Cryptologie asymétrique 2/2

Damien Vergnaud

Sorbonne Université

CRYPTO 1

Contents

1. Digital signatures
   - Security Notions for Digital Signatures
   - Construction from a Trapdoor Permutation
2. One-time signatures
   - Lamport signatures
   - Generalizations
3. Schnorr signatures

Digital Signatures

- A very important public key primitive is the **digital signature**.
- The idea is
  - Message + Alice's Private Key = Signature
  - Message + Signature + Alice's Public Key = YES/NO
- Alice can sign a message using her private key.
- Anyone can verify Alice's signature, since everyone can obtain her public key.
- the verifier is convinced that only Alice could have produced the signature
  - **only Alice knows her private key!**

Digital Signatures

- A very important public key primitive is the **digital signature**.
- The idea is
  - Message + Alice's Private Key = Signature
  - Message + Signature + Alice's Public Key = YES/NO
- Alice can sign a message using her private key.
- Anyone can verify Alice's signature, since everyone can obtain her public key.
- the verifier is convinced that only Alice could have produced the signature
  - **only Alice knows her private key!**

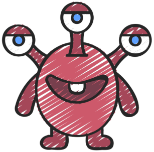
Digital Signatures

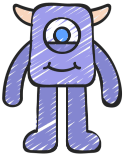
- A very important public key primitive is the **digital signature**.
- The idea is
  - Message + Alice's Private Key = Signature
  - Message + Signature + Alice's Public Key = YES/NO
- Alice can sign a message using her private key.
- Anyone can verify Alice's signature, since everyone can obtain her public key.
- the verifier is convinced that only Alice could have produced the signature
  - **only Alice knows her private key!**

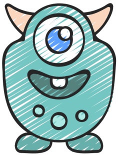
Digital Signatures

- A very important public key primitive is the **digital signature**.
- The idea is
  - Message + Alice's Private Key = Signature
  - Message + Signature + Alice's Public Key = YES/NO
- Alice can sign a message using her private key.
- Anyone can verify Alice's signature, since everyone can obtain her public key.
- the verifier is convinced that only Alice could have produced the signature
  - **only Alice knows her private key!**

