• Public key infrastructure

• Certificates

• Trust models
  – hierarchical model
  – networked PKIs model
  – web browser multiple CAs model
  – user-centric web of trust of PGP
Possibly the biggest challenge in public-key cryptography is ensuring the authenticity of public keys

- Alice wants to encrypt a message for Bob using his public key
- but Alice doesn’t know Bob personally

We have already seen that public keys can be managed through the use of certificates

- there is a trusted certification authority with a known public key
- the CA issues certificates by signing a user’s identity along with her public key
- Bob’s public key \( pk_B \) is authentic if it matches his key in
\[
\text{cert}_B = \text{sig}_{CA}(B, pk_B)
\]
A public-key infrastructure (PKI) is a system for managing trust in the public keys through the use of certificates

- it is the basis of a pervasive security infrastructure whose services are implemented and delivered using public-key techniques

Ideally, a PKI should function without active participation of the user

- when we say that a network user Alice performs various operations, it implies that her software does this
  
  - Alice might not be even aware of the PKI-related procedures

A PKI’s goal is to eliminate the need for users to share precomputed symmetric keys and enable the use of public-key cryptography
Public-Key Infrastructure

- There are many components to a PKI
  - certificate issuance
    - before a certificate can be issued, the identity and credentials of the user must be verified using non-cryptographic means
    - a secure procedure must be used to generate the public and private keys for the certificate’s owner
    - the certificate must be securely transmitted to its owner
  - certificate revocation
    - this is done before a certificate’s expiration date under unforeseen circumstances
    - for example, if a private key is lost or stolen
    - additional infrastructure is need to recognize revoked certificates
• **PKI components** (cont.)
  
  – *key backup/recovery/update*
    
    • *key backup* refers to secure storage of users’ private keys by the administrator of the PKI
    
    • *key recovery* is a protocol that allows a lost or forgotten key to be restored or re-activated
    
    • *key update* occurs when a key is to be changed (e.g., a certificate is about to expire)

  – *timestamping*
    
    • the times at which a key is issued, revoked, or updated may be important
    
    • such timestamps are often included in the certificate
• Once a PKI is built and operational, it allows various applications to be built on top of it
  – such applications can be called **PKI-enabled services**

• Examples of **PKI-enabled services** include
  – **secure communication**
    • secure email protocols include Secure Multipurpose Internet Mail Extensions (S/MIME) and Pretty Good Privacy (PGP)
    • secure web service access is provided through Secure Sockets Layer (SSL) or Transport Layer Security (TLS)
    • secure virtual private networks (VPNs) use the Internet Protocol Security (IPsec) protocols
• Examples of **PKI-enabled services** (cont.)
  
  – **access control**
    
    • access control provides the means of managing user privileges through authentication, authorization, and delegation
    
    • access control normally requires user authentication via a password or cryptographic identification scheme
  
  – **privacy architecture**
    
    • a privacy architecture permits the use of anonymous or pseudonymous credentials
    
    • such credentials (or certificates) allow an individual to show membership or another property without specifying their identity
Certificates

- **Certificates** are important building blocks of PKIs
- It is generally assumed that there is a trusted CA
  - each user has access to an authentic copy of the CA’s public key
- In its simplest form, a certificate is a CA’s signature on an identity and the identity’s public key
- A valid CA’s signature of this form is treated as a confirmation that the public key belongs to that identity
  - i.e., it is assumed that the CA verifies the identity before signing any given key
- **X.509 v3** is a popular type of certificate
X.509 certificates contain the following fields:

- data
  - version
  - serial number
  - signature algorithm
  - issuer name
  - validity period
  - subject name
  - subject public key and public key algorithm
  - optional fields
- signature on all of the above fields
• X.509 certificates were originally defined using X.500 names for subjects

• **X.500 names** have a hierarchical format:

  - C = US
  - O = University at Buffalo
  - OU = Department of Computer Science and Engineering
  - CN = Chunming Qiao

  – here C denotes country, O denotes organization, OU denotes organizational unit, and CN denotes common name

• This format ensures that **everyone has a unique global name**

  – it could be used as a global phone directory

• No global X.500 directory exists today
• Example X.509 certificate

Data:

- Version: 3
- Certificate Signature Algorithm: PKCS #1 SHA-256 With RSA Encryption
- Issuer: CN = Google Internet Authority G3
  - O = Google Trust Services
  - C = US
- Validity:
  - Not Before: September 25, 2018, 7:43:00 AM GMT
  - Not After: December 18, 2018, 7:43:00 AM GMT
- Subject: C = US, ST = California, L = Mountain View,
  - O = Google LLC,
  - CN = www.google.com
• **Example X.509 certificate** (cont.)

