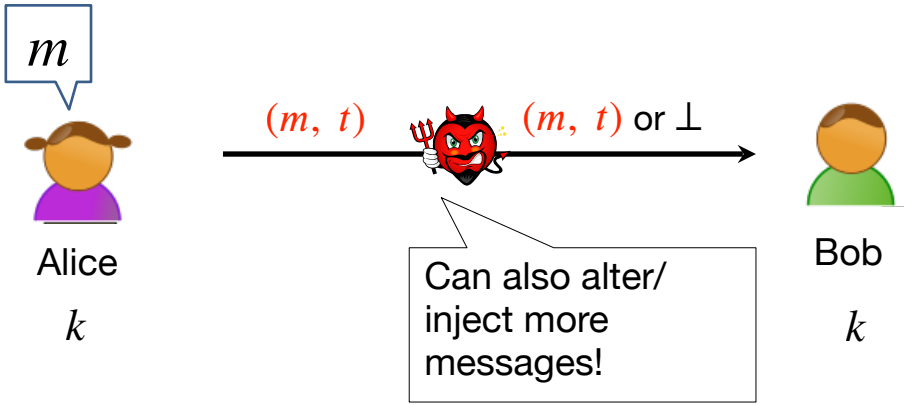
# CIS 5560

# Cryptography
# Lecture 17

# Announcements

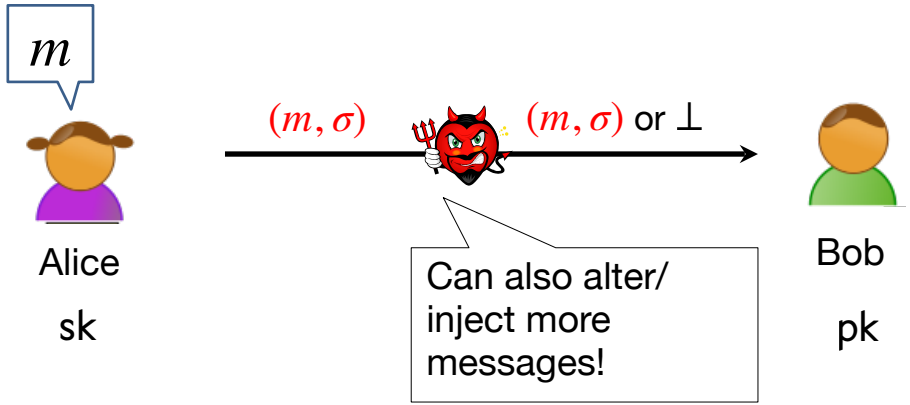
- **Midterm grades have been published**
  - Regrade requests are open

