Day 6: Digital Signatures

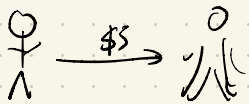
Today:

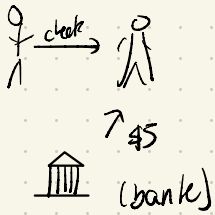
1. Three Problems
2. Digital Signatures
3. Schnorr Signatures
4. How HTTPS works

1 Three problems

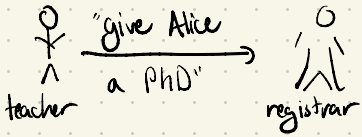
a Digital commerce

→ Alice $\xrightarrow{\$5}$ Bob traditionally ~~cash (uncopyable)~~, checks



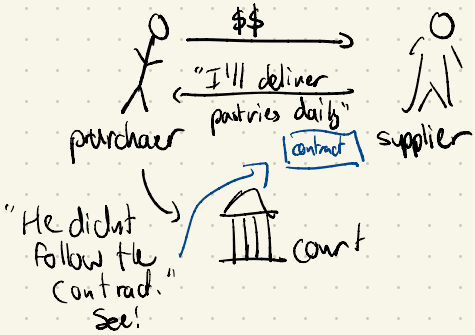


b Digital authorization



traditionally: we sign the order
→ assures the <u>recipient</u> of authorization

c Digital contracts



traditionally: we sign the contract
→ assures <u>others</u> of commitment.

In all cases we need a <u>digital signature</u>
→ publicly verifiable
→ unforgeable (requires a secret key to make)

Observation: it must be <u>message dependent</u> because digital data can always be copied.

2 Digital Signatures Definition & Security

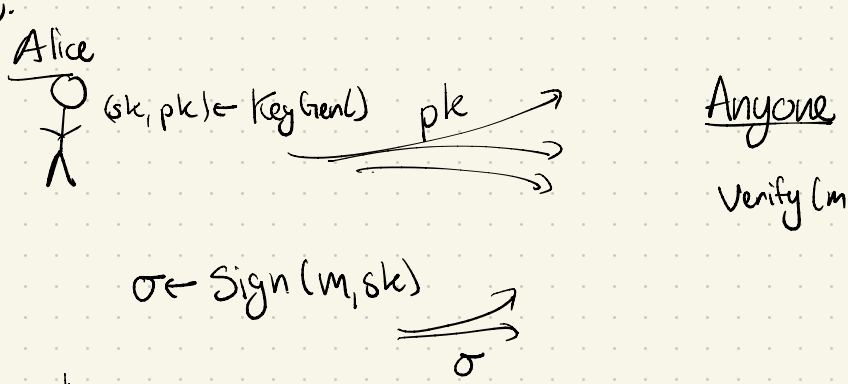
3 algorithms:

$\text{KeyGen}() \to (sk, pk)$

$\text{Sign}(m, sk) \to \sigma$ ← greek letter 'sigma'

$\text{Verify}(m, pk, \sigma) \to \text{True/False}$

Use:



Properties

Correctness: for all m,

$$\Pr[(sk, pk) \leftarrow \text{KeyGen}() ; \text{Verify}(m, pk, \text{Sign}(m, sk)) = \text{True}] = 1.$$

2b Unforgeability:

It should be hard to create a valid σ for a <u>new</u> message (unless you know sk.

Formalized as an "attack game" or "security game" that an adversary A plays:

A	Game Rules
	$(sk, pk) \leftarrow KeyGen()$
$\xleftarrow{\quad pk \quad}$	
$\xrightarrow{\quad m_i \quad}$	$\sigma_i \leftarrow Sign(m_i, sk)$
$\xleftarrow{\quad \sigma_i \quad}$	
$\downarrow m, \sigma$	

repeat as many times as A wants.

A wins if $\underbrace{Verify(m, pk, \sigma) = True}_{\sigma \text{ is valid}}$ and $\underbrace{m \notin \{m_1, \dots, m_Q\}}_{m \text{ is new}}$

For a signature scheme to be secure, for all efficient A, the probability that A wins must be negligible in λ.

Q: Why do we give A signatures on 'other' messages?

A: in the real world, an honest signer may sign many messages before the adversary tries to forge a signature. Which messages? It's safe to let the adversary choose: If we're secure against that then we're secure against anything.

Example where the client signs an potentially adversarial message:

Evil Server		Alice
	I'm Alice! $\rangle$	
	$\langle$ Really? Then sign BYTES	
	Okay! σ, now, let me in!	

← Signing arbitrary data is good for authentication (but allows an Adversary to get many σ's).

3 Schnorr Signatures

KeyGen(): (just DH key generation)

$sk \leftarrow \text{random}(\mathbb{Z}_q)$

$pk \leftarrow G^{sk}$

output (sk, pk)

Sign$(m, sk) \rightarrow \sigma$:

$r \leftarrow \text{random}(\mathbb{Z}_q)$

$R \leftarrow G^r$

$pk \leftarrow G^{sk}$

as bytes

x: int in $\mathbb{Z}_q \leftarrow H(m \,\|\, pk \,\|\, R)$

z: int in $\mathbb{Z}_q \leftarrow sk \times x + r$

output $\sigma \leftarrow (z, R)$

Verify(m, pk, σ):

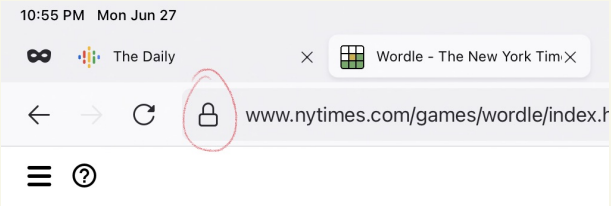
$(z, R) \leftarrow \sigma$

as bytes

$x \leftarrow H(m \,\|\, pk \,\|\, R)$

output whether $pk^x \cdot R = G^z$

4 How HTTPS works (approximately)



Warning: The scheme described IS NOT fully secure! This shouldn't be used. Educational purposes only...

HTTPS: HTTP run over TLS (Transport Layer Security)

TLS:

→ Transport Layer: A way for two computers to send bytes

→ Security: Those messages are <u>private</u> and <u>authentic</u>

TLS phases:

1. Session Establishment
   ↓ symmetric key(s)
2. Communication

Session Establishment

1. Do a key exchange
   - Gives a shared key k AND the public key pk of the entity that you exchanged with?
   - How do we know that this pk really is the one for our intended partner (nytimes.com?) We don't know that! Not yet!
2. Certificate Verification
   - Server sends a certificate:
     - A message: "pk is the public key for nytimes.com"

- also a signature σ on that message
  - created by a certificate authority

    globally trusted entities their pk's are included in your OS or browser.
- You check the certificate's σ
- You check that the pk's match.

Communication:

- Use a symmetric cipher with an authentication system (like a MAC).

Diagram