies of a private key are destroyed

- Is *every* copy really gone?

Public keys may need to survive private keys by quite some time

- Signature on a 20-year mortgage

Long-term key ownership can be a thorny issue

- CA goes bankrupt and auctions off its keys
  - c.f. bankrupt dot-coms selling user lists after they promised not to
  - Only asset the CA had left
  - Bidding quickly shot up to rather high values
- Do you want a third-party CA issuing your corporate certs?

## Online Solution: Key Continuity

Continuity = knowing that what you're getting now is what you've had before/what you were expecting

- McDonalds food is the same no matter which country you're in
- Coke is Coke no matter what shape bottle (or can) it's in, or what language the label is in

Continuity is more important than third-party attestation

- Equivalent to brand loyalty in the real world
- Businesses place more trust in established, repeat customers

Use continuity for key management

- Verify that the current key is the same as the one that you got previously

## Key Continuity in SSH

First app to standardise its key management this way

- On first connect, client software asks the user to verify the key
  - Done via the key fingerprint, a hash of the key components
  - Standard feature for PGP, X.509, …
- On subsequent connects, client software verifies that the current server key matches the initial one
  - Warn the user if it's changed

Concept was formalised in the resurrecting duckling security model

- Device imprints on the first item that it sees
- Device trusts that item for future exchanges

## Key Continuity in SIP

Same general model as SSH

- First connect exchanges self-signed certificates
- Connection is authenticated via voice recognition

Same principle has been used in several secure IP-phone protocols

- Users read a hash of the session key over the link

## Key Continuity in STARTTLS et al

SMTP/POP/IMAP servers are usually configured by sysadmins unconcerned about browser warning dialogs

- Remember the initial certificate, warn if it changes
- Using a self-signed certificate avoids having to pay a CA

The ideal key continuity solution

- Automatically generate a self-signed certificate on install
- Use key continuity to warn if the certificate changes

## Key Continuity in STARTTLS et al (ctd)

Currently still somewhat haphazard

- Many open-source implementations support it fully
- Some still require tedious manual operations for certificate management
- Commercial implementations often require CA-issued certificates, an even more tedious (and expensive) manual operation

## Key Continuity in S/MIME

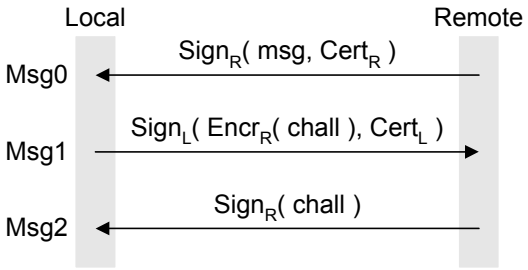
S/MIME has a built-in mechanism to address the lack of a PKI

- Include all signing certificates in every message you send
- Lazy-update PKI distributes certificates on an on-demand basis
  - SSL/TLS does something similar

S/MIME gateways add two further stages

- Auto-generate certificates for new users
- Perform challenge-response to verify any new certificates that they encounter (see next slide)

# Key Continuity in S/MIME (ctd)




Msg0 gets remote certificates to the local server (as standard S/MIME message)

Msg1 gets local certificates to the remote user

Msg2 proves possession of remote server keys/certificates

(Variants, e.g remote server sends challenge in Msg3)

# Key Continuity in S/MIME (ctd)

Provides mutual proof of possession of keys and certificates to both sides

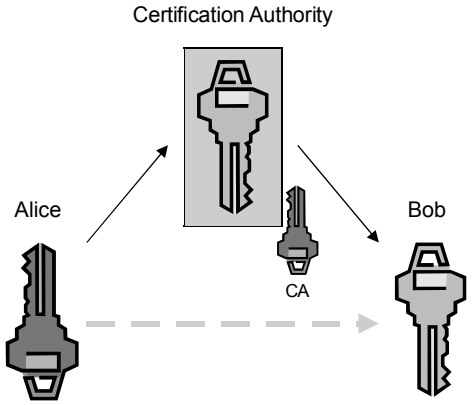
- In practice has a few extra tricks to avoid various attacks, e.g. using the other side as an oracle
- Both parties now have verified keys for the other side
- Fully automatic, no human intervention required

Outlook for the future

- Invented/reinvented as needed by implementors
  - Not specified in any formal standard
    - Standards groups are still waiting for PKI to start working
  - Present in many apps, but needs standardisation to unify the approaches

## Offline Solution: Certificates

Certification authority (CA) solves the MITM problem




CA signs Alice's key to guarantee its authenticity to Bob

- Mallet can't substitute his key since the CA won't sign it

## Certification Authorities

A certification authority (CA) guarantees the connection between a key and an end entity

An end entity is

- A person
- A role ("Director of marketing")
- An organisation
- A pseudonym
- A piece of hardware or software
- An account (bank or credit card)

Some CAs only allow a subset of these types

## Role of a CA

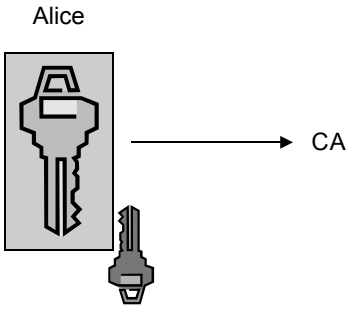
Original intent was to certify that a key really did belong to a given party

Role was later expanded to certify all sorts of other things

- Are they a bona fide business?
- Can you trust their web server?
- Can you trust the code they write?
- Is their account in good standing?
- Are they over 18?

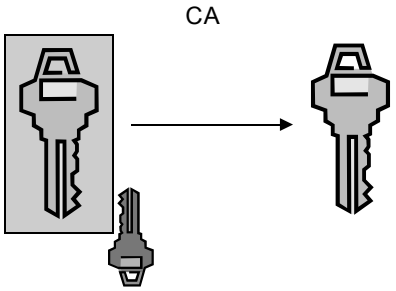
When you have a certificate-shaped hammer, everything looks like a nail

## Obtaining a Certificate (X.509)



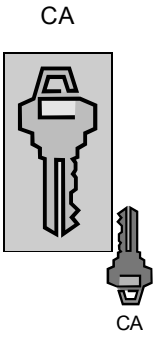

1. Alice generates a key pair and signs the public key and identification information with the private key
   - Proves that Alice holds the private key corresponding to the public key
   - Protects the public key and ID information while in transit to the CA

## Obtaining a Certificate (X.509) (ctd)



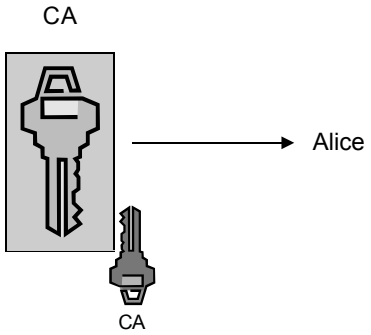

2. CA verifies Alice's signature on the key and ID information and recovers the original key
   - Optional: CA verifies Alice's ID through out-of-band means
     – email/phone callback
     – Business/credit bureau records, in-house records

## Obtaining a Certificate (X.509) (ctd)




3. CA signs the public key and ID with the CA key, creating a certificate
   - CA has certified the binding between the key and ID

## Obtaining a Certificate (X.509) (ctd)




4. Alice verifies the key, ID, and CA's signature
   - Ensures that the CA didn't alter the key or ID
   - Protects the certificate in transit
5. Alice and/or the CA publish the certificate

## Obtaining a Certificate (X.509) (ctd)

Most of these steps are undefined

- How does Alice locate a CA?
- How does Alice get the CA's key?
- How does Alice get her public key to the CA?
- How does the CA verify Alice's information?
- How does Alice get her certificate back from the CA?

Protocols exist for submitting keys and obtaining the certificate response (CMP, CMC), but no-one uses them

- Too little too late

General practice is to kludge something together with a combination of Google, HTTP PUT/GET, and cut & paste from email messages

## Obtaining a Certificate (PGP, SSH)

PGP: Generate a key

- Enter your name and email address at key generation time

SSH: Generate a key

## Certificate History

To understand the X.509 PKI, it's necessary to understand the history behind it

Why does X.509 do otherwise straightforward things in such a weird way?