# Symmetric-key Message Authentication



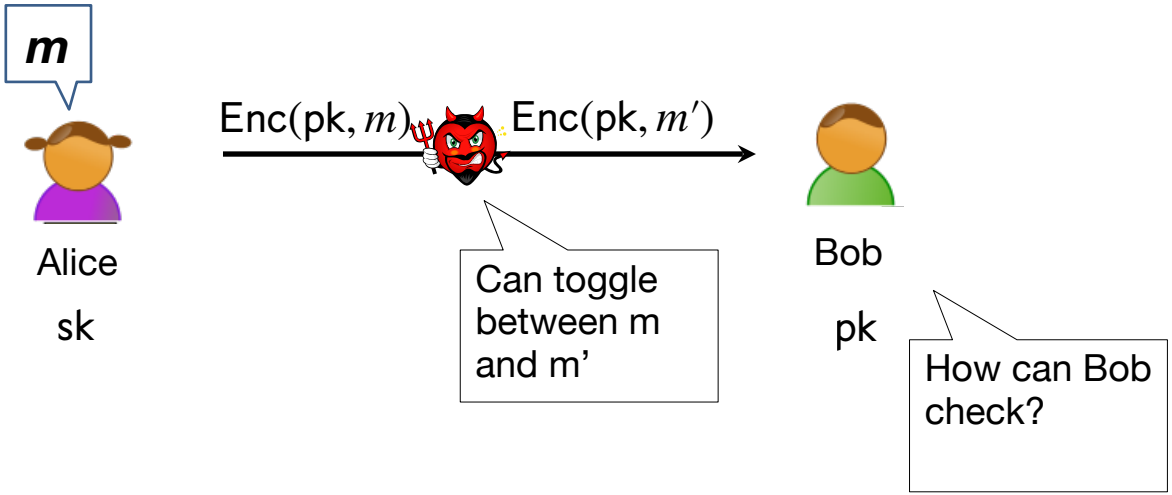

We want Alice to generate a tag for the message *m* which is **hard to generate** without the secret key *k*.

# Public-key Message Authentication?




We want Alice to generate a signature for the message *m* which is **hard to forge** without the secret/signing key *sk*.

# Does PKE not solve this?




Anybody can encrypt, and no way for recipient to check.

# New primitive: Digital Signatures

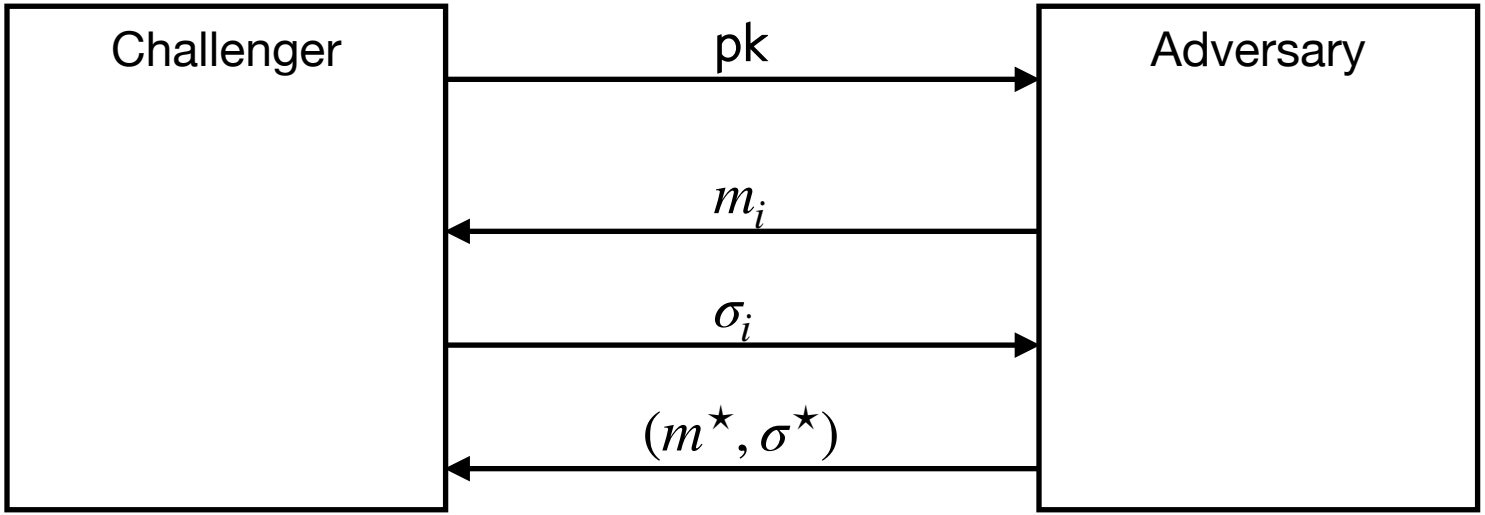
# Digital Signatures: Definition

A triple of PPT algorithms (Gen, Sign, Verify) such that

- Key generation: $\mathsf{Gen}(1^n) \rightarrow (\mathsf{sk}, \mathsf{pk})$
- Message signing: $\mathsf{Sign}(\mathsf{sk}, m) \rightarrow \sigma$
- Signature verification: $\mathsf{Verify}(\mathsf{pk}, m, \sigma) \rightarrow b \in \{0,1\}$

