Lecture 7: Public-key Infrastructure
Plan

* Recap: Digital Signatures
* Signatures in practice
* Public-key infrastructure (PKI)
  - API/Goal
  - Common strategies
  - Common pitfalls

Logistics

* Lab 1 theory & code due tomorrow 10pm ET
* Lab 2 out on 10/1
Recap: Digital Signatures

Key idea: Message integrity w/o shared secret

(Kgen, Ksign, Kverify)

"Gold standard" security defn

EUF-CMA - Existential unforgeability under chosen msg attack

"Attacker can't forge signature on any msg even after seeing sigs on msgs of its choice"

1) To define precisely use a game

\[ \text{chall} \rightarrow \text{adv} \]

("Details matter"

2) This is just one of many possible security defns for signatures.

\( \rightarrow \text{random msg attack, out-time key leakage} \)

But, this is the most standard, most useful

Hash-based signatures

* We saw a tree-based construction

* Take away: hash fn, PRF, are enough to build secure digital sig schemes.
Signatures in practice (briefly)

- One of the most widely used crypto tools
  - HTTPS
  - Software updates
  - Encrypted messaging
  - SSH
  - VPN
  - Essentially any protocol that sends msgs over the Internet

- Two widely used protocols... both use “hash & sign”
  - RSA (classier, going away)
  - EC-DSA + friends (extremely popular)
    (both based on hard problems in number theory)

<table>
<thead>
<tr>
<th></th>
<th>Pk size</th>
<th>Sig size</th>
<th>sign/s</th>
<th>veils</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPHINCS+</td>
<td>32b</td>
<td>8000b</td>
<td>5</td>
<td>750</td>
</tr>
<tr>
<td></td>
<td>sk: 64b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RSA</td>
<td>256b</td>
<td>256b</td>
<td>2,000</td>
<td>50,000</td>
</tr>
<tr>
<td></td>
<td>sk: 32b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECDSA 256</td>
<td>32b</td>
<td>64b</td>
<td>42,000</td>
<td>14,000</td>
</tr>
<tr>
<td>(Schnorr, Ed25519)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHA256 Hash</td>
<td>64 bytes</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \approx 10,000,000 / s \]
Choice of sig schemes

- 99% of time, use ECDSA (or modern variant)
- In rare cases, want to choose a diff scheme.

* Care about sig length?
  * Shorter (32-byte) sigs using fancier crypto (pairings)

* Want short signature + simple implementation + have some design flexibility
  * stateful hash-based signatures (used for software updates)

* Need PQ security? RSA, ECDSA are not!
  * No known quantum attacks on the hash-based sig scheme we saw
  * Other PQ schemes based on fancier crypto (lattices)

\[
\text{NIST is in the middle of developing new PQ crypto algs. NSA stated goal to transition to only PQ-safe algs at some point...}
\]
Key Take-aways

1. In practice, usually use ECDSA.

2. Mental model:
   - pub key = 32 bytes
   - sig = 64 bytes
Public-key infrastructure (PKI)

Last summer...

The right image...

How do I know it was Dean who sent me this email?

Now that we have signatures, answer is clear!

But where do we get VKDean?
Option: Use public key as name.

Dean's "name" is the vk.
Instead of calling him "Dan",
call him 0x2E5C9DB3...0668

- Can imagine that at birth, we're each given an (sk, vk) pair. Everyone calls us by vk.
This sort-of works! Used in Bitcoin & friends, also for hidden services, ...

Problem: Cumbersome. Hard to remember 32b names.

Problem: What happens if you lose your secret key? Or if it gets stolen?
PKI is all about mapping...

- human-intelligible names to public keys.
- email addr
- domain name
- legal entity
- phone #
- kerberos ID

Can think of PKI as having the API (grossly simplified)

\[ \text{IsKeyFor(vk, <name>)} \rightarrow \{0, 1\} \]

* Many many ways to implement a PKI... we will see some.
* But all serve this same purpose.
* No "perfect" solution here — lots of trade-offs.

We will look at a few common schemes...
Trust on first use (TOFU)

→ Accept only first key you see for a name.

Client keeps a cache = 🌐

IsKeyFor (vk, name):

if name not in cache:
    cache [name] = vk
    return true

else:
    return rk == cache [name]

SSH uses TOFU
(Could use this in my email example. Protection if have already gotten email from Dean)

Pros:
- Simple
- Easy to understand
- Surprisingly effective - protects you against an attacker that hijacks 2nd connection

Cons:
- No protection on first communication
- What happens when key changes?