> [The] standards have been written by little green monsters from outer space in order to confuse normal human beings and prepare them for the big invasion
>
> — comp.std.internat

- Someone tried to explain public-key-based authentication to aliens. Their universal translators were broken and they had to gesture a lot
- They were created by the e-commerce division of the Ministry of Silly Walks

## Certificate History (ctd)

Original paper on public-key encryption proposed the Public File

- Public-key white pages
- Key present → key valid
- Communications with users were protected by a signature from the Public File

Signatures (certificates) were a one-time assertion about keys in the public file

- "This key is valid at this instance for this person"

## Certificate History (ctd)

A very sensible, straightforward approach…

- … today
- We have a Public File, it's called the world-wide web

  We have a system, it is called the Web, everyone else lost, get over it

  — Phillip Hallam-Baker, Chief Scientist, Verisign

However, this wasn't so practical in 1976

- Key lookup over X.25?

Adapted for offline operation by Kohnfelder in 1978

- Offline CA signs name + key to bind the two in a certificate
- Online directory distributes certificates

## Certificate History (ctd)

OSI proposed (among many other things) X.500, an all-encompassing global directory run by monopoly telcos

- Hierarchical database (or data organisation model, or both)
- Path through the directory/database to keys is defined by a series of relative distinguished names (RDNs)
- Collection of RDNs form a distinguished name (DN)
- Data being looked up is found at the end of the RDN path

X.500 is a bunch of networking types trying to re-invent 1960s database technology

— Lynn Wheeler

## The X.500 Directory

The directory contains multiple objects in object classes defined by schemas

A schema defines

- Required attributes
- Optional attributes
- The parent class

| | | |
|---|---|---|
| Object | Attribute | Value |
| | Attribute | Value |
| | Attribute | Value |

Attributes are type-and-value pairs

- Type = CN, value = John Doe
- Type may have multiple values associated with it
- Collective attributes are attributes shared across multiple entries (e.g. a company-wide fax number)

# The X.500 Directory (ctd)

Each instantiation of an object is a directory entry

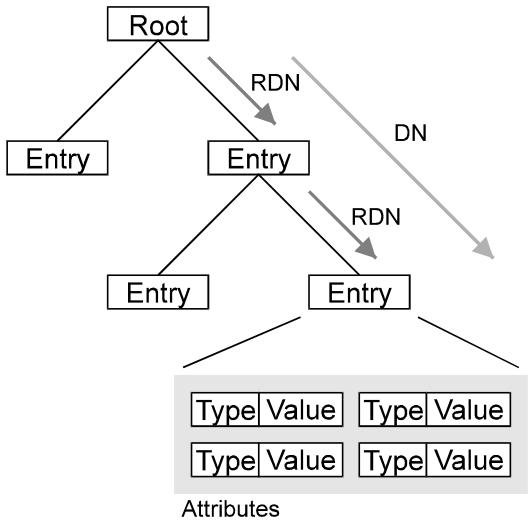
Entries are identified by DNs

- The DN is comprised of relative distinguished names (RDNs) that define the path through the directory

Directory entries may have aliases that point to the actual entry

The entry contains one or more attributes that contain the actual data

# The X.500 Directory (ctd)




Data is accessed by DN and attribute type

## X.500 DN Example

Typical DN components

- Country C
- State or province SP
- Locality L
- Organisation O
- Organisational unit OU
- Common name CN

Typical X.500 DN

C=US/L=Area 51/O=Hanger 18/OU=X.500 Standards Designers/CN=John Doe

- When the X.500 revolution comes, your name will be lined up against the wall and shot

## Searching the Directory

Searching is performed by subtree refinement

- Base specifies where the search starts in the subtree
- Chop specifies how much of the subtree to search
- Filter specifies the object class to filter on

Example

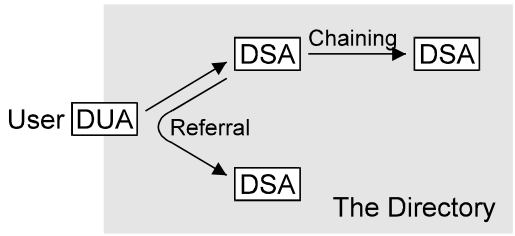
- Base = C=NZ
- Chop = 1 RDN down from the base
- Filter = organisation

Typical application is to populate a tree control for directory browsing

```
SELECT name WHERE O = *
```

## Directory Implementation

The directory is implemented using directory service agents (DSAs)




Users access the directory via a directory user agent (DUA)

- Access requests may be satisfied through referrals or chaining

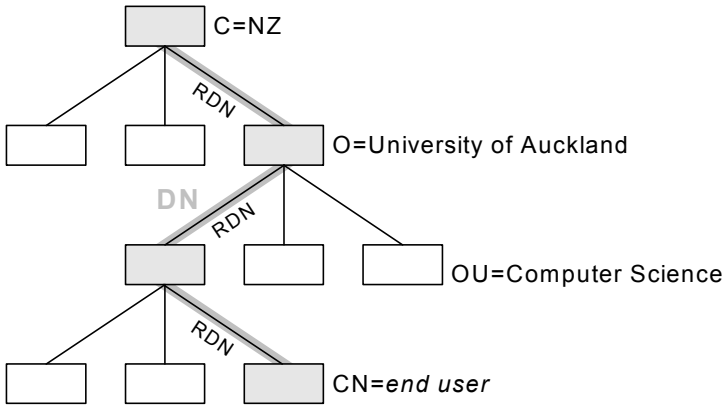
One or more DSAs are incorporated into a management domain

## Directory Access

Typical directory accesses:

- Read attribute or attributes from an entry
- Compare supplied value with an attribute of an entry
- List DNs of subordinate entries
- Search entries using a filter
  - Filter contains one or more matching rules to apply to attributes
  - Search returns the attribute or attributes that pass the filter
- Add a new leaf entry
- Remove a leaf entry
- Modify an entry by adding or removing attributes
- Move an entry by modifying its DN

# Directory Example




Search key is C=NZ, O=University of Auckland, OU = Computer Science, CN = foo

- Complex way of saying `SELECT data WHERE key = 'foo'`

# Directory Access Control

Concerns about misuse of the directory

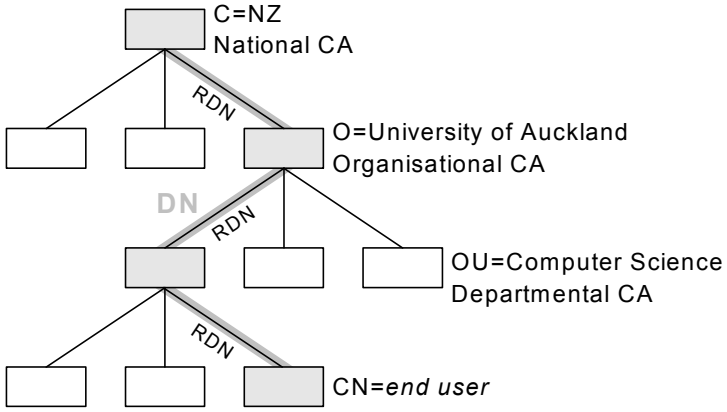
- Companies don't like making their internal structure public
  - Directory for corporate headhunters
- Privacy concerns
  - Directory of single women (then)
  - Directory of children (now)

X.500 proposed various access control mechanisms

- Passwords
- Hashed passwords
- Digital signatures

# Directory Access Control

For signature-based access control, each portion of the directory has a certification authority (CA) attached to it




Top-level CA is called the root CA, a.k.a. "the single point of failure"

# Directory Access Control

X.509v1 clearly shows these origins

- Issuer and subject DN to place a cert in the directory
- Validity period
- Public key

No indication of…

- CA vs. end entity certs
  - Implicit from their position in the directory
- Key usage
  - Only one usage, directory authentication
- Cert policy
  - Only one policy, directory authentication
- Any of the other X.509v3 paraphernalia