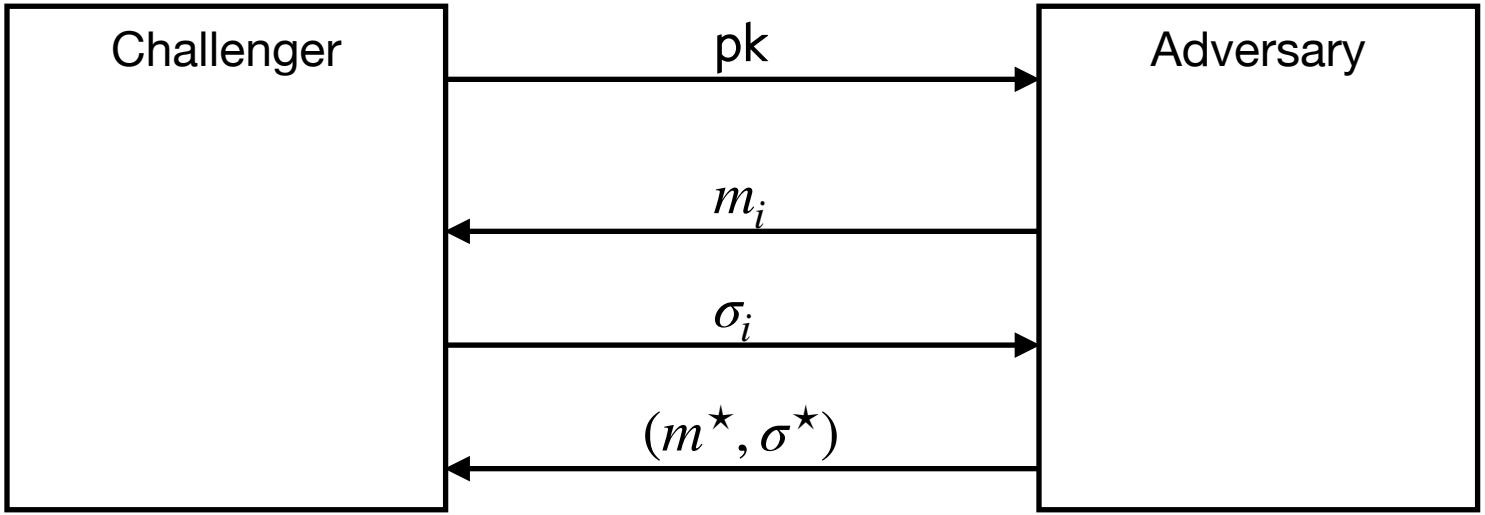
**Correctness:** For all vk, sk, m:
$\mathsf{Verify}(\mathsf{pk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1$

# EUF-CMA for Signatures




$$\Pr\left[\begin{array}{c} m^\star \notin \{m_i\} \\ \text{and} \\ \mathsf{Verify}(\mathsf{pk}, m^\star, \sigma^\star) = 1 \end{array}\right] = \mathsf{negl}(\lambda)$$

# *Strong* EUF-CMA for Signatures




$$\Pr\left[\begin{array}{c}(m^\star, \sigma^\star) \notin \{(m_i, \sigma_i)\} \\ \text{and} \\ \mathsf{Verify}(\mathsf{pk}, m^\star, \sigma^\star) = 1\end{array}\right] = \mathsf{negl}(\lambda)$$

# Digital Signatures vs. MACs

| **Signatures** | **MACs** |
| --- | --- |
| $n$ users require $n$ key-pairs | $n$ users require $n^2$ keys |
| Publicly Verifiable | Privately Verifiable |
| **Transferable** | **Not** Transferable |
| **Provides Non-Repudiation** (is this a good thing or a bad thing?) | Does not provide Non-Rep. |

Let (Gen, Sign,V) be a signature scheme.