→ SSH: Warn ... then what?
Trust on first use (TOFU)

Accept only first key you see for a name.

\[ \hat{pk}_{\text{Dean}}, \text{msg}, \sigma \]

\{ (\text{dean@mit.edu, } \hat{pk}_{\text{Dean}}) \}

Check \( \hat{pk}_{\text{Dean}} = pk_{\text{Dean}} \)

Verify \( \sigma \) on msg...
Certificate-Based System

Let certification authorities (CAs) manage name → key mapping

Client keeps a list of known CAs’ verif keys:

\[ \text{CAs} = \{ \text{vk}_\text{verisign}, \text{vk}_\text{google}, \ldots \} \]

Client accepts \((\text{vk}, \text{name})\) pair if known CA signed it.

CAs “attest” to \text{name} → \text{vk} mappings.

\[
\text{IskeyFor}((\text{vk}, \sigma), \text{name}): \\
\text{For each } \text{vk}_{\text{CA}} \text{ in CAs:} \\
\quad \text{if } \text{Verify}(\text{vk}_{\text{CA}}, (\text{vk}, \text{name}), \sigma) \\
\quad \quad \text{return true} \\
\quad \text{return false}
\]

When a client generates a new keypair, it must get a CA to sign its \text{vk}
Certificate-Based System

→ Let certification authorities (CA) manage have → key mapping

0

PK Dean, Jm ± T J

k & Msg, Isg

{ pkmit, - - -

Verfi (pkmit, (phoean, deamit.edn), omi) = 1

Verify & Dean on msg...

To get a cert, need to talk to CA...
Certificate Issuance

\[ (sk, vk) \leftarrow \text{Gen}(1) \]

\[ \text{CA (sk}_{\text{ca}} \]

\[ \sigma \leftarrow \text{Sign}(sk_{\text{ca}}, (vk, me@mit.edu)) \]

\[ \text{Verify that I own me@mit.edu} \]

Common extension: Accept a \((vk, name)\) pair if it's signed by someone whose key was signed by a known CA

Lots of extra metadata in cert: Expiration date, ...

Used on web (HTTPS/TLS), code signing, S/MIME, ...

...also at MIT

Pros:
- Client only needs a few vks — scales well!
- Client can choose which CAs to trust
- No online interaction w/ CA

Cons:
- Weakest link security — attacker who compromises one CA can impersonate anyone!
- Validation is typically pretty weak... TOFU almost...
There are many variants on certificate-style systems - key directory, web of trust,...

"Key" idea: To prove (vk, name) binding, I can give you signature on (vk, name) from someone you trust.

In practice, where does list of known CAs come from?

- Usually packaged w/ browser or OS
- Includes all sorts of sketchy-looking CAs ("AAA Certificate Services; govt, etc.")
- Many potential points of failure - stealing any CA sk is enough to impersonate any website

2011: Digisitar signing key stolen
- Attackers used it to issue cert for google.com
- Used to decrypt Gmail traffic in Iran
- Browsers pull Digisitar from list of known CAs
- Dutch govt websites break

(IS attacker targeted lower profile domain, would probably not have been discovered so quickly)
How to detect "rogue" CA?

- Have client software look for certain misbehavior, e.g. Chrome has list of Google CA, hardcoded.
- If CA issues a rogue Google CA, Chrome will (I believe) notify Google.
- Doesn't really solve the problem.
- Only works for friends of Google.
- If client knew what the right CA is, wouldn't need PKI.

Certificate Transparency (some browsers, sort of)

- Require CAs to publish all CAs they sign in a public log ... many logs run by many different orgs.
- MIT.edu can inspect logs regularly to make sure that no CA has issued rogue CAs for its domains.
- In theory, when browser gets a CA from a web server, it can "audit" the CA by checking that it appears in the log.

- Lots of messy implementation details
  - prevent logs from cheating
  - ensure that everyone sees same log
  - ensure that client can audit recently issued CAs
  - privacy issues w/ auditing
Revocation

- After a CA has issued a cert, they may want to revoke it → make sure clients reject it in the future.

**Why?**
* Site owner has their secret key stolen (Heartbleed) -> 2011
* Site owner realizes they generated key using bad randomness (Debian bug) -> 2008
* MIT student graduates, account inactivated

Once a CA has signed a cert and given it out, CA can't "unsign" it.

**Approach: Expiration**
* Cert has expiration date, clients will reject cert after that date
* If expiration date is not far away, this handles many routine revocation cases
  e.g. MIT certs expire June 30 every year.
Approach: Software vendor (e.g., Mozilla) ships update to client w/ full list of revoked certs.

- Window of vulnerability as long as update latency.
- b/w storage cost after wave of revocations.

"CRLSet"  "CRLike"  

Approaches: fallen out of favor

- Certificate revocation list (CRL)
  - Ask CA for list of all revoked unexpired certs
  - Expensive after a wave of revocations.
  - What happens if can’t reach CA server?

OCSP
  - Ask CA each time you use cert
  - Privacy issues.
  - “Stapling” = short-lived certificate.
Bottom line:
PKI is about names ⇒ public keys
Key idea: Certificates signed attestation of name ⇒ vk binding
Key challenge: Revocation stolen key, invalid binding

The End