## Directory Access Control

No directories of this type were ever seriously deployed

- We've had to live with the legacy of this approach ever since

## LDAP

X.500 Directory Access Protocol (DAP) adapted for Internet use

- Originally Lightweight Directory Access Protocol, now closer to HDAP

Provides access to LDAP servers (and hence DSAs) over a TCP/IP connection

- `bind` and `unbind` to connect/disconnect
- `read` to retrieve data
- `add`, `modify`, `delete` to update entries
- `search`, `compare` to locate information

## LDAP (ctd)

LDAP provides a complex hierarchical directory containing information categories with sub-categories containing nested object classes containing entries with one or more (usually more) attributes containing actual values

- In one large-scale interop test the use of a directory for cert storage was found to be the single largest cause of problems

Simplicity made complex

"It will scale up into the billions. We have a pilot with 200 users running already"

## LDAP (ctd)

Most practical way to use it is as a simple database

```
SELECT key WHERE name = 'Bob'
```

LDAP equivalent query

```
S(&(|(&(objectclass=inetorgperson)
  (objectclass=organizationalperson))
  (objectClass=StrongAuthenticationUser))
  (usercertificate;binary=*)
  (|(commonname=name)(rfc822mailbox=email
  address)))
```

## LDAP (ctd)

> So far, LDAP has not done a great job of supporting PKI requirements
>
> — Steve Kent, PKIX WG chair
>
> The X.500 linkage […] has led to more failed PKI deployments in my experience than any other. For PKI deployment to succeed you have to take X.500 and LDAP deployment out of the critical path
>
> — Phillip Hallam-Baker, Chief Scientist, Verisign

If you can't be a good example then at least you can be a horrible warning

## Horowitz Key Protocol (HKP)

PGP's key lookup protocol

- Exists as an undocumented student project that requires reverse-engineering the source code to understand
- Handled via a number of "well-known" servers run by volunteers

HTTP-based key search, upload, and download over port 11371

- Various extensions hacked on over time

Probably the most successful online key distribution mechanism (!!)

- (Which shows just how dysfunctional the others are)

## Offline Key Storage Mechanisms

Keys are typically stored offline in

- Flat files
  - One per key
  - Multiple keys per file (PGP keyrings, SSH)
- Relational databases
- Proprietary databases (Netscape/Mozilla)
- Windows registry (MSIE)

## Problems with Naming/Identity Certificates

"The user looks up John Smith's certificate in a directory"

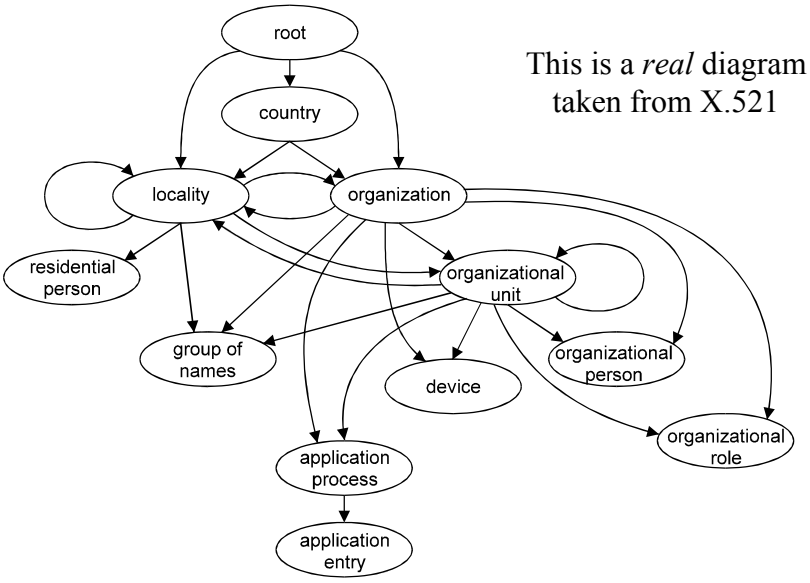
- Which directory?
- Which John Smith?

X.509-style PKI turns a key distribution problem into a name distribution problem

- Cases where multiple people in same O, OU have same first, middle, and last name
- Solved by adding some distinguishing value to DN (e.g. part of the social security number, SSN)
  - Creates unique DNs, but they're useless for name lookups
  - John Smith 8721 vs. John Smith 1826 vs. John Smith 3504

# Problems with X.500 Names

No-one ever managed to figure out how to make DNs work




This is a *real* diagram taken from X.521

# Problems with X.500 Names (ctd)

No clear plan on how to organise the hierarchy

- Attempts were made to define naming schemes, but nothing really worked

No-one understands X.500 DNs

- Locality? Organisational unit? Administrative domain?
- Don't fit any real-world domain
- Require extensive versing in X.500 theology to comprehend

Hierarchical naming model fits the military and governments, but doesn't work for businesses or individuals

- DNs provide the illusion of order while preserving everyone's God-given Freedom to Build a Muddle

## Problems with X.500 Names (ctd)

Simple problem cases

- Communal living (jails, boarding schools)
- Nomadic peoples
- Merchant ships
- Quasi-permanent non-continental structures (oil towers)
- US APO addresses
- LA phone directory contains > 1,000 people called "Smith" in a nonexistent 90000 ZIP code
  - A bogus address is cheaper than an unlisted number
  - Same thing will happen on a much larger scale if people are forced to provide information when they don't want to
  - c.f. cypherpunks login

## Problems with X.500 Names (ctd)

For a corporation, is C, SP, L

- Location of company?
- Location of parent company?
- Location of field office?
- Location of incorporation?

For a person, is C, SP, L

- Place of birth?
- Place of residence/domicile?
  - Dual citizenship
  - Stateless persons
  - Nomads
- Place of work?

## DNs in Practice

Public CAs typically set

C = CA country or something creative ("Internet")

O = CA name

OU = Certificate type / class / legal disclaimer

CN = User name or URI

email = User email address

Some European CAs add oddball components required by local signature laws

- Italy adds IDs like BNFGRB46R69A944C
- This is a value that's both impossible for anyone to know while simultaneously leaking lots of personal data to anyone viewing a certificate

## DNs in Practice (ctd)

Some CAs modify the DN with a nonce to try and guarantee uniqueness

- Armed services CA adds last 4 digits of SSN
- Another CA encodes random CA/RA-specific data

  The disambiguating factor will be variable length alphanumeric […] for example: XYZ221234 […] or, for example ABC00087654321.

  — GTE Government Systems Federal PKI pilot

Some DNs are deliberately mangled

> For educational institutions here in the US, FERPA regulations apply. The way we do this here at Wisconsin is to only include a bunch of random gibberish in the DN as an identifier.
>
> — Eric Norman on ietf-pkix

## DNs in Practice (ctd)

Private CAs (organisations or people signing their own certificates) typically set any DN fields supported by their software to whatever makes sense for them

- Some software requires that all of { C, O, OU, SP, L, CN } be set
  - "Invent random values to fill these boxes in order to continue"
- Resulting certificates contain strange or meaningless entries as people try and guess values, or use dummy values
  - "… a bunch of random gibberish in the DN…"

## Solving the DN Problem

Users put whatever they feel like into the DN

> The goal of a certificate [identifier] is to identify the holder of the corresponding private key in a fashion meaningful to relying parties.
>
> — Steve Kent, PKIX WG chair

- Minimalist DNs
  - "Fred's Certificate"
  - "My key"
  - "202.125.47.110"

## Solving the DN Problem (ctd)

In practice only the CN and/or URI matter

- Email address
- Web server address/name
- Internet host address/name

Rest isn't checked (or even displayed) by most PKI-using applications

- SSL/TLS information displayed in browser status bar, address displayed in email client, …