Suppose an attacker is able to find $m_0 \neq m_1$ such that

$$\mathbf{V(pk, m_0, \sigma) = V(pk, m_1, \sigma)} \quad \text{for all } \sigma \text{ and keys (pk, sk)} \leftarrow \text{Gen}$$

Can this signature be secure?

- ○ Yes, the attacker cannot forge a signature for either $m_0$ or $m_1$
- ○ No, signatures can be forged using a chosen msg attack
- ○ It depends on the details of the scheme

Alice generates a (pk,sk) and gives pk to her bank.

Later Bob shows the bank a message m="**pay Bob 100$**"
properly signed by Alice, i.e. Verify(pk,m,sig) = 1

Alice says she never signed m. Is Alice lying?

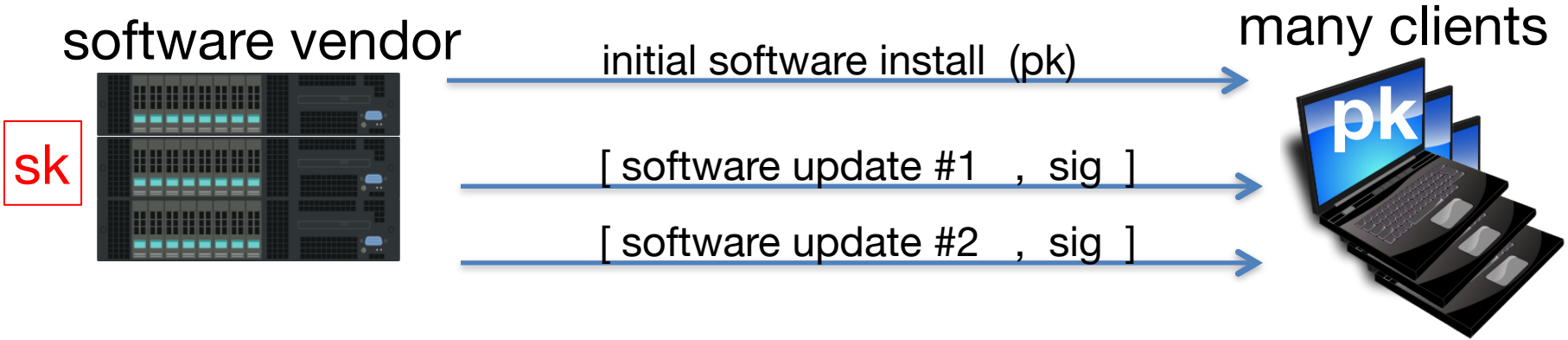
- Alice is lying: existential unforgeability means Alice signed m and therefore the Bank should give Bob 100$ from Alice's account
- Bob could have stolen Alice's signing key and therefore the bank should not honor the statement
- What a mess: the bank will need to refer the issue to the courts

# Applications

# Applications

**Code signing**:

- Software vendor signs code
- Clients have vendor's pk. Install software if signature verifies.

