Lecture 7: Public-key Infrastructure

6.5060

Fall 2021

Corrigan-Gibbs, Devadas,

Kalai, Zeldovich

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

↳ unlike MAC or password-based auth

↳ really revolutionary — no shared secret!

(Gen, Sign, Verify)

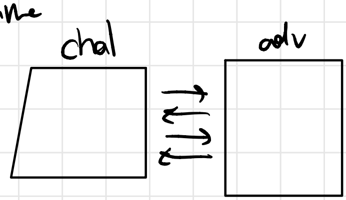
"Gold standard" security defn

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

↳ Details matter!



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

↳ random msg attack, one time, key leakage, ...

↳ But, this is the most standard, most useful

Hash-based signatures

* We saw a tree-based construction
* Take away: hash fns / PRFs 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 (classic, ... going away)
  - ↳ EC-DSA + friends (extremely popular)

(both based on hard problems in number theory)

Short msg

	Pk size	Sig size	sign/s	ver/s
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

similar to what we saw w/ siglen opts

widely used

SHA256 Hash 64 bytes ≈ 10,000,000/s

Choice of sig schemes

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

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

* Want short signatures + 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)
  ↳ 6.875, 6.876

[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.

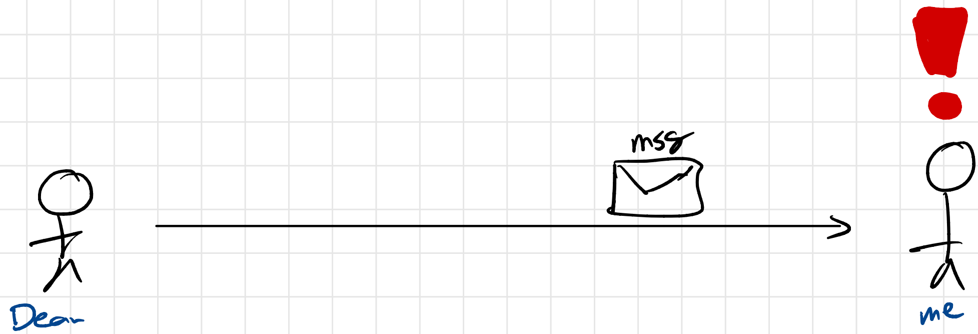
2. Mental model:

   pub key = 32 bytes

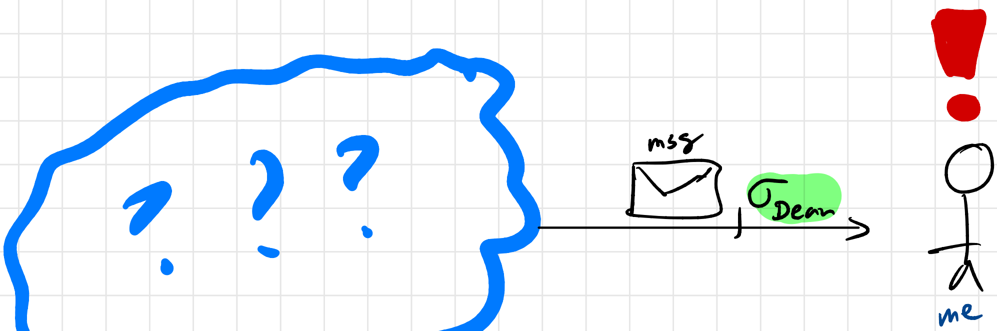
   sig = 64 bytes

Public-key infrastructure (PKI)

Last summer...



The right image...



vk_{Dean}

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

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}?

Option: Use public key as name.

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, ...

Problem: Cumbersome. Hard to remember 32b names.
↳ PKI

Problem: What happens if you lose your secret key? Or if it gets stolen?
↳ Revocation

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.

We will look at a few common schemes...

Trust on first use (TOFU)

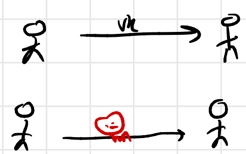
→ Accept only first key you see for a name.