## If you must use a DN…

General layout for a business-use DN

Country + Organisation + Organisational Unit + Common Name

– C = New Zealand
  O = Dave's Wetaburgers
  OU = Procurement
  CN = Dave Taylor

General layout for a personal-use DN

Country + State or Province + Locality + Common Name

– C = US
  SP = California
  L = San Francisco
  CN = John Doe

## Non-DN Names

X.509 v3 added support for other name forms

- email addresses
- DNS names
- URLs
- IP addresses
- EDI and X.400 names
- Anything else (type + value pairs)
- These add-ons are largely ignored in favour of the DN though

For historical reasons, email addresses are often stuffed into DNs rather than being specified as actual email addresses

## Other Identification Approaches

PGP: Used for email encryption

- Identity is name + email address

SPKI: Used for authorisation/access control

- Identity is a name meaningful within the domain of application
  - Account name on a server
  - Credit card number
  - Merchant ID

SSH and SSL/TLS: Indentity is implicit

- "Whatever the server gives me when I connect"

All also use the public key as a unique ID

## Qualified Certificates

Certificate designed to identify a person with a high level of assurance

Precisely defines identification information in order to identify the cert owner in a standardised manner

- Defines additional parameters such as key usage, jurisdiction where the certificate is valid, biometric information, etc
- Qualified certificates only apply to natural persons

Some jurisdictions don't allow this type of unique personal identifier

- Any government that can issue this type of identifier can create un-persons by refusing to issue it

## Qualified Certificates (ctd)

Allows use of a pseudonym

- Pseudonym must be registered, i.e. can be mapped to a real name via an external lookup
- Most implementations assume that every DN contains a CN, so some approximation to a CN must be supplied even if a pseudonym is used

Defines personalData, a new subjectAltName subtype

- Registration authority for personal data information
- Collection of personal data
  - Full (real, not DN) name, gender
  - Date and place of birth
  - Country of residence and/or citizenship
  - Postal address

## Qualified Certificates (ctd)

In practice, qualified certificates are issued to companies by fudging the issuing process

- Issue the certificate to an alias for a company
  - Use the ability for a QC to contain a pseudonym
- Does a complete end-run around the QC legal requirements

Uncertainty as to what demand is actually being met by Qualified Certificates

- Probably a "round up twice the usual number of suspects" response to the lack of success of standard certificates
- The near-universal response to the lack of success of PKI standards is to create more standards

## PGP Certificates

Certificates are key-based, not identity-based

- Keys can have one or more free-form names attached
- Key and name(s) are bound through (independent) signatures

Certification model can be hierarchical or based on existing trust relationships

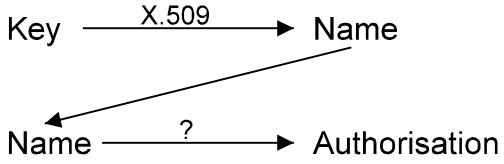
- Parties with existing relationships can use self-signed certificates
  - Self-signed end entity certificates are a logical paradox in X.509v3

Authentication keys are used to certify confidentiality keys

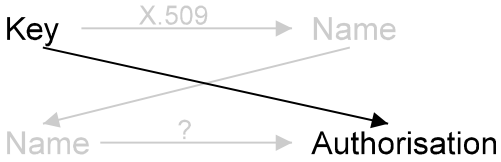
- Confidentiality keys can be changed at any time, even on a per-message basis

## Simple Public Key Infrastructure (SPKI)

Identity certificates bind a key to a name, but require a parallel infrastructure to make use of the result




SPKI certificates bind a key to an authorisation or capability




## SPKI (ctd)

Certificates may be distributed by direct communications or via an online distribution mechanism

Each certificate contains the minimum information required for the job (c.f. X.509 dossier certificates)

- Hearkens back to the original Public File one-time assertions

If names are used, they only have to be locally unique

- Global uniqueness is guaranteed by the use of the key as an identifier
- Certificates may be anonymous (e.g. for balloting)

Authorisation may require *m* of *n* consensus among signers (e.g. any 2 of 3 company directors may sign)

## SPKI Certificate Uses

Typical SPKI uses

- Signing/purchasing authority
- Letter of introduction
- Security clearance
- Software licensing
- Voter registration
- Drug prescription
- Phone/fare card
- Baggage claim check
- Reputation certificate (e.g. Better Business Bureau rating)
- Access control (e.g. grant of administrator privileges under certain conditions)

## SPKI Certificate Structure

SPKI certificates use collections of assertions expressed as LISP-like S-expressions of the form ( *type value(s)* )

( name fred ) ⇒ Owner name = fred

( name *CA_root CA1 CA2 ... CAn leaf_cert* ) ⇒ X.500 DN

( name ( hash sha1 |TLCgPLFlGTzyUbcaYLW8kGTEnUk=| ) fred ) ⇒ Globally unique name with key ID and locally unique name

( ftp ( host ftp.warez.org ) ) ⇒ Keyholder is allowed FTP access to an entire site

( ftp ( host ftp.warez.org ) ( dir /pub/warez ) ) ⇒ Keyholder is allowed FTP access to only one directory on the site

## SPKI Certificate Structure (ctd)

Can even emulate X.509 using an SPKI certificate

```
( cert
    ( issuer ( hash sha1 |TLCgPLFlGTzyUbcaYLW8kGTEnUk=|
      ) )
    ( subject ( hash sha1 |Ve1L/7MqiJcj+LSa/l10fl3tuTQ=l| ) )
    ...
    ( not-before "1998-03-01_12:42:17" )
    ( not-after "2012-01-01_00:00:00" )
  ) ⇒ X.509 certificate
```

## SPKI Certificate Structure (ctd)

Internally, SPKI certificates are represented as 5-tuples
<Issuer, Subject, Delegation, Authority, Validity>

- Issuer, subject are identified via the global key ID
- Delegation = Subject has permission to delegate authority
- Authority = Authority granted to the certificate subject
- Validity = Validity period and/or online validation test information

## SPKI Trust Evaluation

5-tuples can be automatically processed using a general-purpose tuple reduction mechanism

<I1, S1, D1, A1, V1> + <I2, S2, D2, A2, V2>
⇒ <I1, S2, D2, intersection( A1, A2 ), intersection( V1, V2 )
if S1 = I2 and D1 = true

Eventually some chains of authorisation statements will reduce to <Trusted Issuer, *x*, D, A, V>

- All others are discarded

## SPKI Trust Evaluation (ctd)

Example authorisation chain

- A may access resource X. Signed: Service Provider
- B may access resource X. Signed: A
- Service provider, please allow me to access X. Signed: B

Verification

- Service provider checks signatures from B → A → own key
- Authorisation loop requires no CA, trusted third party, or external intervention
- Trust management decisions can be justified/explained/verified
  - "How was this decision reached?"
  - "What happens if I change this bit?"

X.509 has nothing even remotely like this

## SPKI Redux

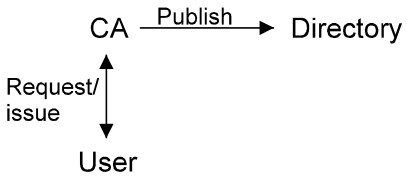
SPKI's greatest feature is that it's not X.509

- SPKI's greatest flaw is that it's not X.509

SPKI was an excellent solution for the authorisation problem, but because it wasn't X.509 it never made any headway

## CAs and Scaling

The standard certification model involves direct user interaction with a CA

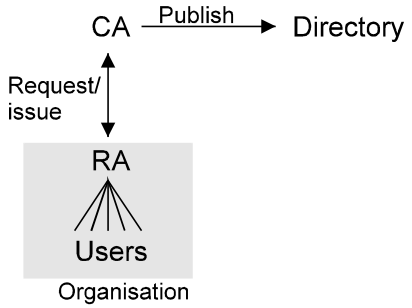



This doesn't scale well

- CA has to verify the details for each user
- Processing many users from a similar background (e.g. a single organisation) results in unnecessary repeated work

