Lecture 7: Public-key Infrastructure
Plan

* Recap: RSA
* Signatures in practice
* Public-key infrastructure (PKI)
  1) Key is name
  2) TOFU
  3) Key directory
  4) Certificates

[Set up laptop]

Logistics

* Lab 1 due tomorrow
* Lab 2 out Friday

Extensions
Recap: RSA Signatures

RSA gives trapdoor OWP

\( \text{Gen}() \rightarrow (sk, pk) \)

\( F(pk, x) \rightarrow y \)

\( I(sk, y) \rightarrow x \)

\( \rightarrow \text{Gives digital sigs + many other things (pub key encryption)} \)

Digital sig from trapdoor OWP

using hash fn \( H: \mathbb{Z}_q \times \mathbb{Z}_q \rightarrow \mathbb{Z}_q \) “modeled as random oracle”

\( \text{Sign}(sk, m) := I(sk, H(m)) \)

\( \text{Ver}(pk, m, \sigma) := \{ F(pk, \sigma) = H(m) \} \)
Recap

RSA Construction

\[ \text{Gen}(X) \rightarrow * \text{ choose random } 2\text{-bit primes } p, q \]
\[ \text{s.t. } \phi(N) = (p-1)(q-1) \text{ and } e \text{ are relatively prime} \]
\[ * \text{Find } d \text{ s.t. } ed \equiv 1 \mod \phi(N) \]
\[ * (sk, pk) = (d, N), \ N = pq \]

TDP is over space \( X = \mathbb{Z}_N^\times \)

\[ F(pk = N, x \in \mathbb{Z}_N^\times) := x^e \mod N \]
\[ I(sk = (d, N), y \in \mathbb{Z}_N^\times) := y^d \mod N \]

Correctness: \( \forall x \in \mathbb{Z}_N^\times \)

\[ I(sk, F(pk, x)) = (x^e)^d \mod N \]
\[ = x^{ed} \mod N \]
\[ = x^{1 + k\phi(N)} \mod N \]
\[ = x \cdot (x^{\phi(N)})^k \mod N \]
\[ = x \mod N \]

By Euler's Theorem

Security: By assumption
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"

  $\Rightarrow$ RSA (classie, going away)
  $\Rightarrow$ EC-DSA + friends (extremely popular)

  (both based on hard problems in number theory)
## Choice of sig schemes

<table>
<thead>
<tr>
<th>Scheme</th>
<th>PK size</th>
<th>Sig size</th>
<th>Sign/s</th>
<th>Ver/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPHINCS+ - 128</td>
<td>32B</td>
<td>8000B</td>
<td>5</td>
<td>750</td>
</tr>
<tr>
<td>~ 2010s</td>
<td>sk: 64B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RSA - 2048</td>
<td>256B</td>
<td>256B</td>
<td>2,000</td>
<td>50,000</td>
</tr>
<tr>
<td>~ 1970s</td>
<td>sk: &quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECDSA 256 (Schnorr, Ed25519)</td>
<td>32B</td>
<td>64B</td>
<td>42,000</td>
<td>14,000</td>
</tr>
<tr>
<td>~1990s</td>
<td>sk: &quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHA256 Hash 64 bytes</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>~10,000,000/s</td>
<td></td>
</tr>
</tbody>
</table>

- 99% of the time, use ECDSA (or modern variant)

- In rare cases, want to choose a different scheme:
  - Post-quantum security (RSA and ECDSA arent! Hash-based sigs seem to be also lattice-based)

- Extra features: aggregation, blind signing, etc...
Public-key infrastructure (PKI)

Last year

Dear

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?
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)

\[ \text{IsKeyFor}(vk, \langle \text{name} \rangle) \rightarrow \{0, 1\} \]

* 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.
Common approaches to PKI

- Names as keys
- Trust on First use (TOFU)
- Key directory
- Certificates

Three questions: (see also "Zooko’s triangle")

1. Who do you trust for name resolution?
   i.e. who does attacker need to compromise to cause client to accept wrong key?

2. How does key revocation work?
   i.e., How does entity update their key?

3. How easy is it for the end user?
1. Names as keys

Dean's "name" is the vk.
Instead of calling him "Dan,"
call him 0x2EEC9DB3...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, ...

**Trust?**
No one. "Best possible" security.

**Revocation?**
No plan. Don't lose your secret keys!

**Usability?**
Cumberbome. Hard to remember 32B names.
2. Trust on first use (TOFU)

$\Rightarrow$ Accept only first key you see for a name.

