b^k ? k

Digital Signatures

What do we expect from a “signature?”

To certify, to a recipient of a document, that a concrete entity has produced the document and/or agrees to be bound by the contents of a document.

What do we expect from a “signature?”

To certify, to a recipient of a document, that a concrete entity has produced the document and/or agrees to be bound by the contents of a document.

Couldn’t this be done by a MAC? Yes, but we expect more from a “signature”.

What do we expect from a “signature?”

~~To certify, to a recipient of a document, that a concrete entity has produced the document and/or agrees to be bound by the contents of a document.~~

Couldn’t this be done by a MAC? Yes, but we expect more from a “signature”.

> To certify, to a recipient of a document and any other 3rd party, at any time after producing the signature, that a concrete entity has produced the document and/or agrees to be bound by the contents of a document.

The additional property is called non-repudiation: the signatory cannot deny signing the document.

Digital signature scheme

Definition 1

A digital signature scheme over $(\mathcal{K}_p, \mathcal{K}_s, \mathcal{M}, \mathcal{S}g)$ is a triple $\mathcal{D}s = (Gen, S, V)$ where

- Gen is an input-less randomized algorithm which produces a pair of keys $(k_p, k_s) \in \mathcal{K}_p \times \mathcal{K}_s$. Possible outputs of Gen are called valid keys for $\mathcal{D}s$, and we denote by $\mathcal{V}(Gen) \subseteq \mathcal{K}_p \times \mathcal{K}_s$ the set of all $\mathcal{D}s$'s valid keys;
- $S\colon \mathcal{K}_s \times \mathcal{M} \to \mathcal{S}g$ is a (possibly randomized) signing algorithm s.t.
- $V\colon \mathcal{K}_p \times \mathcal{M} \times \mathcal{S}g \to \{true, false\}$ is a deterministic verification algorithm such that

$$\forall m \in \mathcal{M}\ \forall (k_p, k_s) \in \mathcal{V}(\mathcal{D}s): \quad V(k_p, m, S(k_s, m)) = true \quad \text{with probability 1.}$$

Secure digital signatures = resistant against forgery.

Levels of security of digital signatures

The security level depends on forger's intent:

- key recovery
- selective forgery
- existential forgery

... and forger's capabilities:

- passive adversary
- chosen message attacks
- adaptive chosen-message attacks (correspond to CPA security of ciphers)

Existential forgery attack game for digital signatures

Let $\mathcal{D}s = (Gen, S, V)$ be a dig. signature scheme over $(\mathcal{K}_p, \mathcal{K}_s, \mathcal{M}, \mathcal{S}g)$. An existential forgery attack game between the challenger and the adversary $\mathcal{A}$ proceeds as follows:

- The challenger generates a pair (k_p, k_s) using Gen. The public key k_p is revealed to the adversary while k_s is kept secret.
- The adversary selects a number of rounds N for which the game will be played.
- In each round i:
  - The adversary computes a message $m_i \in \mathcal{M}$ and sends it to the challenger.
  - The challenger computes $\sigma_i = S(k_s, m_i)$ and send σ_i to the adversary.

