Certificates, part 2

Prof. Clarkson
Spring 2017
Digital certificate is a signature binding together:

- **identity** of principal
- **public key** of that principal (might be encryption or verification key)
- (maybe more)

**Notation:** Cert$(S; I)$ is a certificate issued by principal $I$ for principal $S$

- let $b = id_S, K_S, ...$
- $Cert(S; I) = b, Sign(b; k_I)$
- **Issuer** $I$ is certifying that $K_S$ belongs to **subject** $id_S$
Review: PKI

- System for managing distribution of certificates
- Two main philosophies:
  - Decentralized: anarchy, no leaders
  - Centralized: oligarchy, leadership a few elite
PKI Example 2: CAs

• Uses a centralized PKI philosophy (at least as evolved in marketplace)
• Invented (?) by Digital [Gasser et al. 1989], used in early Netscape browsers
• Certificate authority (CA): principal whose purpose is to issue certificates
Using a CA

• Everyone enrolls with the CA to get a certificate
  – E.g., Alice enrolls and gets Cert(Alice; CA)
• Your system comes pre-installed with CA's self-signed certificate Cert(CA; CA)
• When you receive a message signed by Alice:
  – you contact CA to get Cert(Alice; CA)
  – or Alice just includes that certificate with her message
CAs and web browsers

• Web server has certificate Cert(server; CA) installed
  – Server’s identity is its URL
  – CA is a root for which Cert(CA; CA) is installed in browser

• Browser authenticates web server
  – Using server’s URL and public key from certificate
  – Perhaps based on protocol from last lecture
  – Perhaps based on SSL (this lecture)

• Machines are authenticating machines
Many CAs

• There can't be only one
  – No single CA is going to be trusted by all the world's governments, militaries, businesses
  – Though within an organization such trust might be possible

• So there are many
  – Around 1500 observed on public internet
  – Your OS and/or browser comes with some pre-installed

• Organizations act as their own CA, e.g.:
  – Company issues certificates to employees for VPN
  – Bank issues certificates to customers
  – Central bank issues certificates to other banks
  – Manufacturer issues certificates to sensing devices
**Demo: OS X Keychain Access**

![Keychain Access Interface](image)

**Keychain Access**

Click to unlock the System Roots keychain.

### Keychains

**login**
- iCloud
- System

**System Roots**

### AAA Certificate Services

- **Root certificate authority**
- Expires: Sunday, December 31, 2028 at 8:59:59 PM Eastern Standard Time
- This certificate is valid

<table>
<thead>
<tr>
<th>Name</th>
<th>Kind</th>
<th>Expires</th>
<th>Keychain</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA Certificate Services</td>
<td>certificate</td>
<td>Dec 31, 2029, 0:00:00 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Actalis Authentication Root CA</td>
<td>certificate</td>
<td>Sep 22, 2030, 7:22:02 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AddTrust Class 1 CA Root</td>
<td>certificate</td>
<td>May 30, 2020, 6:38:31 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AddTrust External CA Root</td>
<td>certificate</td>
<td>May 30, 2020, 6:48:38 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AddTrust Public CA Root</td>
<td>certificate</td>
<td>May 30, 2020, 6:41:50 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AddTrust Qualified CA Root</td>
<td>certificate</td>
<td>May 30, 2020, 6:44:50 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Admin-Cert-CA</td>
<td>certificate</td>
<td>Nov 10, 2021, 2:51:07 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AffirmTrust Commercial</td>
<td>certificate</td>
<td>Dec 31, 2030, 0:08:08 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AffirmTrust Networking</td>
<td>certificate</td>
<td>Dec 31, 2030, 9:08:24 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AffirmTrust Premium</td>
<td>certificate</td>
<td>Dec 31, 2040, 8:10:38 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AffirmTrust Premium ECC</td>
<td>certificate</td>
<td>Dec 31, 2040, 9:20:24 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AnF Global Root CA</td>
<td>certificate</td>
<td>Jun 5, 2033, 1:45:38 PM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Apple Root CA</td>
<td>certificate</td>
<td>Feb 9, 2035, 4:40:36 PM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Apple Root CA - G2</td>
<td>certificate</td>
<td>Apr 30, 2039, 2:10:09 PM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Apple Root CA - G3</td>
<td>certificate</td>
<td>Apr 30, 2039, 2:19:06 PM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Apple Root Certificate Authority</td>
<td>certificate</td>
<td>Feb 9, 2025, 7:18:14 PM</td>
<td>System Roots</td>
</tr>
<tr>
<td>ApplicationCA</td>
<td>certificate</td>
<td>Dec 12, 2017, 10:00:00 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>ApplicationCA2 Root</td>
<td>certificate</td>
<td>Mar 12, 2033, 10:00:00 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Autoridad de...ra CIF A6Z63496B</td>
<td>certificate</td>
<td>Jan 30, 2030, 3:38:15 AM</td>
<td>System Roots</td>
</tr>
</tbody>
</table>