  Subject Public Key Info:
  
  Public Key Algorithm:
  
  Algorithm Identifier: Elliptic Curve Public Key
  
  Algorithm Parameters: ANSI X9.62 elliptic curve prime256v1
  (aka secp256r1, NIST P-256)

  Subject’s Public Key:
  
  Key size: 256 bits
  
  Base point order length: 256 bits
  
  Public value: 04 5a cd 02 99 d4 9a a0 ab 0f 9f c0 9e d1 c5 2d
  
  Certificate Signature Value:
  
  Size: 256 Bytes / 2048 Bits
  
  88 14 32 8e 8f 32 95 8f d4 a4 85 0d 2f 55 23 78
  
  4b 58 6d cf b4 5e 1c 2b a8 3f 1c 6c 83 7e 6a fa
  
  …
Certificates

- **Certificate life-cycle management** has several phases, which include:
  - registration
  - key generation (and backup), key distribution
  - certificate issuance
  - certificate retrieval and validation
  - certificate expiration

- Additionally, we might have:
  - key update
  - key recovery
  - certificate revocation
During the validation, the following steps take place

- verify the integrity and authenticity of the certificate by verifying the CA’s signature
  - it is assumed that the CA’s keys is known a priori or can be reliably verified using external resources
- verify that the certificate has not expired
- verify that the certificate has not been revoked
- if applicable, verify that the certificate’s usage complies with the policy constraints specified in the certificate’s optional fields
Certificates

- A PKI needs a mechanism to verify that a certificate has not been revoked prior to its expiration date.

- A **certificate revocation list (CRL)** is the most common technique:
  - A CRL is a list of the serial numbers of certificates that are revoked but not expired.
  - This list is prepared by the CA and is signed.
  - It is updated periodically and made available at a public directory.

- For efficiency reasons, **delta CRLs** can be used:
  - Instead of containing all revoked certificates, delta CRLs contain the changes since the most recent previously issued CRL or delta CRL.

- Alternatively, an **online certificate status protocol** can be used to query a certificate’s status in real time.
Trust Models

- Often there is more than one CA that can sign certificates
  - a certificate can be verified by following a certificate path from a trusted CA to a given certificate
  - each certificate in the path is signed by the owner of the previous certificate in the path
  - if all certificates in the path can be verified, the last certificate is considered to be valid

- A trust model specifies the way in which a certificate path should be constructed, e.g.,
  - for example, strict hierarchy; networked PKIs; web browser model; and user-centric model (or web of trust)
• **Strict hierarchy model**
  
  – there is a single root CA that has a self-signed self-issued certificate
  
  • the root CA is called **trust anchor**
  
  – the root CA may issue certificates for other CAs
  
  – any CA can issue certificates for end users
• Strict hierarchy model
  – what is the meaning of a directed edge?
  – usage example: Alice would like to verify a certificate of user $U_3$
    • $U_3$ sends to Alice
    • Alice performs certificate path validation by
Trust Models

- **Networked PKIs model**
  - sometimes it might be desirable to connect root CAs of two or more different PKI domains
    - this can be called **PKI networking** and it creates one large PKI with users in different domains
  - cases when one CA signs the certificate of another CA are called **cross-certification**
  - in **mesh configuration**, there are several root CAs and all of them cross-certify one another
    - the number of cross-certifications for \( n \) root CAs is \( \binom{n}{2} \)
  - in **hub-and-spoke configuration**, root CAs each cross-certify independently with a new hub CA
    - the number of cross-certifications for \( n \) root CAs is \( n \times (n-1) \)
• Networked PKIs model (cont.)
  – example mesh configuration
  
  \[ \begin{array}{c}
  \text{CA}_{\text{root}1} \\
  \text{CA}_{\text{root}2} \\
  \text{CA}_{\text{root}3} \\
  \text{CA}_{\text{root}4}
  \end{array} \]

  – example hub-and-spoke configuration

  \[ \begin{array}{c}
  \text{CA}_{\text{hub}} \\
  \text{CA}_{\text{root}1} \\
  \text{CA}_{\text{root}2} \\
  \text{CA}_{\text{root}3}
  \end{array} \]
• Networked PKIs model (cont.)
  – to validate Bob’s certificate, Alice must be able to find a path of certificates from her trust anchor $CA_{root_i}$ to Bob
    • this process is called path discovery
  – assume that Bob’s certificate is signed by $CA_{root_j}$
  – what path does Alice construct in mesh configuration?
  – in hub-and-spoke configuration?
Trust Models

- Web browser model
  - most web browsers come preconfigured with a set of independent root CAs
  - all of them are considered to be trusted by a user of the browser
  - security considerations for this model:
    - users normally don’t have information about the security of these pre-configured root CAs
    - the list is editable, but many users are not even aware of it
    - there might be no mechanism to revoke a root CA from a web browser
    - there might be no automated way to update root CAs’ certificates
    - if permitted, most users choose to proceed with expired certificates
Trust Models