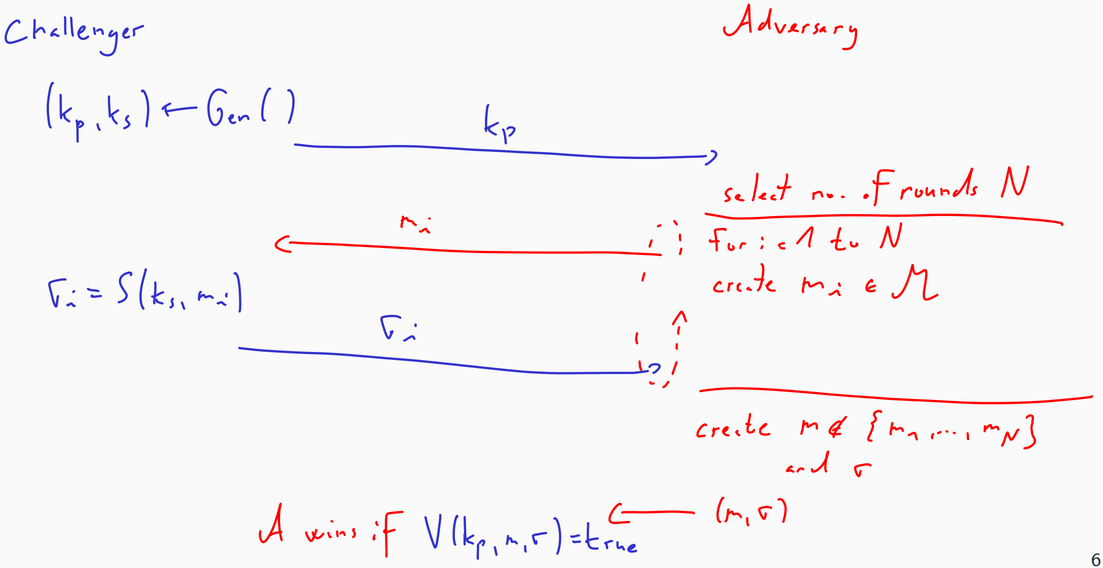
  After the final round, the adversary computes a tuple $(m, \sigma) \in \mathcal{M} \times \mathcal{S}g$ s.t. $m \notin \{m_1, \ldots, m_N\}$. The adversary wins the game if $V(k_p, m, \sigma) = true$.

The advantage of $\mathcal{A}$ against $\mathcal{D}s$ is the quantity

$$ADV_{Sig}(\mathcal{D}s, \mathcal{A}) = \mathbb{P}(\mathcal{A} \text{ wins the e.f. game}).$$

$\mathcal{D}s$ is ε-secure if $ADV_{Sig}(\mathcal{D}s, \mathcal{A}) \leq \varepsilon$ for every efficient adversary $\mathcal{A}$.

EF attack game for digital signatures: picture



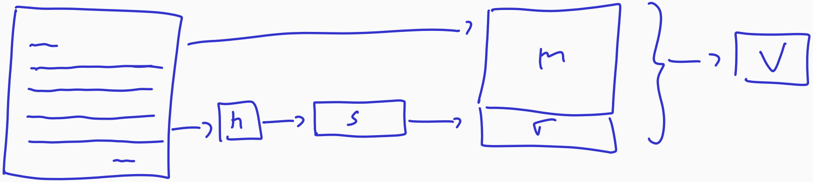
Role of hashing in digital signature schemes

Since asymmetric crypto primitives are less efficient than symmetric ones, it is preferable to apply the algorithms only on relatively short messages.

Role of hashing in digital signature schemes

Since asymmetric crypto primitives are less efficient than symmetric ones, it is preferable to apply the algorithms only on relatively short messages.

This is why practical signature schemes do not sign the original messages but hashes of these messages, computed by collision-resistant hash functions.



In some literature, this is omitted from the description of the algorithms, i.e. it is assumed that $\mathcal{M}$ is a hash space of some collision resistant hash functions.

One can show that this use of signatures is secure in the sense that if $\mathcal{Ds}$ is a d.s. scheme operating over some general message space $\mathcal{M}$, then a scheme $\mathcal{Ds}_h$ which applies a collision-resistant hash function h before signing and verifying the message is also secure.

Generic construction: Full domain hash

Recall that a trapdoor function $f : \mathcal{K}_p \times X \to Y$ is specified by two polynomial-time algorithms: *Fwd* and *Inv* satisfying:

- given $m \in X$ and $k_p \in \mathcal{K}_p$, the algorithm *Fwd* computes $f(k_p, m)$, and $= c$ m^e
- given $c \in Y$ and $k_s \in \mathcal{K}_s$, the algorithm *Inv* computes an element of X s.t. for all $m \in X$, and all valid key pairs (k_p, k_s) it holds

$$Inv(k_s, f(k_p, m)) = m.$$

c^d

Moreover, when given $c \in Y$, without the knowledge of k_s, it is hard to compute $m \in X$ s.t. $f(k_p, m) = c$.