177 items
Enrollment with a CA

• You create a key pair: you do this so that CA doesn't learn your private key
• You generate a certificate signing request (CSR); it contains the identity you are claiming
• You send the CSR to a CA, perhaps along with payment
• The CA verifies your identity (maybe)
• The CA signs your key, thus creating a certificate, and sends certificate to you
Enrollment with a CA

- You create a key pair: you do this so that CA doesn't learn your private key
- You generate a certificate signing request (CSR); it contains the identity you are claiming
- You send the CSR to a CA, perhaps along with payment
- The CA verifies your identity (maybe)
- The CA signs your key, thus creating a certificate, and sends certificate to you
Identity verification

• **Extended validation (EV) certificate:**
  – CA does extra checking of your identity
  – Certificate marked as having received EV
  – Web browser reflects EV mark in UI

• **Examples of extra checking:**
  – Verify legal existence of organization including some sort of registration number; record legal business number as part of subject’s identity in certificate
  – Verify physical operation of organization by a site visit
  – Verify phone number as listed by a public phone company

• CA record all those data in the certificate as part of subject's identity

• Example: [https://www.paypal.com](https://www.paypal.com)
Issuing certificates

Conflicting goals:

• CA private signing key must be kept secret
  — the public verification key is pre-installed on user systems; hard to update
  — if ever leaked, signing key could be used to forge certificates
  — easy way to realize goal: keep it in cold storage

• CA private signing key must be available for use
  — to sign new certificates when users request them
  — easy way to realize goal: keep it in computer's memory
Issuing certificates

Solution: use root and intermediate CAs

• **root CA:** the certificate at root of trust in a chain; pre-installed; key kept in highly secure storage

• **intermediate CA(s):** certified by root CA, themselves certify user keys; might be run by a different organization than root

• example: [https://www.facebook.com](https://www.facebook.com)
## Authentication

<table>
<thead>
<tr>
<th></th>
<th>Humans</th>
<th>Machines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Humans authenticating...</td>
<td>Faces, tickets, passwords</td>
<td>Secure attention key, visual secrets</td>
</tr>
<tr>
<td>Machines authenticating...</td>
<td>Passwords, biometrics</td>
<td>Tokens, CAs as used in web</td>
</tr>
</tbody>
</table>
Success!

We've solved the phonebook problem!

To publish public key, user can:
• distribute it as part of web of trust
• or engage CA to provide certificate

...or, have we???
PROBLEMS WITH PKI
Problem 1: Revocation

- Keys (subject's, issuer's) get compromised
- Or subject leaves an organization
  ...certificates therefore need to be revoked
- There's no perfect solution
  - Fast expiration
  - Certificate revocation lists (CRLs)
  - Online certificate validation
Revocation

Fast expiration

• **Idea:**
  – Validity internal is short, e.g. 10 min to 24 hr
  – A kind of revocation thus happens automatically
  – Any compromise is bounded

• **Problem:**
  – CAs have to issues new certificates frequently, including checking identities
  – Machines have to update certificates frequently
Revocation

Certificate revocation lists (CRLs)

• Idea:
  – CA posts list of revoked certificates
  – Clients download and check every time they need to validate certificate

• Problems:
  – Clients don't (because usability)
  – Or they cache, leading to TOCTOU attack
  – CRL must always be available (so an attractive DoS target)

• Chromium does this, with a CRL limited to 250kb
Revocation

Online certificate validation

• **Idea:**
  – CA runs *validation server*
  – Clients contact it each time to validate certificate

• **Problems:**
  – Clients don't
  – Server must always be available (so an attractive DoS target)
  – Reveals to CA which websites you want to access
Revocation

Online certificate validation

• **Follow-on solution:** stapling
  – Certificates must be accompanied by fresh assertion from CA that certificate is still valid
  – Whoever presents certificate to client is responsible for acquiring assertion

• Firefox [does this](#) but doesn't *hard fail* because "[validation servers] aren't yet reliable enough"
  – Unless web site has previously served up a certificate to browser with Must Staple extension set
Problem 2: Authority

- CAs go rogue, get hacked, issue certificates that they should never have issued
  - e.g., Dutch CA DigiNotar (2011), which was included in many root sets: 500 bogus certificates issued, including for Google, Yahoo, Tor

- Missing a means for authorization of who may issue certificates for which principals
There’s no perfect solution

- **Key pinning:** upon first connection to a server, client learns a set of public keys for server; in future connections, certificate must contain one of those keys

- **Certificate transparency:** maintain a public log of issued certificates; require any presented certificate to be in that log; monitor log to notice misbehavior

