Public-key Infrastructure

- Adams and Lloyd ("Understanding PKI, Second Edition", Addison Wesley, 2003) define a public-key infrastructure as follows:

  > *A PKI is the basis of a pervasive secruity infrastructure whose services are implemented and delivered using public-key concepts and techniques*

- a PKI has many components, such as the following:
  - certificate issuance
  - certificate revocation
  - key backup / recovery / update
  - cross-certification
  - non-repudiation
  - time-stamping

PKI-enabled Services

- secure communication:
  - secure email: S/MIME, PGP
  - secure web service access: SSL/TLS
  - secure VPN: IPsec/IKE
- access control (privilege management):
  - authentication
  - authorization
  - delegation
- privacy architecture
  - anonymous / pseudonymous certificates

Certificates

- the concept of a certificate was introduced in 1978 by Kohnfelder, in a *bachelor's thesis*
- in its simplest form, a certificate binds an *identity* to a *public key*
- this is done by having a trusted *certification authority* (*CA*) sign the information on a certificate
- the CA's signature can be verified using a public verification algorithm, which allows the information on the certificate to be authenticated
- certificates allow *secure* and *scalable* PKIs to be built

X.509 v3 Certificates

X.509 v3 certificates contain the following fields:

- version number
- serial number
- signature algorithm ID
- issuer name
- validity period
- subject name (certificate owner)
- owner's public key
- optional fields
- the CA's signature on all the previous fields

X.500 Names

- X.509 certificates use X.500 names
- X.500 names have a hierarchical format, e.g. C=US, O=Microsoft, OU=Management, CN= Bill Gates
- "C" denotes country, "O" denotes organization, "OU" denotes organizational unit and "CN" denotes common name
- the information is encoded numerically, using *object identifiers*
- X.500 names are not compatible with DNS names (e.g., `dstinson@cacr.math.uwaterloo.ca`)
- the DNS maps names for computers (and users) to IP addresses
- X.500 is intended to be a true directory, allowing X.509 certificates to be looked up and accessed
- however, no widely deployed X.500 directory exists at the present time

Other Certificate Formats

- SPKI (simple public key infrastructure) certificates emphasize authorizations rather than identities; incompatible with X.509 certificates
- PGP (pretty good privacy) is a user-based email system; incompatible with X.509 certificates
- SET (secure electronic transaction) specifications use enhanced X.509 certificates

Certificate Life-Cycle Management

- registration
- key generation and distribution
- key backup
- certificate issuance
- certificate retrieval / validation
- key update / recovery
- certificate revocation / expiration
- key history / archive

Certificate Validation

In its simplest form, certificate validation incorporates the following operations:

- verify the integrity of the certificate by verifying the signature of a trusted CA on the certificate (note that the verification key of the CA either is known *a priori*, implicitly trusted, or verified by *out-of-band* information)
- verify that the certificate has not expired
- verify that the certificate has not been revoked
- verify that the usage of the certificate conforms to any policy constraints specified in optional fields on the certificate

Trust Models

- often, a certificate will not be signed directly by a trusted CA
- instead, it is necessary to follow a *certificate path* from a trusted CA to a given certificate
- the *trust model* specifies rules which determine how these certificate paths should be constructed
- here are some examples of trust models:
  - *strict hierarchy*
  - *distributed trust architecture*
  - *web browser model*
  - *user-centric model* (web of trust)
- *cross-certification* is when two CAs sign each other's certificates

Strict Hierarchy Model

- the *root CA* has a self-signed, self-issued certificate
- the root *CA* is a *trust anchor*
- (optional) the root *CA* issues certificates for lower-level CA's
- any *CA* can issue certificates for *end users*
- an end user, say *Alice*, in order to verify the certificate of another end user, say *Bob*, needs a *certificate path* from the root *CA* to *Bob*
- in a strict hierarchy, *Bob* could send all the certificates in this path to *Alice*
- it is assumed that *Alice* knows the verification key of CA_{root}, say $ver_{CA_{root}}$

Certificate Path Validation

- suppose, for example, that CA_{root} has signed the certificate of CA_1, and CA_1 has signed Bob's certificate
- it is assumed that $Alice$ knows the verification key of CA_{root}, say $ver_{CA_{root}}$
- the certificate path is validated as follows:
  1. $Alice$ validates $\mathsf{Cert}(CA_{root})$ using the key $ver_{CA_{root}}$
  2. $Alice$ validates $\mathsf{Cert}(CA_1)$ using the key $ver_{CA_{root}}$
  3. $Alice$ extracts the key ver_{CA_1} from $\mathsf{Cert}(CA_1)$
  4. $Alice$ validates $\mathsf{Cert}(Bob)$ using the key ver_{CA_1}
  5. $Alice$ extracts Bob's public key(s) from $\mathsf{Cert}(Bob)$
- $Alice$ must also ensure that the path adheres to the trust model

Distributed Trust Models

- the strict hierarchy may work well within a single organization
- it may be desirable to "connect" root CAs of two or more different PKI domains, a process sometimes called *PKI networking*
- this creates a "super-PKI" made up of users in different domains
- in the *mesh configuration*, all the root CAs will cross-certify each other
- n root CAs will lead to $n(n-1)$ cross-certifications
- an alternative approach is the *hub-and-spoke configuration*, in which the n root CAs each cross-certify with a new *hub CA*
- this reduces the number of cross-certifications to $2n$