Generic construction: Full domain hash

Recall that a trapdoor function $f : \mathcal{K}_p \times X \to Y$ is specified by two polynomial-time algorithms: *Fwd* and *Inv* satisfying:

- given $m \in X$ and $k_p \in \mathcal{K}_p$, the algorithm *Fwd* computes $f(k_p, m)$, and
- given $c \in Y$ and $k_s \in \mathcal{K}_s$, the algorithm *Inv* computes an element of X s.t. for all $m \in X$, and all valid key pairs (k_p, k_s) it holds

$$Inv(k_s, f(k_p, m)) = m.$$

Moreover, when given $c \in Y$, without the knowledge of k_s, it is hard to compute $m \in X$ s.t. $f(k_p, m) = c$.

Full domain hash: use the secret key operation $Inv(k_s, \cdot)$ for signing and the public-key operation $Fwd(k_p, \cdot)$ for verification.

$h(m)^d = \sigma$ $\qquad$ σ^e

Full domain hash: pseudocode

When using a trapdoor permutation to construct a d.s. scheme (Gen, S, V), the key generation algorithm is the same as for the corresponding encryption scheme, while signing and verification are performed as follows:

Algorithm 1: Signing using the secret-key operation $Inv(k_s, \cdot)$ and hash function h

---

Input: Secret key $k_s \in \mathcal{K}_s$, message $m \in \mathcal{M}$

Output: $S(k_s, m)$

return $Inv(k_s, h(m))$

$(\underline{h(m)})^d$

Algorithm 2: Verification using the public-key operation $Fwd(k_p, \cdot)$ and hash function h

---

Input: Public key $k_p \in \mathcal{K}_p$, message $m \in \mathcal{M}$, signature $\sigma \in \mathcal{S}g$

Output: $V(k_p, m, \sigma)$

if $Fwd(k_p, \sigma) = h(m)$ **then return** *true*;

else return *false*;

σ^e

$(h(m)^d)^e = h(m)^{d \cdot e}$

$\overset{?}{=}$ $h(m)$

Full domain hash: pseudocode

When using a trapdoor permutation to construct a d.s. scheme (Gen, S, V), the key generation algorithm is the same as for the corresponding encryption scheme, while signing and verification are performed as follows:

Algorithm 1: Signing using the secret-key operation $Inv(k_s, \cdot)$ and hash function h

Input: Secret key $k_s \in \mathcal{K}_s$, message $m \in \mathcal{M}$
Output: $S(k_s, m)$
return $Inv(k_s, h(m))$

Algorithm 2: Verification using the public-key operation $Fwd(k_p, \cdot)$ and hash function h

Input: Public key $k_p \in \mathcal{K}_p$, message $m \in \mathcal{M}$, signature $\sigma \in \mathcal{S}g$
Output: $V(k_p, m, \sigma)$
if $Fwd(k_p, \sigma) = h(m)$ **then return** *true*;
else return *false*;

The use of hash function is essential for security here!