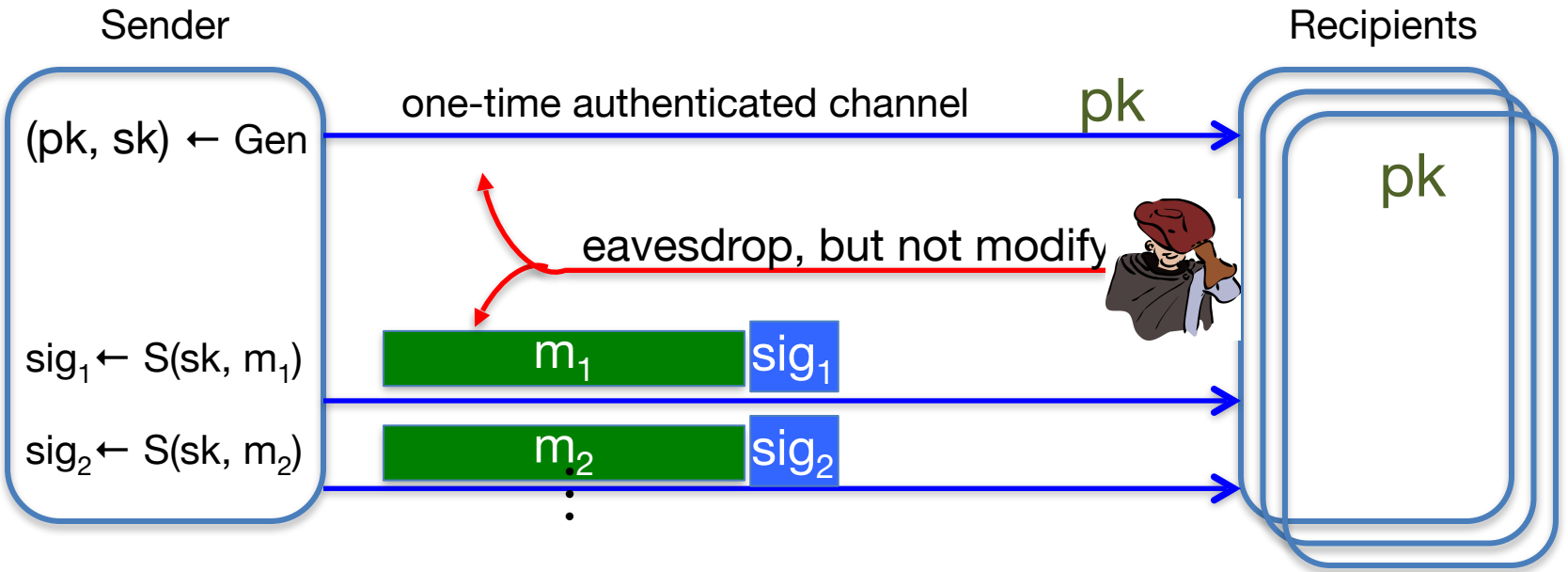


# More generally:

One-time authenticated channel (non-private, one-directional)

$\Longrightarrow$ many-time authenticated channel

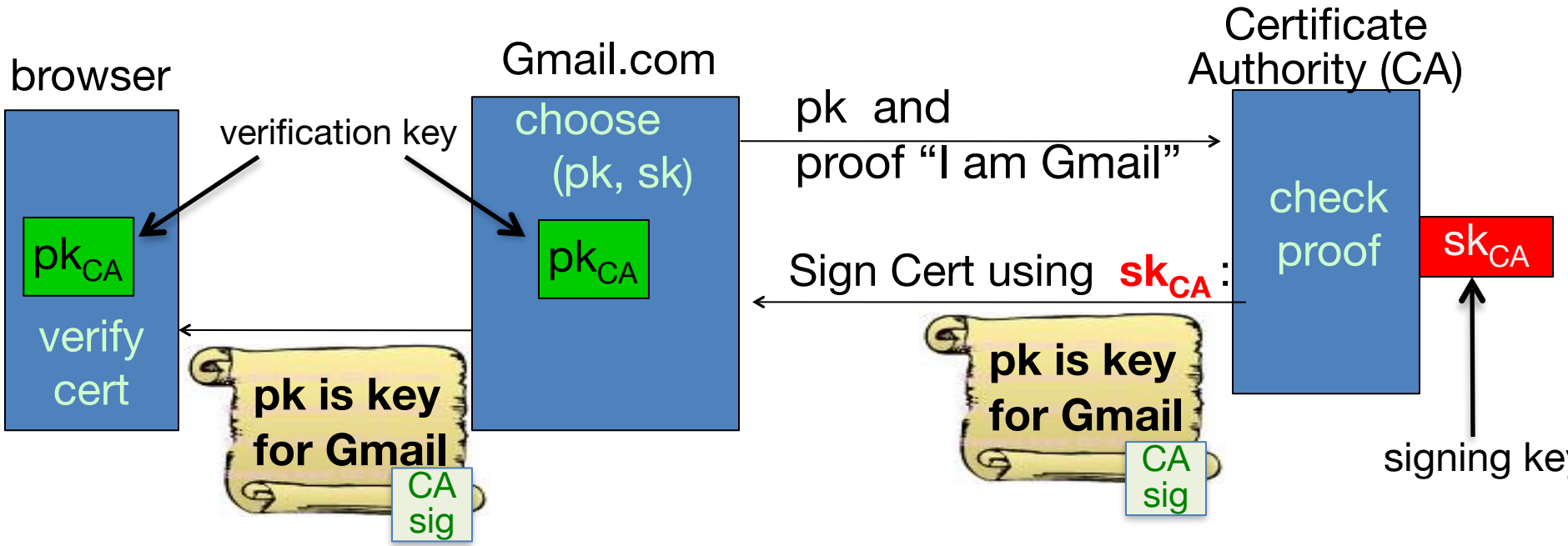
Initial software install is authenticated, but not private



