Lecture 7: Public-key Infrastructure

6.1600 - MIT

Fall 2023

Corrigan-Gibbs & Zeldovich

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
  extensions
* Lab 2 out Friday

Recap: RSA Signatures

RSA gives trapdoor OWP

$$\left.\begin{array}{l} Gen() \to (sk, pk) \\ F(pk, x) \to y \\ I(sk, y) \to x \end{array}\right\} x, y \in \mathcal{X}$$

↳ Gives digital sigs + many other things (pub key encryption)

Digital sig from trapdoor OWP

using hash fn $H: \{0,1\}^* \to \mathcal{X}$ "modeled as random oracle"

$$Sign(sk, m) := I(sk, H(m))$$

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

Recap

RSA Construction w/ public exponent e $\left\{ \begin{array}{l} \text{small} \\ \text{prime} \end{array} \right.$

Gen(λ) $\rightarrow$ * Choose random λ-bit primes p, q
s.t. $\phi(N) = (p-1)(q-1)$ and e are relatively prime

* Find d s.t. $ed \equiv 1 \mod \phi(N)$

e.g. Extended Euclidean alg.

* $(sk, pk) = ((d, N), \; N = p \cdot q)$

TDP is over space $X = \mathbb{Z}_N^*$

$F(pk = N, x \in \mathbb{Z}_N^*) := x^e \mod N$

$I(sk = (d, N), y \in \mathbb{Z}_N^*) := y^d \mod N.$

Correctness: $\forall \; x \in \mathbb{Z}_N^*$

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

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"
  ↳ RSA (classic, ... going away)
  ↳ EC-DSA + friends (extremely popular)
  (both based on hard problems in number theory)

Choice of sig schemes

	Pk size	Sig size	sign/s	ver/s
PQ Standard SPHINCS+-128 ~2010s	32B sk: 64B	8000B	5	750
RSA-2048 ~1970s	256B sk: "	256B	2,000	50,000
ECDSA 256 (Schnorr, Ed25519) ~1990s	32 B sk: "	64B	42,000	14,000

← Short msg

similar to what we saw w/ siglen opts

widely used

SHA256 Hash 64 bytes ≈ 10,000,000/s

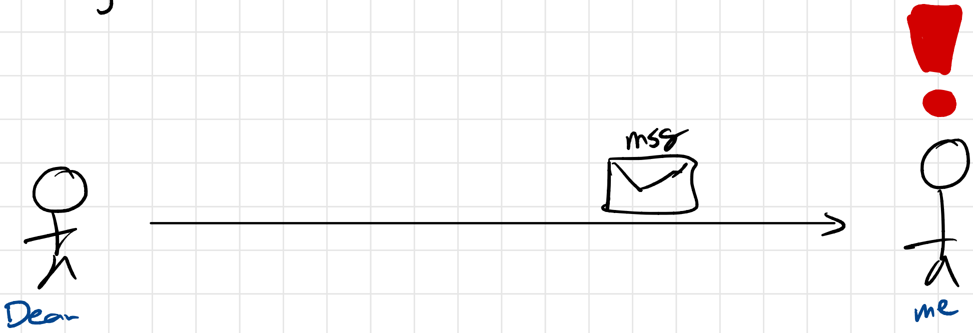
- 99% of time, use ECDSA (or modern variant)

- In rare cases, want to choose a dif scheme.
  * Post-quantum security (RSA and ECDSA aren't! Hash-based sigs seem to be. Also lattice-based.)