Hash is essential in FDH

Adv: $m \in \mathbb{Z}_n^\times$ Ch

$m_0 \longrightarrow$

$\sigma: m_0^d \longleftarrow m^d$

$m_1 \longrightarrow$

$m_1^d \longleftarrow$

$(m_0 \cdot m_1,\ m_0^d \cdot m_1^d)$

$(m_0^2,\ \sigma^2)$

$\boxed{(h(m)^d)^2}$

$h(m_1)^d \quad h(m)^d$

$\neq (h(m^2))^d$

$(\sigma^2)^e = \sigma^{2 \cdot e} = (m_0^d)^{2e} = m_0^{2 \cdot e \cdot d} = m_0^2$

$m^x = m^{x \pmod{\varphi(N)}}$

$e \cdot d = 1 \pmod{\varphi(N)}$

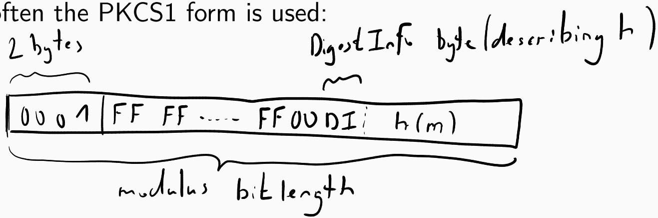
Digital signatures from trapdoor functions: security theorem (informal)

Theorem 1

Let f be a trapdoor function and h a collision-resistant hash function. Then the full-domain hash derived from f and h is secure.

RSA signature scheme

- Instantiates the generic construction with $Fwd(k_p, m) = m^e \pmod N$ and $Inv(k_s, c) = c^d \pmod N$.
- First widely adopted digital signature scheme.
- In practice, often the PKCS1 form is used:



However, if implemented badly, it is susceptible to a variant of the Bleichenbacher's attack.

DL-based digital signatures: Schnorr's signature

- Developed from Schnorr's authentication protocol via Fiat-Shamir trick (see next lecture).
- Strong security guarantees and efficiency. Not widely adopted due to patent protection at time of standardization.
- Depends on the use of ephemeral key which must be unique and randomly chosen for each signed message.
- Can be formulated over an arbitrary group. In the following, we have a group G of order p and some generator b of a cyclic sub-group of prime order q. I.e., the secret key is $k_s = (G, b, q, \underline{k})$ and the public key is $k_p = (G, b, q, \underline{b^k})$ for some $k \in \{0, \ldots, q-1\}$.

$b^0, b^1, \ldots, b^{\mathrm{ord}(b)-1}$ $\quad q = \mathrm{ord}(b)$

Schnorr's signature

Algorithm 3: Signing in Schnorr's signature with a hash function h mapping inputs into $\mathbb{Z}_q^\times$

Input: Secret key $k_s = (G, b, q, k)$, message $m \in \mathcal{M}$

Output: $S_{Schnorr}(k_s, m)$

$r \leftarrow$ sample randomly from $\mathbb{Z}_q^\times$;

$z \leftarrow k \cdot h(m \,||\, b^r) + r \pmod q$;

return (b^r, z)

m_1, m_2

$(b^r, \underbrace{k \cdot h(m_1 || b^r) + r}_{z_1}) \cdot G_1$

$(b^r, \underbrace{k \cdot h(m_2 || b^r) + r}_{z_2}) \cdot G_2$

$b^0, b^1, \ldots, b^{q-1}$

$z = 0$

$z = 1000$

$\rho = \underbrace{b^z}_{1} \cdot \underbrace{(b^k)^{-h(m||\rho)}}_{\rho = \_}$

Algorithm 4: Verification in Schnorr's signature

Input: Pub. key $k_p = (G, b, q, \kappa = b^k)$, message $m \in \mathcal{M}$, signature $(\rho, z) \in \mathcal{S}g$

Output: $V_{Schnorr}(k_p, m, (\rho, z))$

$v \leftarrow b^z \cdot \kappa^{-h(m \,||\, \rho)}$;

if $v = \rho$ **then return** *true*;

else return *false*;

$\rho = b^z \cdot (b^k)^{-h(m||\rho)}$

$(b^r, k \cdot h(m || b^r) + r)$

$\rightarrow z_1 - z_2 = k \cdot \underbrace{\left(h(m_1 || b^r) - h(m_2 || b^r)\right)}_{x}$

$b^r = b^{k \cdot h(m||b^r) + r} \cdot b^{-h(m||b^r) \cdot k} = b^r$

x^{-1}

$(z_1 - z_2) \cdot x^{-1} = k$

DSS and DSA

- Digital signature standard (DSS) adopted by NIST as a US federal standard in 1994.
- Further revisions published subsequently, the latest in 2013.
- Contains a specification of Digital signature algorithm (DSA), a digital signature scheme based on the discrete logarithm problem in $\mathbb{Z}_p^\times$.