# RAs

Registration authorities offload user processing and checking from the CA




RA acts as a trusted intermediary

- RA has a trusted relationship with the CA
- RA has access to user details

# RAs (ctd)

Everyone wants to be the CA

- Stamps out certificates and collects fees

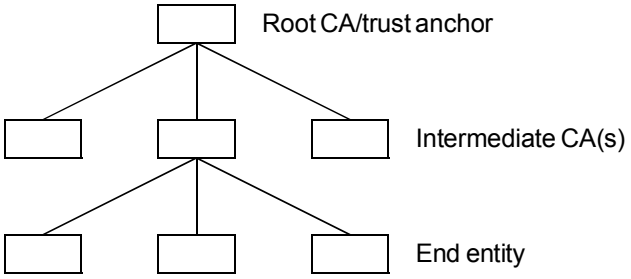
Someone else has to be the RA

- Does all the checking and other work

Rarely used in practice

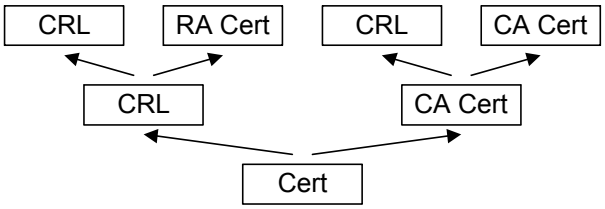
# Certificate Chains

Collection of certificates from a leaf up to a root or trust anchor




# Certificate Chains (ctd)

All previous problems are multiplied by the length of the chain




- Complexity of certificate checking is proportional to the square of the depth of the issuance hierarchy

## Cross-Certification

Original X.500-based scheme envisaged a strict hierarchy rooted at the X.500 directory root

- PEM tried (and failed) to apply this to the Internet

Later work had large numbers of hierarchies

- Many, many flat hierarchies
- Every CA has a set of root certificates used to sign other certificates in relatively flat trees

What happens when you're in hierarchy A and your trading partner is in hierarchy B?

## Cross-Certification (ctd)

Solution: CAs cross-certify each other

- A signs B's certificate
- B signs A's certificate

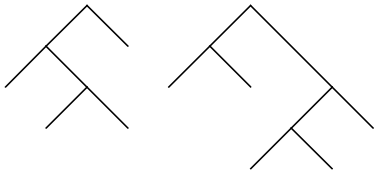
Problem: Each certificate now has *two* issuers

- All of X.509 is based on the fact that there's a unique issuer

Toto, I don't think we're in X.509 any more

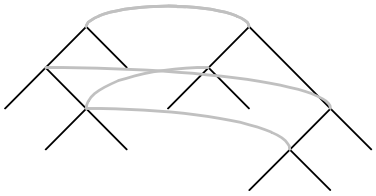
## Cross-Certification (ctd)

With further cross-certification, re-parenting, subordination of one CA to another, revocation and reissuance/ replacement, the hierarchy of trust…

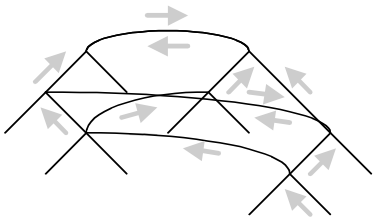


## Cross-Certification (ctd)

…becomes the spaghetti of doubt…



…with multiple certificate paths possible



## Cross-Certification (ctd)

Different CAs and paths have different validity periods, constraints, etc etc

- Certificate paths can contain loops
- Certificate semantics can change on different iterations through the loop
- Are certificate paths Turing-complete?
- No software in existence can handle these situations

Cross-certification is the black hole of PKI

- All existing laws break down
- No-one knows what it's like on the other side

## Cross-Certification (ctd)

The theory: A well-managed PKI will never end up like this

- "If it does occur, we can handle it via nameConstraints, policyConstraints, etc etc"

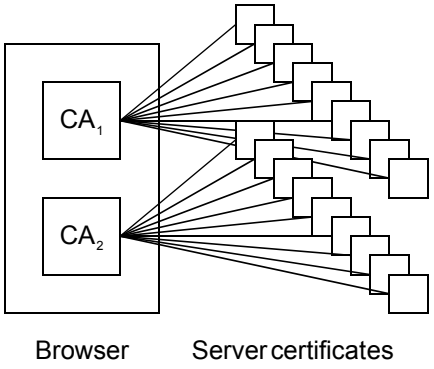
The practice: If you give them the means, they will build it

- Allow cross-certification and it's only a matter of time before the situation will collapse into chaos
- c.f. CA vs. EE certificates
  - There are at least 5 different ways to differentiate the two
  - Only one of these was ever envisaged by X.509
- Support for name and policy constraints is dubious to nonexistant
  - Playing Russian roulette with your security

# Cross-Certification in Browsers

CAs are hard-coded into browsers

- Implicitly trusted
- Totally unknown entities
  - CA keys have been on-sold to third parties when the original CA went out of business
- Moribund web sites
- 512-bit keys
- 40-year cert lifetime (!!)
- How much would you trust a "NO LIABILITY ACCEPTED" CA?




# Cross-Certification in Browsers (cont)

All CA certificates are trusted equally

- Implicit universal cross-certification

Any CA can usurp a certificate issued by any other CA

- Certificate from "Verisign Class 1 Public Primary Certification Authority" could be issued by "Honest Al's Used Cars and Certificates"
- Browser trusts Verisign and Honest Al equally

Accepting one single new certificate from a web site can break the entire browser security model

I'm in ur browser forgin ur certs

## Cross-Certification in Browsers (cont)

CAs are often presented as "trusted third parties"

- They're usually just plain "third parties" (Scott Rea)
  - The user has no basis for trusting them
- Often, they're explicitly untrusted third parties
  - The user has no idea who they are

> This must be some strange new use of the word 'trust' with which I wasn't previously familiar
>
> — Arthur Dent (almost)

Overall security is that of the least trustworthy CA

> Anyone who selects a public CA on a factor other than price fails to understand the trust models that underlie today's use of CAs
>
> — Lucky Green

## Cross-Certification in Browsers (cont)

Disabling all of these certificates…

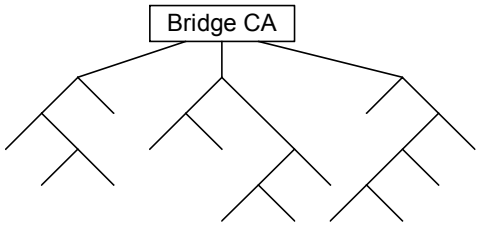
- Netscape 6: ~600 mouse clicks
- MSIE 6: ~700 mouse clicks

Why do browsers do this?

- Prime directive: Don't expose the users to scary warning dialogs
- One-size-fits-all browser can't know in advance which entities the user has a trust relationship with
  - Need to include as many certificates as possible to minimise the chances of users getting scary warning dialogs
- The ideal user-friendly situation would be to automatically trust all certificates

# Bridge CAs

Attempt to solve the cross-certification chaos by unifying disparate PKIs with a super-root

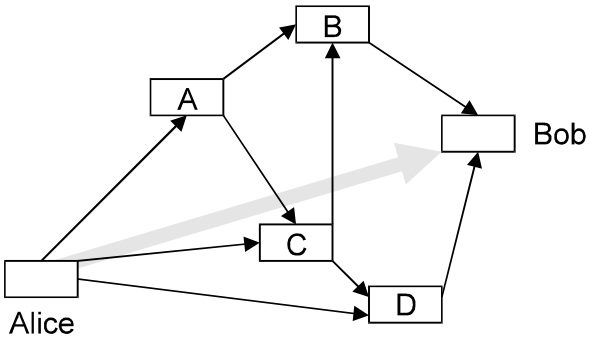



Still has problems

- PKIn root has different semantics than bridge root
- What if PKI1 = CIA, PKI2 = KGB, PKI3 = Mossad?

