Lecture 14: Authenticating Machines
Where we were…

- **Authentication**: mechanisms that bind principals to actions
- **Authorization**: mechanisms that govern whether actions are permitted
- **Audit**: mechanisms that record and review actions
Where we were…

- **Authentication**: mechanisms that bind principals to actions
  - Authenticating Humans
  - Authenticating Machines
  - Authenticating Programs
Authentication Techniques

1. 123456
2. password
3. 12345678
4. qwerty
5. 12345
SSL/TLS Handshake

Version, cipher suites, nonce

ClientHello

ServerHello

ServerKeyExchange

ClientKeyExchange

ClientKeyExchange

ChangeCipherSpec

ChangeCipherSpec

Encrypted Messages

Compute master secret

Version, cipher suite, nonce, certificate

(optional)

Compute master secret
SSL/TLS Handshake

Version, cipher suites, nonce

rC, [ECDH,…]

rS, ECDH, \(g^a\) \(\text{rt} \ S\)

Compute
\(ms_p = g^{ab}\)
\(ms = \text{PRF}(ms_p, rC, rS)\)

ChangeCipherSpec

Encrypted Messages

Version, cipher suite, nonce, certificate

Compute
\(ms_p = g^{ab}\)
\(ms = \text{PRF}(ms_p, rC, rS)\)

ChangeCipherSpec
Certificates

- **Digital certificate** is a signature binding together:
  - **identity** of principal
  - **public key** of that principal (might be encryption or verification key)
- **Notation**: $\text{Cert}(S; I)$ is a certificate issued by principal $I$ for principal $S$
  - let $b = \text{id}_S, \text{K}_S$
  - $\text{Cert}(S; I) = b, \text{Sign}(b; k_I)$
  - Issuer $I$ is certifying that $\text{K}_S$ belongs to subject $\text{id}_S$
- **Fingerprint**: $H(\text{Cert}(S; I))$
X.509 certificates

[ RFC 5280 ]

Contents of certificate:

• serial number (unique within certs issued by this issuer)
• issuer distinguished name
• validity interval (start and end time)
• subject distinguished name
• subject public key (and the name of the algorithm)
• extensions...
• issuer's signature on the above (and the name of the algorithm)
X.509 distinguished names

• Originally designed for general purpose directory services
• As commonly used in X.509 certificates:
  • Common name (CN): e.g., a person's full name, a server's name or domain name
  • Organizational unit (OU): e.g., Finance, HR, CS
  • (might be many nested OUs...)
  • Organization (O): e.g., Cornell, Google
  • Other fields: Street Address, Locality, State, Country, Postal Code, etc.
Certificate examples

- https://www.google.com
- https://www.cs.cornell.edu
X.509 certificate extensions

• **Informational extensions:** extra information about certificate, issuer, subject

• **Constraint extensions:** warn user of certificate about what not to do with it

• **Critical flag:** if set, software must process extension or reject certificate
Some informational extensions

- **Key usage:**
  - digital signature
  - encryption of session keys
  - encryption of data
  - verification of certificates (i.e., issuer key)
  - (others)

- **Alternative name:** anything that doesn't fit in a distinguished name, e.g., email address, URL, IP address
Public-key infrastructure (PKI)

- System for managing distribution of certificates
- Two main philosophies:
  - Decentralized: anarchy, no leaders
  - Centralized: oligarchy, leadership by a few elite
PKI Example 1: PGP

- Uses a decentralized PKI philosophy
- "Pretty Good Privacy" [Zimmerman 1991]
  - toolset for PKI, encryption, signing of files and emails
  - OpenPGP is implemented by GNU Privacy Guard (GPG)
- Users manage a keyring:
  - Alice has her own key in her keyring
  - When Alice meets up with Bob at a key-signing party...
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
Finding a useful certificate

**Certificate chain:** sequence of certificates that certify each other

- on one end, a certificate for the principal you want to authenticate
- on the other end, a certificate for a principal you already know: the *root of trust*
- you must trust every issuer in the chain to issue certificates
A constraint extension

• "Basic constraint": two values:
  • a Boolean: is this key permitted to be used to verify other certificates? i.e., can it be an issuer's key?
    • At best redundant w.r.t key usage extension, which itself is more precise
  • an integer: number of intermediate certificates permitted to follow this one in a chain
  • ought to be marked critical
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

• 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
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
  • 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 \textit{cold storage}

• CA private signing key must be \textit{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>..</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Machines authenticating.</td>
<td>Passwords, biometrics</td>
<td>Tokens, CAs as used in web</td>
</tr>
<tr>
<td>..</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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
Authority

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