DSA

DSA key generation (high-level) for M-bit modulus length (recommended 3072) and L-bit signature length (recommended 256):

- Randomly select an L-bit prime q.
- Randomly select an M-bit prime p s.t. $q \mid p-1$ (DSS recommends concrete algorithms for this selection).
- Compute a generator g of $\mathbb{Z}_p^\times$.
- Compute a generator b of a sub-group of $\mathbb{Z}_p^\times$ of order q by putting $b = g^{\frac{p-1}{q}}$. The element b will serve as the base of the discrete logarithm in the scheme.
- Select a random number $k \in \{1, \ldots, q-1\}$.
- Put $\kappa = b^k \pmod{p}$.

$b \in \mathbb{Z}_p^\times$

$\underline{\underline{\mathbb{Z}_p^\times}}$ $\quad |\langle b \rangle| = q$

The public key is $(\underline{p, q, b}, \kappa)$, the secret key is $(\underline{p, q, b}, k)$.

$\kappa = b^k \pmod{p}$

$1 \le k \le q-1$

DSA (pseudocode)

SHA-256 is used as h. If hash size $< L$, only leftmost L bits of the hash are taken.

Algorithm 5: Signing in DSA

Input: Secret key $k_s = (p, q, b, k)$, message $m \in \mathcal{M}$

Output: $S_{\mathrm{DSA}}(k_s, m)$

$r \leftarrow$ sample randomly from $\{1, \ldots, q-1\}$;

$\rho \leftarrow (b^r \pmod p) \pmod q$;

$z \leftarrow r^{-1} \cdot (h(m) + k \cdot \rho) \pmod q$;

return (ρ, z)

$z^{-1} = r \cdot \cancel{(h(m) + k \cdot \rho)^{-1}}$

$b^r = b^{h(m) \cdot z^{-1}} \cdot b^{k \cdot b^r \cdot z^{-1}}$

$= b^{z^{-1} \cdot \cancel{(h(m) + k \cdot b^r)}}$

$= b^r$

Algorithm 6: Verification in DSA

Input: Pub. key $k_p = (p, q, b, \kappa = b^k \pmod p)$, message $m \in \mathcal{M}$, signature $(\rho, z) \in \mathcal{S}g$

Output: $V(k_p, m, (\rho, z))$

compute $z^{-1} \pmod q$ and $\mu = b^{h(m)} \pmod p$;

$v \leftarrow (\mu^{z^{-1}} \cdot \kappa^{\rho \cdot z^{-1}} \pmod p) \pmod q$;

if $\rho = v$ **then return** *true*;

else return *false*;

ECDSA

- A variant of DSA with elliptic curves. Also standardized by NIST.
- Works exactly as DSA, but with operations $\pmod{p}$ replaced by the operations in the EC group.
- The only difference: how to get from a member $x = b^r$ of the group G an exponent: a number ρ in $\{1, \ldots, |G|\}$:
  - In DSA, x was an integer, so we just performed $\rho \leftarrow x \pmod{q}$.
  - In ECDSA, x is a point (x_1, x_2) where the coordinates are integers modulo some prime. We put $\rho \leftarrow x_1 \pmod{|G|}$. If, by coincidence, $\rho = 0$, we need to sample a new x.
- In both DSA an ECDSA (and in Schnorr), it is important that r is indeed random, unpredictable, and not re-used (guaranteed with high probability if random): otherwise, the signature scheme is completely insecure (e.g. the PlayStation 3 exploit).