# PGP Web of Trust

Bob knows B and D who know A and C who know Alice
⇒ Bob knows the key came from Alice




Web of trust more closely reflects real-life trust models

- In practice it doesn't really deliver though

## Certificate Validity Checking

The target: A yes/no response in as close to real time as possible

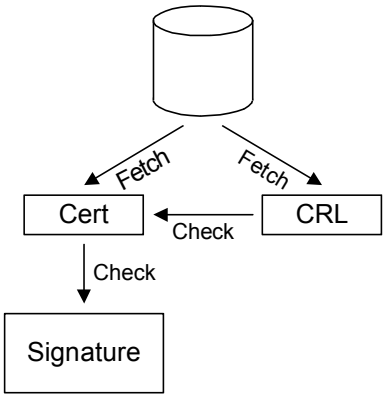
- Follows the standard credit card model of checking: "accepted", "declined"

For assorted historical and religious reasons, almost no PKI actually supports this basic valid/not valid check

## X.509 Certificate Usage Model

Relying party wants to verify a signature

- Fetch certificate
- Fetch certificate revocation list (CRL)
- Check certificate against CRL
- Check signature using certificate



## X.509 Certificate Usage Model (ctd)

X.509 model turns certificates into capabilities

- Tickets that can be used for authorisation/access control purposes
- Capabilities can be passed around freely
- Revocation is very hard

Tried to address revocation with…

- Replacing the cert with a new one
- Notifying the owner "by some off-line procedure"
- Certificate revocation lists (CRLs), a blacklist of revoked certs
- Assorted handwaving

## Certificate Revocation

Revocation is managed with a certificate revocation list (CRL), a form of anti-certificate that cancels a certificate

- Equivalent to 1970s-era credit card blacklist booklets
  - Based on even earlier cheque blacklists
- Relying parties are expected to check CRLs before using a certificate
  - "This certificate is valid unless you hear somewhere that it isn't"

## CRL Problems

CRLs mirror credit card blacklist problems

- Not issued frequently enough to be effective against an attacker
- Expensive to distribute
- Vulnerable to simple DoS attacks
  - Attacker can prevent revocation by blocking delivery of the CRL

Blacklist approach was abandoned by credit card vendors a quarter of a century ago because it didn't work properly

## CRL Problems (ctd)

CRLs add further problems of their own

- Can contain retroactive invalidity dates
- CRL issued right now can indicate that a cert was invalid last week
  - Checking that something was valid at time *t* isn't sufficient to establish validity
  - Back-dated CRL can appear at any point in the future

Destroys the entire concept of nonrepudiation

## CRL Problems (ctd)

Two schools of thought on handling revocation dates in CRLs

- Accuracy School: Date reflects the time of key compromise even if it's backdated
- Consistency School: Date is the date of CRL issue even if it's after the key compromise was detected/reported

PKI standards groups are split roughly 50:50 on this

## CRL Problems (ctd)

Why CRLs don't work

- Violate the cardinal rule of data-driven programming

  "Once you have emitted a datum you can't take it back"
- In transaction processing terms, viewing a certificate as a PREPARE and a revocation as a COMMIT
  - No action can be taken between the two without destroying the ACID properties of the transaction
  - Allowing for other operations between the PREPARE and the COMMIT results in nondeterministic behaviour

## CRL Problems (ctd)

CA cert revocation is more difficult than end-entity revocation

- One interop test found that revoking a CA cert would require a "system rebuild"
  - Replace the current PKI software with updated software
  - (Reformat and reinstall)
- Testing of CA cert revocation was deferred until later

## CRL Problems (ctd)

Revoking self-signed certificates is even hairier

- Cert revokes itself
- Applications may
  - Accept the CRL as valid and revoke the certificate
  - Reject the CRL as invalid since it was signed with a revoked certificate
  - Crash
- Computer version of Epimenides paradox "All Cretans are liars"
  - Crashing is an appropriate response

# CRL Problems (ctd)

## CRL Distribution Problems

- CRLs have a fixed validity period
  - Valid from *issue date* to *expiry date*
- At *expiry date*, all relying parties connect to the CA to fetch the new CRL
  - Massive peak loads when a CRL expires (DDoS attack)
- Issuing CRLs to provide timely revocation exacerbates the problem
  - 10M clients download a 1 MB CRL issued once a minute = ~150 GB/s traffic
  - Even per-minute CRLs aren't timely enough for high-value transactions with interest calculated by the minute

# CRL Problems (ctd)

## Clients are allowed to cache CRLs for efficiency purposes

- CA issues a CRL with a 1-hour expiry time
- Urgent revocation arrives, CA issues an (unscheduled) forced CRL before the expiry time
- Clients that re-fetch the CRL each time will recognise the cert as expired
- Clients that cache CRLs won't
- Users must choose between huge bandwidth consumption/ processing delays or missed revocations

## Kludging around CRL Problems

Segment CRLs based on the urgency of revocation

- "Key compromise" issued once a minute
- "Affiliation changed" issued once a day
- Possible attacks
  - Substitute one CRL for another
  - Attacker can place a key on a low-priority CRL before the victim can place it on high-priority CRL

Delta CRLs

- Short-term CRLs that modify a main CRL
- Discussion on PKI mailing lists indicates that use of delta CRLs will be an interesting experience

## Kludging around CRL Problems (ctd)

Stagger CRLs

- Over-issue CRLs so that multiple overlapping CRLs exist at one time
- Timeliness guarantees vanish
- Plays havoc with CRL semantics
  - Cert may or may not appear on any of several CRLs valid at a given time

## Kludging around CRL Problems (ctd)

Real problem is that a blacklist approach to validity checking (in any security context, not just PKI) is fundamentally broken

- "Enumerating Badness" is classed as the No.2 Dumbest Idea in Computer Security (Marcus Ranum)
- No amount of kludging can fix it

## Bypassing CRLs

SET sidesteps CRL problems entirely

- End user certificates are "revoked" by cancelling the credit card
- Merchant certificates are "revoked" by marking them as invalid at the acquiring bank
- Payment gateways have short-term certificates that can be quickly replaced

Account Authority Digital Signatures (AADS/X9.59)

- Public key is tied to an existing account
- Revocation is handled by removing the key
- Matches the 1970s model of certificates: "This key is valid at this instant for this account"

# Online Revocation Checking

Many applications require prompt revocation

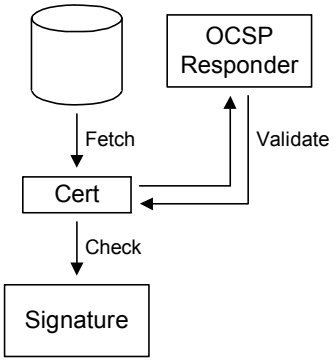
- CAs (and X.509) don't really support this
- CAs are inherently an offline operation

Requirements for online checks

- Should return a simple boolean value "Certificate is valid/not valid right now"
- Can return additional information such as "Not valid because …"
- Historical query support is also useful, "Was valid at the time the signature was generated"
- Should be lightweight (c.f. CRLs, which can require fetching and parsing a 10,000 entry CRL to check the status of a single certificate)

# Online Status Checking

Online Certificate Status Protocol, OCSP




- Inquires of the issuing CA whether a given certificate is still valid
  - Acts as a simple responder for querying CRLs
  - Still requires the use of a CA to check validity

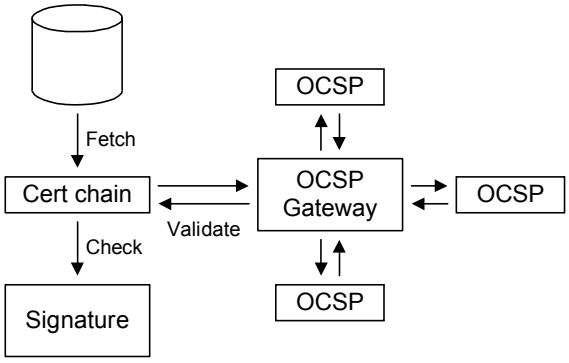
# OCSP

Acts as a selective CRL query protocol

- Standard CRL process: “Send me a CRL for everything you’ve got”
- OCSP process: “Send me a pseudo-CRL/OCSP response for only these certs”
  - Lightweight pseudo-CRL avoids CRL size problems
- Reply is created on the spot in response to the request
  - Ephemeral pseudo-CRL avoids CRL validity period problems

