Public Key Distribution, Certification and Revocation

Public Key Infrastructure (Distribution) Problem:

❖ How to determine the correct public key of a given entity
  - Binding between IDENTITY and PUBLIC KEY
❖ Possible attacks
  ◆ Name spoofing: Eve associates Alice’s name with Eve’s public key
  ◆ Key spoofing: Eve associates Alice’s key with Eve’s name
  ◆ DoS: Eve associates Alice’s name with a nonsensical (bogus) key
❖ What happens in each case?
Public Key Distribution

❖ Diffie - Hellman (1976) proposed the “public file” concept
  - universally accessible
  - no unauthorized modification
  - not scalable

Public Key Distribution

❖ Popek - Kline (1979) proposed using “trusted third parties” (TTPs) for this purpose:
  - TTPs know public keys of the entities and distribute them on-demand basis
  - Must use an on-line protocol (a disadvantage)
Certificates

❖ Kohnfelder (BS Thesis, MIT, 1978) proposed “certificates” as a public-key distribution method
❖ Explicit binding between the public-key and its owner/name
❖ Issued (signed) by a Certification Authority (CA)
❖ Issuance done off-line

Authenticated Public-Key-based Key Exchange (Station-to-Station or STS Protocol)

Choose random $v$

\[ K_{ab} = (y_b^v)^m \mod p \]

\[ SIG_{alice} = \{y_a, y_b\}_{alice} \]

\[ CERT_{alice} \]

Choose random $w$, Compute

\[ CERT_{bob} \]
Certificates

❖ Procedure
- Bob registers at local CA
- Bob receives his certificate:

\[
(PK_B, ID_B, issuance\_time, expiration\_time, etc.,...)SK_{CA}
\]
- Bob sends certificate to Alice
- Alice verifies CA’s signature
  ✤ PK_{CA} hard-coded in software
- Alice uses PK_{B} for encryption and/or verifying signatures

Who issues certificates?

CA: Certification Authority
  e.g. GlobalSign, VeriSign, Thawte, etc.
  look into your browser...

❖ Trustworthy (at least to its users/clients)
❖ Usually, off-line operation
❖ Has a well-known long-term certificate
❖ May store client certificates
❖ Very secure: physically and electronically
How does it work?

❖ A public/private key-pair is generated by user
❖ User requests certificate via local application (e.g., web browser) or (better option!) by off-line means
  - Good idea to prove knowledge of private key as part of the certificate request. Why?
❖ Public key and “name” usually part of a PK certificate

CA

❖ CA checks that requesting user is who he claims to be (in the certificate request)

❖ CA’s own certificate is signed by a higher-level CA. Root CA’s certificate is self-signed and his identity/name is “well-known”

❖ CA is a critical part of the system and must operate in a secure and predictable way according to some policy
Who needs them?

❖ Alice’s certificate is checked by whoever wants to: 1) verify her signatures, and/or 2) encrypt data for her.

❖ A verifier must:
  – know the public key of the CA(s)
  – trust all CAs involved

❖ Certificate checking is: verification of the signature and validity

❖ Validity: expiration + revocation checking

Verifying a certificate (assuming common CA)

To be covered later
What are PK certificates good for?

- Secure channels in TLS / SSL for web servers
- Signed and/or encrypted email (PGP, S/MIME)
- Authentication (e.g., SSH with RSA)
- Code signing
- Encrypting files
- IPSec: encryption/authentication at the network layer

Components of a certification system

- Request and issue certificates (different categories) with verification of identity
- Storage of certificates
- Publishing/distribution of certificates (LDAP, HTTP)
- Pre-installation of root certificates in a trusted environment
- Support by OS platforms, applications and services
- Maintenance of database of issued certificates (no private keys!)
- Helpdesk (information, lost + compromised private keys)
- Advertising revoked certificates (and support for applications to perform revocation checking)
- Storage "guidelines" for private keys
CA Security

- Must minimize risk of CA private key being compromised
- Best to have an off-line CA
  - Requests may come in electronically but not processed in real time
- Tamper-resistant hardware for the CA helps, i.e., should be impossible to extract private key

Storage of private key

- User management of his private key (use/storage support, key loss or compromise)
- Some OS-s offer Protected Storage to safe-guard private keys (encrypted).
- Applications take advantage of this; browsers sometimes save private keys encrypted in configuration directory
- users who mix applications or platforms must manually import / export private keys via PFX files.
Key lengths

- Strong encryption has been adopted since the relaxation of US export laws
- E.g., 1024-bit RSA and DES are unsafe
- Root CA should have an (RSA) key length of $\geq 2048$ bits given importance and typical lifetime of 3-5 years
- A personal RSA certificate should have key length of $\geq 1536$ bits (2048 is better)

Some relevant standards

- The IETF reference site
  - http://ietf.org/html.charters/wg-dir.html#Security_Area
- Public-Key Infrastructure (X.509, PKIX)
  - RFC 2459 (X.509 v3 + v2 CRL)
- LDAP v2 for certificate and CRL storage
  - RFC 2587
- Guidelines & practices
  - RFC 2527
- S/MIME v3
  - RFC 2632 & 2633
- TLS 1.0 / SSL v3
  - RFC 2246
**CertificationTree / Hierarchy**