Client keeps a cache = $\exists S$

```python
IsKeyFor(vk, name):
    if name not in cache:
        cache[name] = vk
        return True
    else:
        return vk == cache[name]
```

Used in SSH
(Could use this in my email example: Protection if have already gotten email from Dean)

Trust?
First connection to other party

Revocation?
Unclear. What do you do when you see an SSH key warning or Signal "security #" change?

Usable?
Pretty convenient, modul revocation.
3. Key directory

Used in WhatsApp, Signal, iMessage

\[\text{IsKeyFor}(vk, name):\]
* ask server for vk for name.
* return vk

Extensions:
* UI to let users see keys for each other
  \(\Rightarrow\) catch evil server?
* Public "audit log" - server has to publish tamper-evident log of all (name, vk) pairs
  \(\Rightarrow\) catch evil server? Meta AKD
  (Certificate transparency is similar)

Trust?
- Key server. 😞 Maybe the mitigations help?

Revocation?
- Simple & fast ⇐ Big benefit

Usability?
- Simple
4. Certificates

Let certification authorities (CA) manage name → key mapping

Each participant holds sig-veris public keys for many "certsicate authorities" CAs.

Packaged w/ browser or OS or crypto lib

"Key idea: To prove (name, vk) binding, a remote party can send you a sig on the (name, vk) pair from a CA that you trust.

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

als at MIT

[Pub key certs introduced in 1978 by Loren Kohnofeller in B.S. thesis.]
Client accepts \((vk, name)\) pair if known CA signed it.

\[ \Rightarrow \text{CAs "attest" to name } \Rightarrow vk \text{ mappings.} \]

\[
\text{CAs} = \{ vk_{\text{version}}, vk_{gpg}, \ldots \}
\]

\[
\text{IsKeyFor}((vk, \sigma), name):
\]

\[
\text{For each } vk_{CA} \text{ in CAs:}
\]

\[
\text{if Verify}(vk_{CA}, (vk, name), \sigma) \text{ return true}
\]

\[
\text{return false}
\]

---

**Demo**

- Show cert & chain of trust for mit.edu

- Dump CRL data
  
  ```
  openssl crl -inform DER -text -noout -in <CRL>
  ```

- Q: Why intermediate CAs?
Obtaining a certificate

1. To obtain a cert, entity proves that it owns "name" to CA

   ↓ How?
   MIT: Kerb login
   Let's Encrypt: post string at mysite.com
   Add DNS TXT record
   "Extended validation" legal process

2. CA attests to (name, vk) binding with sig under its signing key

   When a client generates a new keypair, it must get a CA to sign its vk

Details

Common extension: Cert "chains"

CA → intermediate CA → vk

why?

Lots of extra metadata in cert: Expiration date, ...
Certificates: Trust?

⚠️ Any malicious/compromised CA can issue certs for any domain.
  → Your browser trusts many sketchy CAs (goats, random businesses, etc.)
  → "AAA cert service" can issue cert for mit.edu... you'll never know

2011: - Digicert signing key stolen
  - Attackers used it to issue cert for google.com
  - Used to decrypt Gmail traffic in Iran

  - Browsers pull Digicert from list of known CAs
  - Dutch govt websites break

"Certificate transparency" is one partial answer...
Certificates: Revocation?

- After a CA has issued a cert, it 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
* Crypto standards change (SHA1, RSA1024, ...)

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.
  e.g. Let's Encrypt uses 90-day expiration
**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

“CR-Set” “CR-Like”

**Approaches: fallen out of favor**

- Certificate revocation list (CRL)
  - Expensive after a wave of revocations
  - What happens if can’t reach CA servers?

**OCSP**

- Ask CA each time you use cert
  - Browsing history leaks to CA
  - CA on critical path of page load

“Stapling” is short-lived cert
Certificates: Usability?

- Used to be
  * annoying to get cert $$$
  * also a pain to set up in web server, etc.

- Let's Encrypt CA changed everything (ca. 2014)
  * free
  * relatively easy to set up for web apps

- Still annoying in other contexts (e.g. email)
Bottom line:
PKI is about names \Rightarrow public keys

Key idea: Certificates signed attestation of name \Rightarrow vk binding

Key challenge: Revocation stolen key, invalid binding

The End
How to detect "rogue" CA?

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

Certificate Transparency (some browsers, sat of)

- Require CAs to publish all certs 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 certs for its domains

- In theory, when browser gets a cert from a web server, it can "audit" the cert 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 certs
  ➔ privacy issues w/ auditing