- **Pretty Good Privacy (PGP)**
  - PGP is used for email, where each user is her own CA
  - a PGP certificate contains an email address (UID), a public key (PK), and one or more signatures on this (UID, PK) pair
  - when a Alice creates her key, she first self-signs it:
    \[
    \text{cert}(\text{Alice}) = (\text{data}, \text{sig}_A(\text{data})), \text{ where }
    \text{data} = (\text{UID} = \text{alice@buffalo.edu}, \text{PK} = 0x\text{BEEF1234})
    \]
  - later, other users may add their signatures to Alice’s key, so she has
    \[
    \text{cert}(\text{Alice}) = (\text{data}, \text{sig}_A(\text{data}), \text{sig}_B(\text{data}), \text{sig}_C(\text{data}), \ldots)
    \]
• **Pretty Good Privacy (PGP)** (cont.)
  – when Bob wants to sign Alice’s key, he needs to
    • retrieve a copy of Alice’s key (from Alice or a key server)
    • verify Alice’s identity
    • ensure that the key he has for Alice is the same as what Alice has
      – this is done by comparing the fingerprints of Alice’s key and Bob’s version of Alice’s key
    • sign Alice’s key
    • send the updated certificate to Alice or a key server
      – if Bob performed all checks, he is likely to trust the key
      – he can encrypt emails to Alice or receive signed emails from her
Trust Models

- **Pretty Good Privacy (PGP) (cont.)**
  - Alice keeps a collection of certificates she obtained from different sources in a data structure called a **keyring**
  - she is able to declare how much she trusts the owner of a certificate
    - often the choices are implicitly trusted, completely trusted, partially trusted, or untrusted
  - Alice’s own key is implicitly trusted
  - if Alice is convinced that Bob’s public key is valid and Bob would not sign invalid keys, she declares Bob’s key completely trusted
  - she can also set how the trust of unknown keys is computed
    - e.g., she can set that keys with the distance larger than 3 from her should not be trusted
Trust Models

- PGP web of trust
  - each user can set the trust of a key at signing time and also choose how trust of unsigned keys is computed
  - best connected keys http://pgp.cs.uu.nl/doc/top_50.html

<table>
<thead>
<tr>
<th>#</th>
<th>dist</th>
<th>key</th>
<th>uid</th>
<th># sigs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.48573</td>
<td>94C09C7F</td>
<td>Peter Palfrader</td>
<td>678 sigs</td>
</tr>
<tr>
<td>2</td>
<td>3.57826</td>
<td>C82E0039</td>
<td>Peter Palfrader</td>
<td>441 sigs</td>
</tr>
<tr>
<td>3</td>
<td>3.58812</td>
<td>68FD549F</td>
<td>Martin Michlmayr</td>
<td>569 sigs</td>
</tr>
<tr>
<td>4</td>
<td>3.59351</td>
<td>248AEB73</td>
<td>Rene Engelhard</td>
<td>560 sigs</td>
</tr>
<tr>
<td>5</td>
<td>3.59754</td>
<td>E263FCD4</td>
<td>Kurt Gramlich</td>
<td>559 sigs</td>
</tr>
<tr>
<td>6</td>
<td>3.60146</td>
<td>F081195D</td>
<td>Matthias Bauer</td>
<td>392 sigs</td>
</tr>
<tr>
<td>7</td>
<td>3.60405</td>
<td>75BE8097</td>
<td>Florian Lohoff</td>
<td>379 sigs</td>
</tr>
<tr>
<td>8</td>
<td>3.61369</td>
<td>3F3E6426</td>
<td>Guido Guenther</td>
<td>411 sigs</td>
</tr>
</tbody>
</table>

- key servers: keyserver.pgp.com, pgp.mit.edu, etc.
There are several difficulties that prevent a large-scale deployment of PKIs:

- the first problem is who should be responsible for development, maintenance, and regulation of PKIs,
- the second problem is what standards should be used in PKIs,
- the third problem is that different PKIs are needed in different environments,
- also, a lack of PKI-compatible applications is slowing the deployment of PKIs.

• this is a chicken-and-egg problem
Are There Alternatives?

- **Emerging technologies** might provide alternative solutions to having PKI
  - one example is **contract signing using blockchain technology**
    - blockchain offers immutable storage organized by time
    - contract signing could be realized using digital signatures
      - this requires having an authentic copy of the signer’s key as otherwise the security guarantees don’t hold
    - by reliably storing one’s consent to executing the contract on the blockchain, we could have similar legal protection
      - laws and regulations often lag behind technological changes
Summary

- **Public key infrastructure** target at solving the problem of public-key management between users who don’t know each other
  - this is often addressed through the use of certificates
  - many different architectures for building trust exist
  - different applications use different models
    - multiple CA certificates on the web
    - web of trust in PGP
- No large-scale public PKI is currently in place