- **Certificate Authority Authorization (CAA):** piggyback on DNS system; DNS record for entity specifies allowed CAs; a good CA won’t issue cert unless they are authorized

- **DNS-based Authentication of Named Entities (DANE):** piggyback like CAA; client checks whether cert comes from authorized CA
USING CAs IN SSL
Secure Sockets Layer (SSL)

• aka Transport Layer Security (TLS)

• SSL 3.1 = TLS 1.0 (1999)
  – Broken by attack in 2011 based on improper choice of IVs for CBC mode

• SSL 3.2 = TLS 1.1 (2006)
  – Fixes IVs

• SSL 3.3 = TLS 1.2 (2008)
  • Upgrades crypto primitives (AES, SHA-256, etc.)
# Network stack

<table>
<thead>
<tr>
<th>Layer</th>
<th>e.g.</th>
<th>Connects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>HTTP</td>
<td>processes</td>
</tr>
<tr>
<td>Transport</td>
<td>TCP</td>
<td>hosts</td>
</tr>
<tr>
<td>Internet</td>
<td>IP</td>
<td>networks</td>
</tr>
<tr>
<td>Link</td>
<td>WiFi</td>
<td>devices</td>
</tr>
</tbody>
</table>
# Network stack

<table>
<thead>
<tr>
<th>Layer</th>
<th>e.g.</th>
<th>Connects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>HTTP</td>
<td>processes</td>
</tr>
<tr>
<td></td>
<td>SSL</td>
<td></td>
</tr>
<tr>
<td>Transport</td>
<td>TCP</td>
<td>hosts</td>
</tr>
<tr>
<td>Internet</td>
<td>IP</td>
<td>networks</td>
</tr>
<tr>
<td>Link</td>
<td>WiFi</td>
<td>devices</td>
</tr>
</tbody>
</table>

- SSL provides secure channel atop underlying guarantees of transport layer
- HTTPS = HTTP + SSL
SSL terminology

• **Record:** message sent during session

• **Session:**
  – communication channel
  – between *client* and *server*
  – logical
  – bi-directional (and direction matters)
  – *optionally* secured for confidentiality and/or integrity against Dolev-Yao attacker
SSL protocols

- **Handshake protocol:** initial channel setup
- **Record protocol:** exchange of messages

Caveats:

- *what follows is common way of configuring those protocols, not the only way*
- *no official rationale for the protocol*
Record protocol

Connection state:
• **cmk**: client HMAC key
• **smk**: server HMAC key
• **cek**: client symmetric encryption key
• **sek**: server symmetric encryption key
• **civ**: client IV
• **siv**: server IV
• **cseq**: client sequence number
• **sseq**: server sequence number
Record protocol

Directional communication:

- both client and server are meant to know the entire state, but...
- from client to server uses cXX state
- from server to client uses sXX state

... defends against reflection attacks
Record protocol

For client to send record to server:

1. C: \[ t = \text{MAC}(r, \text{cseq}; \text{cmk}); \]
   \[ c = \text{Enc}(r, t; \text{civ}; \text{cek}); \]
   \[ \text{cseq}++; \quad \text{// if overflow, re-key} \]
   \[ \text{civ} = \text{rand}() \]

2. C -> S: c

Server to client is the same with sXX part of connection state
Handshake protocol

- **Purpose:**
  - Establish **ciphersuite**
  - Then establish connection state

- **Ciphersuite:** triple of cryptographic choices...
  1. Protocol for key establishment
  2. Block cipher and mode
  3. PRF (typically a hash function for HMAC)

- **Example ciphersuites:**
  - RSA, AES128/CBC, SHA-256
  - DH_anon, 3DES/CBC, SHA-1 **(beware DH_anon!)**
  - null, null, null

- Henceforth assume RSA key establishment...
Handshake protocol

Warning:

• attacks on SSL sometimes involve rollback to deprecated algorithms that your crypto library still supports

• YOUR responsibility to make sure only current algorithms are enabled
Handshake protocol

1. C->S: Suites_C, N_C
2. S->C: Suite_S, Cert(S; CA), N_S
3. C: PS = rand(); // premaster secret
ePS = Enc(PS; K_S)
4. C->S: ePS
5. S: PS = Dec(ePS; k_S)
6. C and S:
   MS = PRF(PS, "master secret"; N_C+N_S);
   derive connection state from MS
   by splitting into bits
Handshake protocol

See online notes for some omitted details:

• Verify that client and server have agreed on same keys

• **Unilateral vs. mutual authentication:**
  – **unilateral:** server authenticates to client
  – **mutual:** server authenticates to client and client authenticates to server
Upcoming events

- [Fri] A4 due; happy Dragon Day!
- [next week] Happy Spring Break!

Do not believe anything just because you heard it from a seeming authority. – The Buddha