Logical tree of CA-s

```
root
  PK_root
CA3 [PK_{CA3}]SK_{root} -> CA1 [PK_{CA1}]SK_{root} -> CA2 [PK_{CA2}]SK_{CA1}
CA4 [PK_{CA4}]SK_{CA3}
```

**Hierarchical PKI Example**

CAs

UCI UCSB UCSD UCR

End users
Hierarchical PKI Example

Upper level CAs

UCI

UCOP

UCLA

CSOP

CSUN

CSULB

End users

gtsudik@uci.edu
Cross Certificate Based PKI Example

CAs

End users

Cross certificates
Hybrid PKI example

Note that no cross arrows down or up!

Certificate Paths

Derived from PKI
Verifier must know public key of the first CA
Other public keys are ‘discovered’ one by one
All CAs on the path must be (implicitly) trusted by the verifier
X.509 Standard

X.509v3 - current version

ITU standard

ISO 9495-2 is the equivalent ISO standard

Defines certificate format, not PKI

Identity and attribute certificates

Supports both hierarchical model and cross certificates

End users cannot be CAs

X.509 Service

• Assumes a distributed set of servers maintaining a database about certificates

• Used in S/MIME, IPsec, SSL/TLS, SET.

• RSA, DSA, MD5*, HASH* are most commonly used algorithms
Format fields (some):

- version
- serial number
- signature algorithm ID
- issuer name (X.500 Distinguished Name)
- validity period
- subject(user) name (X.500 Distinguished Name)
- subject public key information
- issuer unique identifier (version 2 and 3 only)
- subject unique identifier (version 2 and 3 only)
- extensions (version 3 only)
- signature on the above fields
A sample certificate

Certificate:
Data:
  Version: 3 (0x2)
  Serial Number: 28 (0x1c)
  Signature Algorithm: md5WithRSAEncryption
  Issuer: C=US, O=Globus, CN=Globus Certification Authority
  Validity
    Not After : Apr 22 19:21:50 2009 GMT
  Subject: C=US, O=Globus, O=University of Southern California, \
    ou=ISI, CN=bonair.isi.edu
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    RSA Public Key: (1024 bit)
      Modulus (1024 bit):
        00:bf:4c:9b:ae:51:e5:ad:ac:54:4f:12:52:3a:69:
        <snip>
        b4:e1:54:e7:87:57:b7:d0:61
      Exponent: 65537 (0x10001)
  Signature Algorithm: md5WithRSAEncryption
    <snip>

Certificates in Practice

X.509 certificate format is defined in Abstract Syntax Notation 1 (ASN.1)
ASN.1 structure is encoded using the Distinguished Encoding Rules (DER)
A DER-encoded binary string is typically base-64 encoded to get an ASCII representation
Certificates in Practice

-----BEGIN CERTIFICATE-----
MIIDTzCCAmgXwIBAgIBATANBgkqhkiG9w0BAQQQADB8cHMEaWhvYDVQQKEwFmc3Jv
v3d0BiBQJQ0UtVEVMXHbyb2plY3QxIzAABgNVBlMwTWF1UENsb30xMjAgMgMw
MhA4BhA6BjAMBgNVBAMTQ2FVeTtQWWOBGQ03WU2c5b0AOG0U32YHA4Mv
-----END CERTIFICATE-----

Certificate Revocation Scenario

What if:
Bob's CA goes berserk?
Bob forgets his private key?
Someone steals Bob's private key?
Bob loses his private key?
Bob willingly discloses his private key?
  Eve can decrypt/sign while Bob's certificate is still valid...
  Bob reports key loss to CA (or CA finds out somehow)
CA issues a Certificate Revocation List (CRL)
  Distributed in public announcements
  Published in public databases
When verifying Bob's signature or encrypting a message for Bob, Alice first checks if Bob's certificate is still valid!

IMPORTANT: what about signatures "Bob" generated before he realized his key is lost?
More generally: Certificate is a capability!

Certificate revocation needs to occur when:
- certificate holder key compromise/loss
- CA key compromise
- end of contract (e.g. certificates for employees)

Certificate Revocation Lists (CRLs) hold the list of certificates that are not yet naturally expired but revoked
Reissued periodically (even if no activity!)
More on revocation later...

Requirements for revocation

Timeliness
Must check most recent revocation status

Efficiency
- Computation
- Bandwidth and storage
- Availability

Security
Types of Revocation

• Implicit
  Each certificate is periodically re-issued
  Alice has a fresh certificate → Alice not revoked
  No need to distribute/publish revocation info

• Explicit
  Only revoked certificates are periodically announced
  Alice’s certificate not listed among the revoked →
  Alice not revoked
  Need to distribute/publish revocation info

Revocation methods

• CRL - Certificate Revocation List
  • CRL-DP, indirect CRL, dynamic CRL-DP,
  • delta-CRL, windowed CRL, etc.
  • CRT and other Authenticated Data Structures

• OCSP - On-line Certificate Status Protocol

• CRS - Certificate Revocation System
CRL

• Off-line mechanism

• CRL = list of revoked certificates (e.g., SNs) signed by a revocation authority (RA)

• RA not always CA that issued the revoked PKC

• Periodically issued: daily, weekly, monthly, etc.