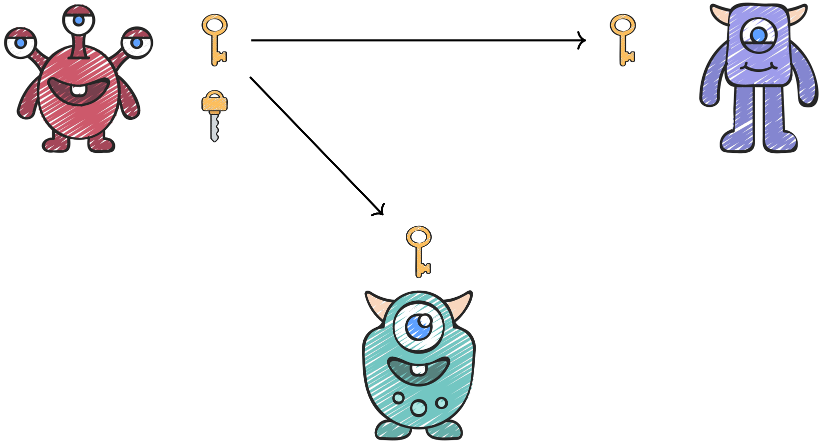
Digital signature schemes



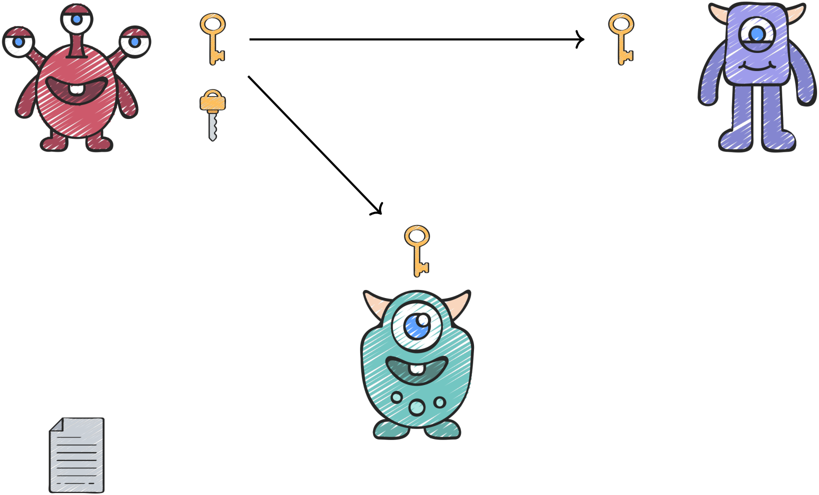




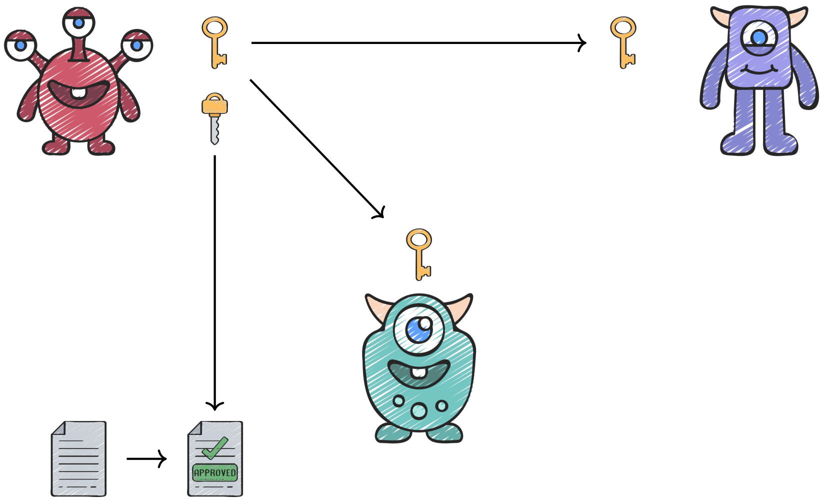
Digital signature schemes



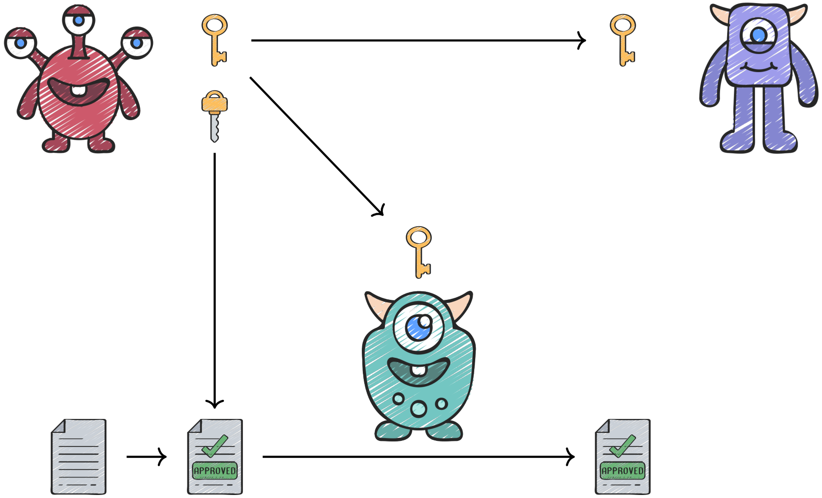
Digital signature schemes



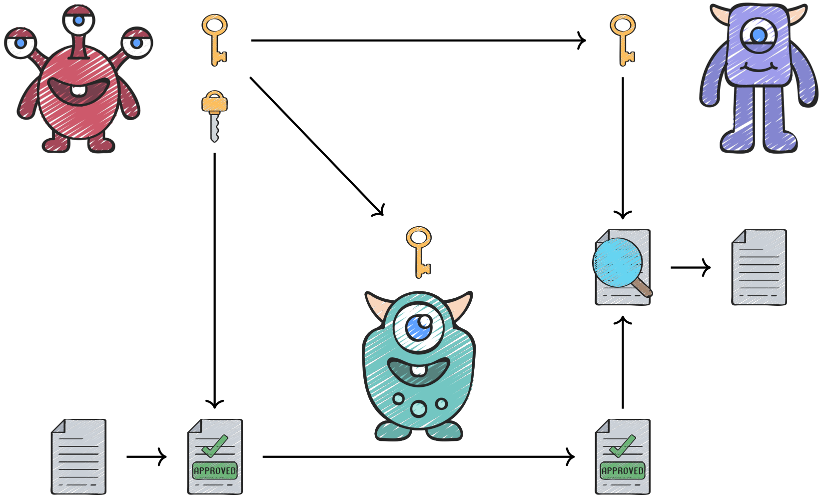
Digital signature schemes



Digital signature schemes



Digital signature schemes



Digital Signatures : Services

- The verification algorithm is used to determine whether or not the signature is properly constructed.
- the verifier has guarantee of
  - message integrity and
  - message origin.
- also provide non-repudiation - not provided by MACs.

Most important cryptographic primitive!

Security Notions

Depending on the context in which a given cryptosystem is used, one may formally define a security notion for this system,

- by telling what **goal** an adversary would attempt to reach,
- and what means or information are made available to her (the **attack model**).

A security notion (or level) is entirely defined by pairing an adversarial goal with an adversarial model.

Examples: UB-KMA, UUF-KOA, EUF-SOCMA, EUF-CMA.

Signature Schemes

- Signer Alice generates a public/private key pair (pk, sk) by running a probabilistic **key generation algorithm** $G(k)$, k being the security parameter. Alice publishes pk.