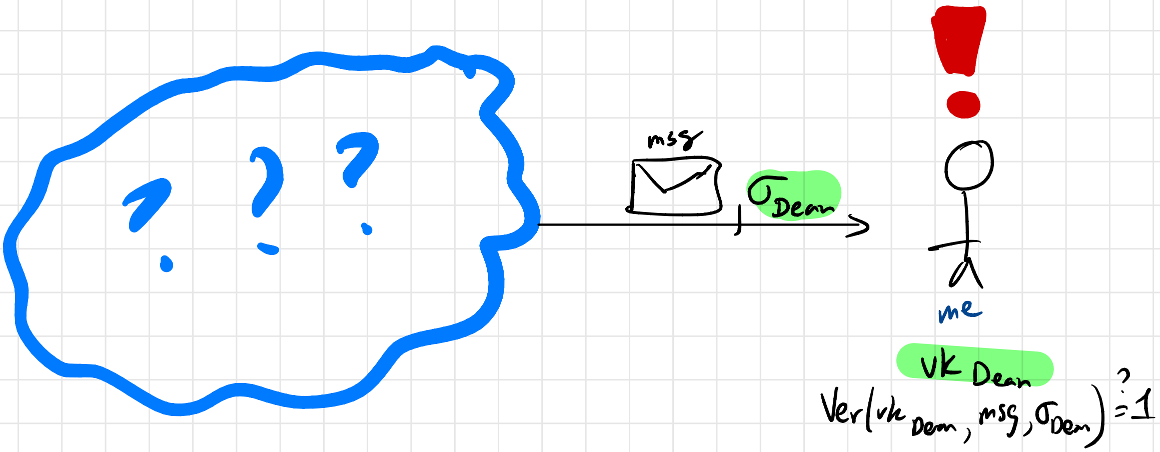
  * Extra features: aggregation, blind signing, etc...

Public-key infrastructure (PKI)

Last year



The right image...



$Ver(vk_{Dean}, msg, \sigma_{Dean}) \overset{?}{=} 1$

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

Now that we have signatures, answer is clear!
(add vk_{Dean})

But where do we get vk_{Dean}?

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)

$$\mathrm{IsKeyFor}(vk, \langle name \rangle) \longrightarrow \{0, 1\}$$

* 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 $\underbrace{\text{0x2EEC9DB33.....0668}}_{\text{32 bytes}}$

- 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 Tor hidden services, ...

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

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

Usability? Cumbersome. Hard to remember 32B names.

2. Trust on first use (TOFU)

→ Accept only first key you see for a name.

Client keeps a cache = {} ← dictionary/hash table

```
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 modulo revocation.

3. Key directory

Used in WhatsApp, Signal, iMessage

```
IsKeyFor(vk, name):
    * ask server for vk for
      name.
    * return vk
```

Extensions:

* UI to let users see keys for each other
  ↳ catch evil server?
* Public "audit log" – serve has to publish tamper-evident log of all (name, vk) pairs
  ↳ 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 (CAs) manage name → key mapping

Each participant holds sig-verif. public keys for many "certificate authorities" CAs.
(certification authorities)
↳ 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, ...
... also at MIT

[Pub-key certs introduced in 1978 by Loren Kohnfelder in B.S. thesis.]

$\Rightarrow$ Client accepts (vk, name) pair iff known CA signed it.

↳ CAs "attest" to name $\rightarrow$ vk mappings.

```
CAs = { $vk_{verisign}$, $vk_{google}$, ... }
IsKeyFor(($vk$, $\sigma$), name):
    For each $vk_{CA}$ in CAs:
        if Verify($vk_{CA}$, ($vk$, name), $\sigma$)
            return true
    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
     - lets 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"

$$\begin{tikzcd} \text{CA} \arrow[r] & \text{intermediate CA} \arrow[r, bend left] & \text{vk} \end{tikzcd}$$

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 (govts, random businesses, etc.)

→ "AAA cert services" can issue cert for mit.edu... you'll never know

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

- Browsers pull Diginotar 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 $\longrightarrow$ 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

"CRLSet" "CRLite"

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
  - browsing history leaks to CA
  - CA on critical path of page load

"Stapling" ~ 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 ⇒ public keys

Key idea: Certificates — signed attestation of name ↦ 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 pks 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, sort 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
  ⋮