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

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

SSH uses TOFU

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



Pros:

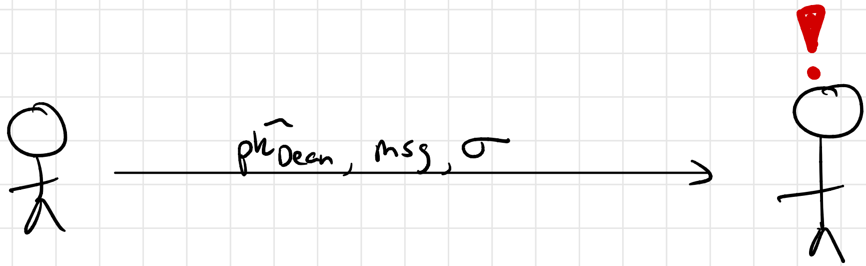
- Simple
- Easy to understand
- Surprisingly effective – protects you against an attacker that hijacks 2nd connection.

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.



$\{(\text{dean@mit.edu}, pk_{Dean})\}$

Check $\hat{pk}_{Dean} = pk_{Dean}$

Verify σ on msg...

Certificate - Based System

→ Let certification authorities (CAs) manage name → key mapping

Client keeps a list of known CAs' verf keys.

$CAs = \{vk_{Verisign}, vk_{Google}, \ldots\}$

⇒ Client accepts (vk, name) pair iff known CA signed it.

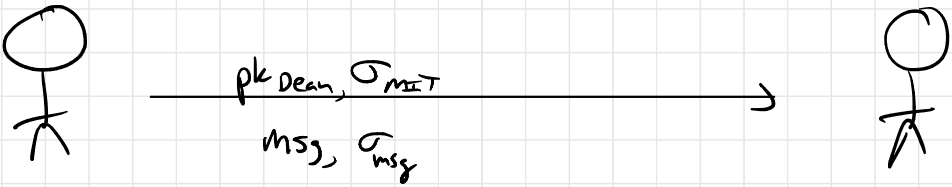
↳ CAs "attest" to name → vk mappings.

```
IsKeyFor((vk, σ), name):
    For each vk_CA in CAs:
        if Verify(vk_CA, (vk, name), σ)
            return true
    return false
```

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

Certificate-Based System

→ Let certification authorities (CAs) manage name → key mapping



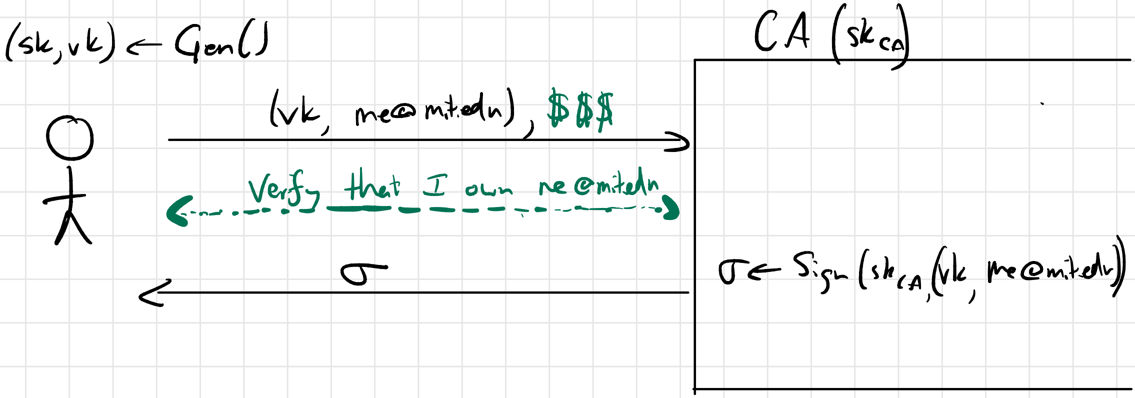
$\{pk_{mit}, \dots\}$

$\text{Verify}(pk_{mit}, (pk_{Dean}, \text{dean@mit.edu}), \sigma_{mit}) \stackrel{?}{=} 1$

Verify σ_{Dean} on msg...

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

Certificate Issuance



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

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

<u>Cons</u>: - Weakest link security — attacker who compromises one CA can impersonate anyone!
- Validation is typically pretty weak... TOFU almost
- Stolen key?

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", govts, etc.)

→ Many potential points of failure — stealing any CA sk is enough to impersonate any website

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 gov't websites break

(If 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 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, 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
  ⋮

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" "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

- privacy issues

↳ "stapling" ≈ short-lived certificate.

Bottom line:

PKI is about names $\Rightarrow$ public keys

Key idea: Certificates — signed attestation of name $\mapsto$ vk binding

Key challenge: Revocation — stolen key, invalid binding.

The End