- Whenever Alice wishes to sign a digital document $m \in \{0,1\}^*$, she computes the signature $s = S(sk, m)$ where S is the (possibly probabilistic) **signing algorithm**. She outputs s and maybe also m.
- Knowing m and s (and Alice's public key pk), Bob can verify that s is a signature of m output by Alice by running the **verification algorithm** $V(pk, m, s)$ returning 1 if $s = S(sk, m)$ or 0 otherwise.

The signature scheme is the triple (G, S, V) and their domains.

Security Goals

[Unbreakability] the attacker recovers the secret key *sk* from the public key *pk* (or an equivalent key if any). This goal is denoted **UB**. Implicitly appeared with public-key cryptography.

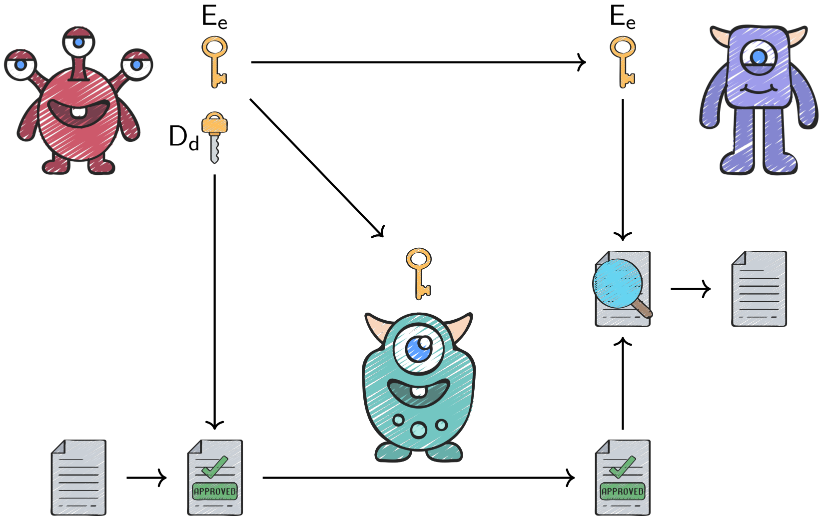
[Universal Unforgeability] the attacker, without necessarily having recovered *sk*, can produce a valid signature of any message in the message space. Noted **UUF**.

[Existential Unforgeability] the attacker creates a message and a valid signature of it (likely not of his choosing). Denoted **EUF**.

Adversarial Models

- **Key-Only Attacks** (KOA), unavoidable scenario.
- **Known Message Attacks** (KMA) where an adversary has access to signatures for a set of known messages.
- **Chosen-Message Attacks** (CMA) the adversary is allowed to use the signer as an oracle (full access), and may request the signature of any message of his choice

Digital signature from a Trapdoor Permutation ...