# Important application: Certificates

Problem: browser needs server's public-key to setup a session key

Solution: server asks trusted 3rd party (CA) to sign its public-key pk

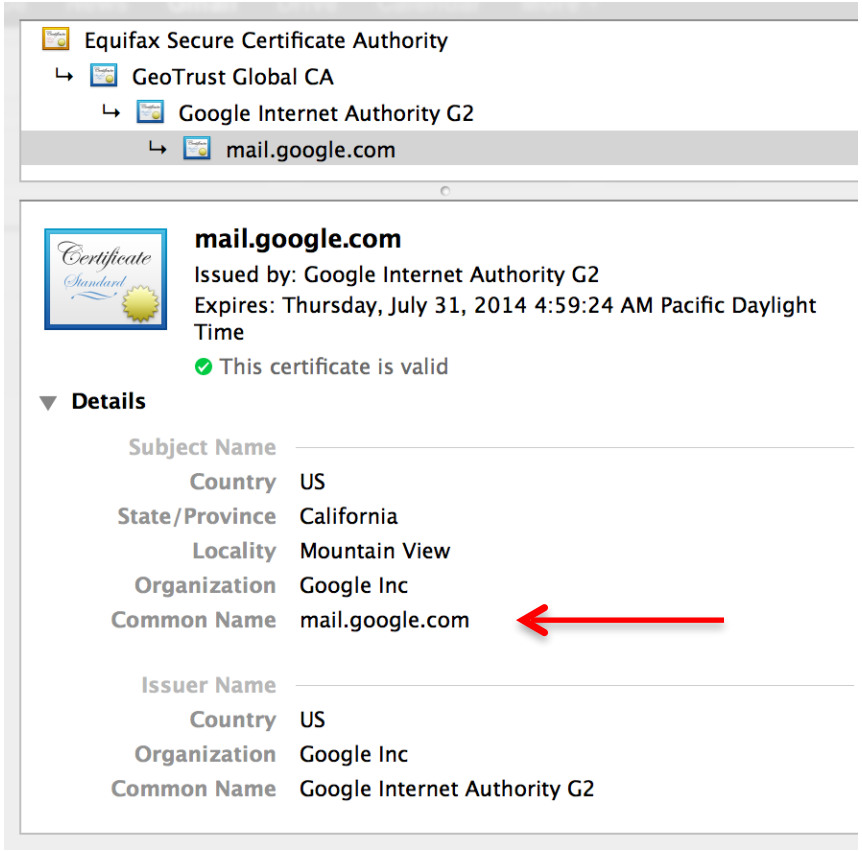



**Server uses Cert for an extended period** (e.g. one year)

# Certificates: example

Important fields:

| | |
|---|---|
| Serial Number | 5814744488373890497 |
| Version | 3 |
| Signature Algorithm | SHA-1 with RSA Encryption ( 1.2.840.113549.1.1.5 ) |
| Parameters | none |
| Not Valid Before | Wednesday, July 31, 2013 4:59:24 AM Pacific Daylight Time |
| Not Valid After | Thursday, July 31, 2014 4:59:24 AM Pacific Daylight Time |
| Public Key Info | |
| Algorithm | Elliptic Curve Public Key ( 1.2.840.10045.2.1 ) |
| Parameters | Elliptic Curve secp256r1 ( 1.2.840.10045.3.1.7 ) |
| Public Key | 65 bytes : 04 71 6C DD E0 0A C9 76 ... |
| Key Size | 256 bits |
| Key Usage | Encrypt, Verify, Derive |
| Signature | 256 bytes : 8A 38 FE D6 F5 E7 F6 59 ... |



What entity generates the CA's secret key $sk_{CA}$ ?

- the browser
- Gmail
- the CA
- the NSA

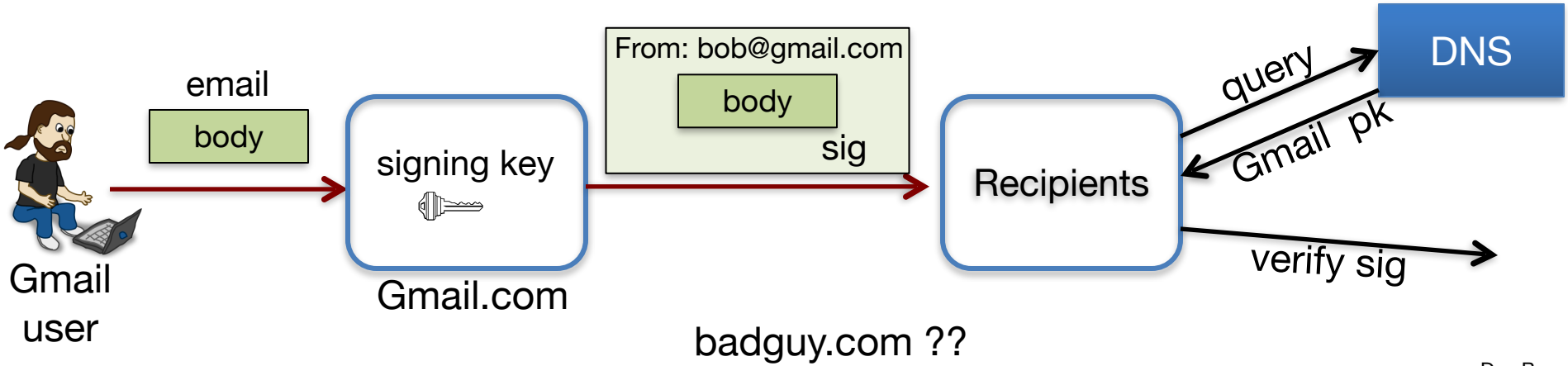
# Signing email: DKIM (domain key identified mail)

Problem: bad email claiming to be from **someuser@gmail.com**

but in reality, mail is coming from domain **badguy.com**

⇒ Incorrectly makes gmail.com look like a bad source of email

Solution: **gmail.com** (and other sites) sign every outgoing mail



# When to use signatures

Generally speaking:

- If one party signs and **one** party verifies: **use a MAC**
  - Often requires interaction to generate a shared key
  - Recipient can modify the data and re-sign it before passing the data to a 3rd party

- If one party signs and **many** parties verify: **use a signature**
  - Recipients **cannot** modify received data before passing data to a 3rd party (non-repudiation)

# Constructions

# “Vanilla” RSA Signatures

Start with any trapdoor permutation, e.g. RSA.