# OCSP (ctd)

Can be used with an access concentrator to handle chains




- Gateway does all the work
- Requests can be forwarded to further gateways
- User is billed once at the access concentrator

## OCSP (ctd)

Returned status values are non-orthogonal

- Status = "not revoked", "revoked", or "unknown"
- "Not revoked" doesn't mean "good", merely "not on the CRL"
- "Unknown" could be anything from "Certificate was never issued" to "It was issued but I can't find a CRL for it"
- If asked "Is this a valid cert" and fed…
  - A freshly-issued cert, can't say "Yes"
  - An MPEG of a cat, can't say "No"
- Compare this with the credit card authorisation model
  - Response is "authorised" or "declined" (with optional reasons)

OCSP was designed to be fully bug-compatible with CRLs

## OCSP (ctd)

Problems are due in some extent to the CRL-based origins of OCSP

- CRL can only report a negative result
- "Not revoked" doesn't mean that a cert was ever issued
- Some OCSP implementations will report "I can't find a CRL" as "not revoked"
- Some relying party implementations will assume that "revoked" $\Rightarrow$ "not good", so any other status = "good"
- Much debate among implementors about OCSP semantics

## OCSP (ctd)

OCSP has no scalability

- Vendors eliminate replay-attack protection in order to get usable performance

  The changes we are making to scale our OCSP responder will result in the discontinuation of the nonce extension

  — Verisign
- This breaks OCSP's security (anyone can forge/replay status responses), but at least you can claim that you have a workable responder now

## Other Online Revocation Protocols

Simple Certificate Validation Protocol (SCVP)

- Relying party submits a full chain of certificates
- Server indicates whether the chain can be verified
- "We can't solve the problem here so we'll hand it off to someone else"

Data Validation and Certification Server Protocols (DVCS)

- Provides facilities similar to SCVP disguised as a general third-party data validation mechanism

Integrated CA Services Protocol (ICAP)

Real-time Certificate Status Protocol (RCSP)

Web-based Certificate Access Protocol (WebCAP)

## Other Online Revocation Protocols (ctd)

Open CRL Distribution Protocol (OpenCDP)

Directory Supported Certificate Status Options (DCS)

Data Certification Server (also DCS)

Delegated Path Validation (DPV)

- Offshoot of the SCVP/DVCS debate and an OCSP alternative called OCSP-X

Many, many more

- See earlier comment about "fixing" the failure of PKI standards by creating more PKI standards
- Protocol debate has been likened to religious sects arguing over differences in dogma

## Cost of Revocation Checking

CAs charge fees to issue a certificate

- Most expensive collection of bits in the world

Revocation checks are expected to be free

- CA can't tell how often or how many checks will be made
- CRLs require
  - Processor time
  - Multiple servers (many clients can fetch them)
  - Network bandwidth (CRLs can get large)
- Active disincentive for CAs to provide any real revocation checking capabilities

## Cost of Revocation Checking (ctd)

Example: ActiveX

- Relatively cheap cert can sign huge numbers of ActiveX controls
- Controls are deployed across hundreds of millions of Windows machines
- Any kind of useful revocation checking would be astronomically expensive

Example: email certificate

- Must be made cheap (or free) or users won't use them
- Revocation handling isn't financially feasible

## Cost of Revocation Checking (ctd)

Revocation checking in these cases is, quite literally, worthless

- Leave an infrequently-issued CRL at some semi-documented location and hope few people find it

Charge for revocation checks

- Allows certain guarantees to be associated with the check
- Identrus charges for every revocation check (i.e. certificate use)
- GSA cost was 40¢…$1.20 each time a certificate was used
- Sweden charges €0.25 (inspired by the cost of a postage stamp)

Serious user acceptance problems

- Why pay a transaction fee for a mostly useless certificate when other online transactions carry no (obvious) fees?

## Status Checking in the Real World

Online protocols place an enormous load on the CA

- CA must carefully protect their signing keys

… but …

- CA must be able to sign *x*,000 status requests per second
- CRL is inherently a batch operation
  - Once an hour, scan a database table and sign the resulting list
- Online status protocols have a high processing overhead
  - For each query, check for a revocation and produce a signed response
  - By their very nature, it's not possible to pre-generate responses, since they must be fresh
    - Well, except for the OCSP "scalability" hack

## Status Checking in the Real World (ctd)

CA key compromise: Everyone finds out

- Sun handled revocation of their CA key via posts to mailing lists and newsgroups

SSL server key compromise: No-one finds out

- Stealing the keys from a typical poorly-secured server isn't hard (c.f. web page defacements)
- Revocation isn't necessary since certificates are included in the SSL handshake
  - Just install a new certificate

email key compromise: Who cares?

- If necessary, send a copy of your new certificate to everyone in your address book

## Example: SSL Server Certificate

Using the X.509 revocation status codes as usage cases

- Key compromise → unlikely to be used unless the attacker helpfully notifies the victim
- Affiliation changed → handled automatically when the new certificate for the new name is issued
- Superseded → as above
- Cessation of operation → shut down the server

In none of these cases is revocation actually needed, or useful

## Status Checking in the Real World (ctd)

In practice, revocation checking is often turned off in user software

- Serves no real purpose, and slows everything down a lot

CRLs are useful in special-case situations where there exists a statutory or contractual obligation to use them

- Relying party needs to be able to claim CRL use for due diligence purposes or to avoid liability

PKI researchers like to tinker with revocation in the same way that petrol-heads tinker with car engines

- Geek's dream problem: No clear solution, no pressure to produce a result, endless funding is available

## Other Revocation Techniques

### Self-signed revocation (suicide note)

- Used by PGP
- Suicide note is created as part of the key generation process
- Not possible in X.509 because it's something you can't do without requiring a CA

### Certificate of health/warrant of fitness for certificates (anti-CRL)

- Provides better proof than CRLs
  - CRL is a negative statement
  - Anti-CRL is a positive statement
- Proving a negative is much harder than proving a positive
- c.f. "aliens don't exist" vs. "aliens exist"

## Other Online Authorisation Checks

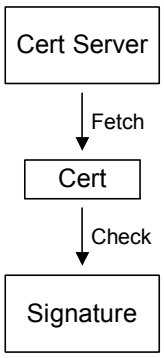
### Simple Public Key Infrastructure (SPKI)

- Prefers online authorisation/validation checks
  - This is a true online authorisation check, not the OCSP silly-walk
- Cert renewal interval is based on risk analysis of potential losses
  - X.509 renewal interval is usually one year, motivated by billing concerns
  - Treated like a domain name: Once a year, re-certify the same key
- Provides for one-time renewal
  - Cert is valid for a single transaction

## Closing the Circle

Fetching a cert and then immediately having to perform a second fetch to determine whether it's any good is silly

- Fetch a known-good cert (no revocation check necessary)
- Solves the previous revocation-checking problems
- Simplify further: Submit a hash of the certificate on hand
  - "It's good, go ahead and use it"
  - "It's no good, use this one instead"




## Closing the Circle (ctd)

All we really care about is the key

- Issuer/subject DN, etc are historical artifacts/baggage

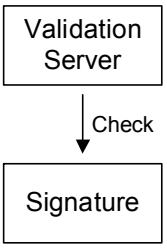
"Bring me the key of Alfredo Garcia"

- This operation is already performed locally when the key is fetched from a certificate store/Windows registry/flat file
- Moving from a local to a remote query allows centralised administration

## Closing the Circle (ctd)

Key-fetch is still an unnecessary step

- Validation server performs the check directly
- Similar to the 1970s Davies and Price model
  - Arbitrator provides a dispute resolution mechanism via a one-time interactive certificate for the transaction
- Fits the banking/online settlement transaction model




## Setting up a CA

No-one makes money running a CA

- You make money by selling CA products and services

Typical cost to set up a proper CA from scratch: $1M

Writing the policy/certificate practice statement (CPS) requires significant effort

Getting the top-level certificate (root certificate) installed and trusted by users can be challenging