… Isn't Fully Secure On Its Own

Remark. Assume Eve picks some random σ and computes $m = E_e(\sigma)$. Then (m, σ) is a valid pair since $\sigma = D_d(m)$ is a valid signature of m.

> *Eve can generate signatures for messages he doesn't control. This capability is known as existential forgery.*

- weak form of forgery
- **What if stronger attacks exist ?**

RSA - Key Generation

Rivest, Shamir, Adleman (1978)
A method for obtaining digital signatures and public key cryptosystems.
Communications of the ACM 21 (2): pp.120-126.

- **Key generation:**
  - Generate two large primes p and q ($p \neq q$).
  - Compute $N = p \cdot q$ and $\varphi(N) = (p-1)(q-1)$.
  - Select a random integer e, $1 < e < \varphi(N)$, such that $\gcd(e, (p-1)(q-1)) = 1$.
  - Compute the unique integer d, $1 < d < \varphi(N)$ with $e \cdot d \equiv 1(\mod \varphi(N))$.

  Public key $= (N, e)$ which can be published.
  Private key $= (d, p, q)$ which needs to be kept secret

RSA - Signature / Verification

- **Signature:** if Alice wants to sign a message, she does the following:
  - Represent the message as a number $0 < m < N$.
  - Use her private key d to compute $s = m^d \mod N$.
  - Send the signature s to Bob.

- **Verification:** to check the validity of s on m, Bob does the following:
  - Obtain Alice's authentic public key (N, e).
  - Check whether $m = s^e \mod N$.

RSA - Signature / Verification

- **Signature:** if Alice wants to sign a message, she does the following:
  - Represent the message as a number $0 < m < N$.
  - Use her private key d to compute $s = m^d \mod N$.
  - Send the signature s to Bob.

- **Verification:** to check the validity of s on m, Bob does the following:
  - Obtain Alice's authentic public key (N, e).
  - Check whether $m = s^e \mod N$.

Universal Forgery of RSA Signatures

RSA is a morphism: for $m_1, m_2 \in \mathbb{Z}_N^*$,

$$(m_1 m_2)^d = m_1^d m_2^d \pmod{N} ,$$

meaning that $S(m_1 m_2) = S(m_1)S(m_2)$.

Attack: now suppose Eve wants Alice's signature for some specific message $m =$"I owe Eve 10,000 euros".

1. she picks a random $m_1 \in \mathbb{Z}_N^*$ and computes $m_2 = m/m_1 \mod N$,
2. assume Eve asks Alice to sign m_1 and m_2 and receives $S(m_1)$ and $S(m_2)$,
3. Eve computes $S(m) = S(m_1)S(m_2) \mod N$ on her own.

Universal Forgery of RSA Signatures

RSA is a morphism: for $m_1, m_2 \in \mathbb{Z}_N^*$,

$$(m_1 m_2)^d = m_1^d m_2^d \pmod{N} ,$$

meaning that $S(m_1 m_2) = S(m_1)S(m_2)$.

Attack: now suppose Eve wants Alice's signature for some specific message $m =$"I owe Eve 10,000 euros".

1. she picks a random $m_1 \in \mathbb{Z}_N^*$ and computes $m_2 = m/m_1 \mod N$,
2. assume Eve asks Alice to sign m_1 and m_2 and receives $S(m_1)$ and $S(m_2)$,
3. Eve computes $S(m) = S(m_1)S(m_2) \mod N$ on her own.

Universal Forgery of RSA Signatures

This is a universal forgery under a chosen-message attack.

- much worse than existential forgery
- **but**, this attack assumes that Eve has access to Alice's signing operation

The Need for Hashing

Instead of signing the message m directly, let's apply a hash function H to it:

- Alice generates and publishes some trapdoor permutation E_e,
- she keeps D_d private,
- to sign m, Alice computes $s = D_d(H(m))$ and sends the pair (m, s) to Bob,
- to verify the signature, Bob checks whether $H(m) = E_e(s)$.

$\rightsquigarrow$ Hash-then-Invert paradigm. H is now a part of the scheme.