Gen($1^\lambda$): Pick primes $(p, q)$ and let $N = pq$. Pick $e$ relatively prime to $\varphi(N)$ and let $d = e^{-1} \pmod{\varphi(N)}$.

$$\text{sk} = (p, q, d) \text{ and } \text{pk} = (N, e)$$

Sign(sk, $m$): Output signature $\sigma = m^d \mod N$

Verify(pk, $m, \sigma$): Check if $\sigma^e = m \mod N$

**Problem**: Existentially forgeable!

# “Vanilla” RSA Signatures

Sign(sk, $m$): Output signature $\sigma = m^d \mod N$

Verify(pk, $m$, $\sigma$): Check if $\sigma^e = m \mod N$

**Problem**: Existentially forgeable!

*Attack:* Pick a random $\sigma$ and output ($m = \sigma^e$, $\sigma$) as the forgery.

**Problem**: Malleable!

*Attack:* Given a signature of $m$, you can produce a signature of $2^e * m, 3^e * m, \ldots, m^2, m^3, \ldots$

# “Vanilla” RSA Signatures

Sign(sk, $m$): Output signature $\sigma = m^d \mod N$

Verify(pk, $m$, $\sigma$): Check if $\sigma^e = m \mod N$

**Fundamental Issues**:

1. Can ”reverse-engineer” the message starting from the signature (Attack 1)

2. Algebraic structure allows malleability (Attack 2)

# How to Fix Vanilla RSA

Start with any trapdoor permutation, e.g. RSA.

Gen($1^\lambda$): Pick primes $(p, q)$ and let $N = pq$. Pick $e$ relatively prime to $\varphi(N)$ and let $d = e^{-1} \pmod{\varphi(N)}$.

$$\text{sk} = (p, q, d) \text{ and } \text{pk} = (N, e, H)$$

Sign(sk, $m$): Output signature $\sigma = H(m)^d \mod N$

Verify(pk, $m, \sigma$): Check if $\sigma^e = H(m) \mod N$

**So, what is H? Some very complicated "hash" function.**

# How to Fix Vanilla RSA

Start with any trapdoor permutation, e.g. RSA.

Gen($1^\lambda$): Pick primes $(p, q)$ and let $N = pq$. Pick $e$ relatively prime to $\varphi(N)$ and let $d = e^{-1} \pmod{\varphi(N)}$.

$$\text{sk} = (p, q, d) \text{ and } \text{pk} = (N, e, H)$$

Sign(sk, $m$): Output signature $\sigma = H(m)^d \mod N$

Verify(pk, $m, \sigma$): Check if $\sigma^e = H(m) \mod N$

**H should be at least one-way to prevent Attack #1.**

# How to Fix Vanilla RSA

Start with any trapdoor permutation, e.g. RSA.

Gen($1^\lambda$): Pick primes $(p, q)$ and let $N = pq$. Pick $e$ relatively prime to $\varphi(N)$ and let $d = e^{-1} \pmod{\varphi(N)}$.

$$\text{sk} = (p, q, d) \text{ and } \text{pk} = (N, e, H)$$

Sign(sk, $m$): Output signature $\sigma = H(m)^d \mod N$

Verify(pk, $m, \sigma$): Check if $\sigma^e = H(m) \mod N$

**Hard to "algebraically manipulate" H(m) into H(related m').**
**(to prevent Attack #2.)**

# How to Fix Vanilla RSA

Start with any trapdoor permutation, e.g. RSA.

Gen($1^\lambda$): Pick primes $(p, q)$ and let $N = pq$. Pick $e$ relatively prime to $\varphi(N)$ and let $d = e^{-1} \pmod{\varphi(N)}$.

$$\text{sk} = (p, q, d) \text{ and } \text{pk} = (N, e, H)$$

Sign(sk, $m$): Output signature $\sigma = H(m)^d \mod N$

Verify(pk, $m, \sigma$): Check if $\sigma^e = H(m) \mod N$

**Collision-resistance does not seem to be enough.**

(Given a CRHF $H(m)$, you may be able to produce $H(m')$ for related $m'$.)