Pros & Cons of CRLs

Pros

Simple
Don’t need secure channels for CRL distribution

Cons

Timeliness: “window of vulnerability”
CRLs can be huge
How to distribute CRLs reliably?
On January 29 and 30, 2001, VeriSign, Inc. issued two certificates for Authenticode Signing to an individual fraudulently claiming to be an employee of Microsoft Corporation.

Any code signed by these certificates appears to be legitimately signed by Microsoft.

Users who try to run code signed with these certificates will generally be presented with a warning dialog, but who wouldn’t trust a valid certificate issued by VeriSign, and claimed to be for Microsoft?

Certificates were very soon placed in a CRL, but:

- code that checks signatures for ActiveX controls, Office Macros, and so on, didn’t do any CRL processing.

According to Microsoft:

- since the certificates don’t include a CRL Distribution Point (DP), it’s impossible to find and use the CRL!
Certificate Revocation Tree (CRT)

- proposed by P. Kocher (1998)
- based on Merkle’s Hash Trees (MHTs)
  - hash trees first proposed by R. Merkle in another context in 1979 (one-time signatures)
  - improvement to Lamport-Diffie OTS scheme
- based on the following idea:
  - A wants to sign a bit of information. A gives B the image (y) produced as y=F(x)
  - Eventually A reveals the pre-image: x
  - B checks that: y=F(x)

CRT contd.

- express ranges of SN of PKC’s as tree leaf labels:
  - E.g., (5 -- 12) means: 5 and 12 are revoked, the others larger than 5 and smaller than 12 are okay
  - Place the hash of the range in the leaf
- response includes the corresponding tree leaf, the necessary hash values along the path to the root, the signed root
- the CA periodically updates the structure and distributes to un-trusted servers called Confirmation Issuers
Characteristics of CRT

- each response represents a proof

- length of proof is: $O(\log n)$
  - Much shorter than CRL which is $O(n)$
  - Where $n$ is # of revoked certificates

- only one “real” signature for tree root (can be done off-line)
Explicit Revocation: OCSP

- OCSP = On-line Certificate Status Protocol (RFC 2560) - June 1999
- In place of or, as a supplement to, checking CRLs
- Obtain instantaneous status of a PKC
- OCSP may be used in sensitive, volatile settings, e.g., stock trades, electronic funds transfer, military

OCSP players

1. Cert request
2. Bob
3. Transaction + request
4. OCSP request
5. OCSP response / Error message
6. Transaction response
OCSP definitive response

All definitive responses have to be signed by:

- issuing CA
- Trusted Responder (OCSP client trusts the TR’s PKC)
- CA Authorized Responder which has a special PKC (issued by the CA) saying that it can issue OCSP responses on CA’s behalf

Responses for each certificate

Response format:
- target PKC SN
- PKC status:
  - good - positive answer
  - revoked - permanently/temporarily (on-hold)
  - unknown - responder doesn’t know about the certificate being requested
- response validity interval
- optional extensions
Special Timing Fields

A response contain three timestamps:

- **thisUpdate** - time at which the status being indicated is known to be correct
- **nextUpdate** - time at or before which newer information will be available
- **producedAt** - time at which the OCSP responder signed this response. Useful for response pre-production

Security Considerations

- **On-line method**
- **DoS vulnerability**
  - flood of queries + generating signatures!
  - unsigned responses → false responses
  - pre-computing responses offers some protection against DoS, but...
- pre-computing responses allows replay attacks (since no nonce included)
  - but OCSP signing key can be kept off-line
Open questions

- Consistency between CRL and OCSP responses
  - possible to have a certificate with two different statuses.
- If OCSP is more timely and provides the same information as CRLs, do we still need CRLs?
- Which method should come first: OCSP or to CRL?

Implicit Revocation:
Certificate Revocation System (CRS)

- Proposed by Micali (1996)
- Improves CRL communication costs / size
- Basic idea: signing a message for every certificate stating its status
- Uses off-line/on-line signatures to reduce update costs
**CRS: creation of a certificate**

Two new parameters in PKC: $Y_{MAX}$ and $N$

\[ Y_{MAX} = H^{MAX}(Y_0) \]
\[ N = H(N_0) \]

\[ [Y_0, N_0] \] -- per-PKC secrets stored by CA

$H()$ -- public one-way function

**How CRS works**

- daily update $U_k$ for each certificate $C_k$
  - if still valid, $U_k = Y_{365-i} = H^{365-i}(Y_0)$
  - if revoked, $U_k = N_0$

NOTE: $i=0$ at the issuance date