# Bootstrapping a CA

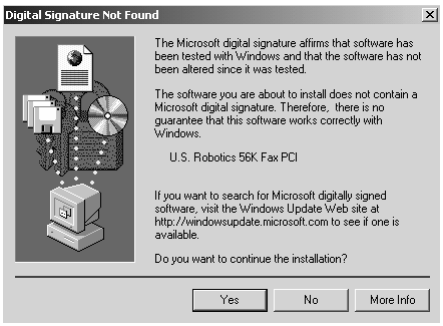
Get your root certificate signed by a known CA

- Your CA's certificate is certified by the existing CA
- Generally requires becoming a licensee of the existing CA
- Your CA is automatically accepted by existing software
  - But see the section on CA survivability

# Bootstrapping a CA (ctd)

Get users to install your CA certificate in their applications

- Somewhat cumbersome for users to do
- However, no more complex than the procedure for installing an (unsigned) driver for new hardware
  - Half the Windows drivers on the planet have unsigned drivers
  - Google for "digital signature not found"




- If users are expecting the certificate install dialog and know where to click, it works pretty well

## Bootstrapping a CA (ctd)

Publish your CA certificate(s) by traditional means

- Global Trust Register, `http://www.cl.cam.ac.uk/Research/Security/Trust-Register/`
- Book containing a register of fingerprints of the world's most important public keys
- Implements a top-level CA using paper and ink
- Requires massive amounts of manual user intervention though

Install custom software containing the certificate on user PCs

- Less transparent than manually installing CA certificates

## Business Expectations of a CA

Current work follows the "if you build it, they will (might) come" model

- Industry (particularly governments) make great testbeds for PKI experimentation
  - They'll even pay you for it!

Survey of US businesses revealed that they require CAs to be insurable

- Must be possible to quantify risk reliably enough to make meaningful warranties
- c.f. Verisign's null-semantics certificates

## Business Expectations of a CA (ctd)

Two approaches to this problem:

1. Practical solution: CA has only two warranted responsibilities
    1. Ensure that each name is unique
    2. Protect the CA's key(s)
        – Interpreting the certificate is left to the relying party
2. Legal solution: If you do *x*, the government will indemnify you
    - *x* expands to "jump through all the hoops defined in this digital signature law"
    - Type, size, and number of hoops varies from country to country

## Finding a Workable Business Model

PKI requires of the user

- Certificate management software to be installed and configured
- Payment for each certificate
- Significant overhead in managing keys and certificates

PKI provides to the user

- "…disclaims any warranties... makes no representation that any CA or user to which it has issued a digital ID is in fact the person or organisation it claims to be... makes no assurances of the accuracy, authenticity, integrity, or reliability of information"

## Finding a Workable Business Model (ctd)

A PKI is not just another IT project

- Requires a combined organisational, procedural, and legal approach
- Staffing requires a skilled, multidisciplinary team
- Complexity is enormous
  - Initial PKI efforts vastly underestimated the amount of work involved
  - Current work is concentrating on small-scale pilots to avoid this issue

To be accepted, a PKI must provide perceived value

- Failure to do so is what killed SET
- No-one has really figured out a PKI business model yet

## CA Business Model

Free email certs

- No-one will pay for them
- Clown suit certificates

SSL certificates run as a protection racket

- Buy our certs at US$500/kB/year or your customers will be scared away
- Actual CA advertising:

  If you fail to renew your Server ID prior to the expiration date, operating your Web site will become far riskier than normal […] your Web site visitors will encounter multiple, intimidating warning messages when trying to conduct secure transactions with your site. This will likely impact customer trust and could result in lost business for your site.

CA consulting services

## Getting your CA Key into Browsers

Total cost: $0.5M per browser

- Netscape: Hand over the cash and a floppy
- MSIE: No special charge, but you must pass an SAS70 electronic data security audit
  - US CPA Statement on Auditing Standards 70
  - Lengthy (up to 6 months), expensive, and painful
  - Infrastructure, policy, staff, and auditing costs run to $0.5M

CA keys are bought and sold on the secondary market

- Equifax's certificates are actually owned by Geotrust
- Cheaper to buy another CA's HSM than to have your own key added
- Keep an eye out for CA HSMs on eBay

## CA Survivability

Something you'll never find in any PKI text: What happens when your CA fails?

- "PKI or Bust"
- The majority of the commercial CAs have failed

Where do their keys end up?

- (eBay or equivalent)
- Those keys control access to your data and your infrastructure

How is business continuity handled?

- Example: Large commercial CA fails
- No-one left knows how to issue CRLs

## Timestamping

Certifies that a document existed at a certain time

Used for added security on existing signatures

- Timestamped countersignature proves that the original signature was valid at a given time
- Even if the original signature key is later compromised, the timestamp can be used to verify that the signature was created before the compromise

Requires a data format that can handle multiple signatures

- Only PGP keys and S/MIME signed data provide this capability

## PKI Design Guidelines

Identity

- Use a locally meaningful identifier
  - User name
  - email address
  - Account number
- Don't try and do anything with DNs
  - Treat them as meaningless blobs

## PKI Design Guidelines (ctd)

### Revocation

- If possible, design your PKI so that revocation isn't required
  - SET
  - AADS/X9.59
  - ssh
  - SSL/TLS
- If that isn't possible, use a mechanism that provides freshness guarantees
- If that isn't possible, use an online status query mechanism
  - Valid/not valid responder
  - OCSP
- If the revocation is of no value, use CRLs

## PKI Design Guidelines (ctd)

### Control

- Any externally-controlled link in the certification chain can control any assets covered by your PKI
- What happens when an external CA goes out of business?
  - Where do the CA's keys go?
  - Who issues new certificates?
  - Who revokes old certificates?

## PKI Design Guidelines (ctd)

### Application-specific PKIs

- PKIs designed to solve a particular problem are easier to work with than a one-size-(mis)fits all approach
- One-size-fits-all approach is only useful to verify well-known entities
  - amazon.com et al
  - Banks
  - Government departments
- Application-specific approaches work better for everything else
  - Use the same channels to verify Bob's key as you use to verify other transactions with Bob
- Third-party CAs merely get in the way

## PKI Design Guidelines (ctd)

### Application-specific PKIs

- SPKI
  - Binds a key to an authorisation
  - X.509 binds a key to an (often irrelevant) identity that must then somehow be mapped to an authorisation
- PGP
  - Designed to secure email
  - Laissez-faire key management tied to an email address solves the "Which directory" and "Which John Doe" problems

## PKI Design Guidelines (ctd)

In many situations no PKI of any kind is needed

Example: Authority-to-individual communications (e.g. tax filing)

- The authority knows who its users/clients are; everyone knows who the authority is
- Obvious solution: S/MIME or PGP
- Practical solution: SSL/TLS web server with access control
- Revocation = disable user access
  - Instantaneous
  - Consistently applied
  - Administered by the organisation involved, not some third party

## PKI Design Guidelines (ctd)

Example: AADS/X9.59

- Ties keys to existing accounts
- Handled via standard business mechanisms
- Revocation = remove the key/close the account
- (US) Business Records Exception allows standard business records to be treated as evidence (rather than hearsay) in court
  - Following standard legal precedent is easier than becoming the test case for PKI

## PKI Design Guidelines (ctd)

Example: Business transactions

- Ask Citibank about certificate validity

vs.

- Ask Citibank to authorise the transaction directly
  → Use an online authorisation
- Well-established mechanisms (and much legal precedent) for online authorisation
- Strong consumer protection via Reg.E and Reg.Z
  - Report loss within 2 days: No liability
  - Report loss within 2-60 days (time to get a bank statement): Liability of $50
- Enacted when ATM/credit cards were introduced to keep the banks honest

## PKI Design Guidelines (ctd)

There's nothing that says you have to use X.509 as anything more than a complex bit-bagging scheme

- Provides broad toolkit and crypto token support without tying you to X.509 peculiarities
- Given the complete absence of quality control in X.509 implementations, the only portions that you can rely on are public-key handling and some form of identifying value
  - See the X.509 certificate section for more details

If you have a cert management scheme that works, use it

- Be careful about holding your business processes hostage to your PKI (or lack thereof)