It must map messages to elements of E's domain, say X.

The Need for Hashing (Cont'd)

What have we done? Well, if H maps $\{0,1\}^*$ to X, then arbitrarily long messages can now be signed. Better.

What about existential forgery? Assume Eve picks some random σ and computes $\mu = E(\sigma)$, she faces the problem of finding an m such that $H(m) = \mu$.

The hope is that with a "good choice" for H, Eve cannot do that (in particular H must be one-way). Hopefully better.

What about universal forgery? Getting back to the multiplicative attack for E =RSA, the attacker has to find m_1, m_2 such that

$$H(m_1)H(m_2) = H(m) \pmod{N}.$$

Again, with a "good choice" for H, these should be difficult to find (H must somehow destroy the algebraic structure of $\mathbb{Z}_n^*$). Hopefully better.

The Need for Hashing (Cont'd)

What have we done? Well, if H maps $\{0,1\}^*$ to X, then arbitrarily long messages can now be signed. Better.

What about existential forgery? Assume Eve picks some random σ and computes $\mu = E(\sigma)$, she faces the problem of finding an m such that $H(m) = \mu$.

The hope is that with a "good choice" for H, Eve cannot do that (in particular H must be one-way). Hopefully better.

What about universal forgery? Getting back to the multiplicative attack for $E =$RSA, the attacker has to find m_1, m_2 such that

$$H(m_1)H(m_2) = H(m) \pmod{N}.$$

Again, with a "good choice" for H, these should be difficult to find (H must somehow destroy the algebraic structure of $\mathbb{Z}_n^*$). Hopefully better.

The Need for Hashing (Cont'd)

What have we done? Well, if H maps $\{0,1\}^*$ to X, then arbitrarily long messages can now be signed. Better.

What about existential forgery? Assume Eve picks some random σ and computes $\mu = E(\sigma)$, she faces the problem of finding an m such that $H(m) = \mu$.

The hope is that with a "good choice" for H, Eve cannot do that (in particular H must be one-way). Hopefully better.

What about universal forgery? Getting back to the multiplicative attack for E =RSA, the attacker has to find m_1, m_2 such that

$$H(m_1)H(m_2) = H(m) \pmod{N} .$$

Again, with a "good choice" for H, these should be difficult to find (H must somehow destroy the algebraic structure of $\mathbb{Z}_n^*$). Hopefully better.

The Need for Hashing (Cont'd)

Is there a drawback? Well, yes but moderate.

The use of H introduces a new type of attacks based on finding collisions. If the attacker finds m_1, m_2 such that $H(m_1) = H(m_2)$ then a signature of m_1 is also a signature of m_2.

$\rightsquigarrow$ existential forgery under a chosen-message attack: Eve queries Alice on m_1 to get s and then outputs (m_2, s) as a valid signature.

Here too, we hope that H is chosen in a way that makes collisions hard to find

The Need for Hashing (Cont'd)

Is there a drawback? Well, yes but moderate.

The use of H introduces a new type of attacks based on finding collisions. If the attacker finds m_1, m_2 such that $H(m_1) = H(m_2)$ then a signature of m_1 is also a signature of m_2.

$\rightsquigarrow$ existential forgery under a chosen-message attack: Eve queries Alice on m_1 to get s and then outputs (m_2, s) as a valid signature.

Here too, we hope that H is chosen in a way that makes collisions hard to find

The Need for Hashing (Cont'd)

Is there a drawback? Well, yes but moderate.

The use of H introduces a new type of attacks based on finding collisions. If the attacker finds m_1, m_2 such that $H(m_1) = H(m_2)$ then a signature of m_1 is also a signature of m_2.

$\rightsquigarrow$ existential forgery under a chosen-message attack: Eve queries Alice on m_1 to get s and then outputs (m_2, s) as a valid signature.

Here too, we hope that H is chosen in a way that makes collisions hard to find

So, How Good is DH's Approach?

- We pinpointed features of H that are necessary for the Hash-then-Invert scheme to thwart certain attacks.
- **But** the true question should be: what features of H are **sufficient** to prevent **all** attacks?
- $\rightsquigarrow$ provable security, that is, the set of techniques by which one assesses the security level of a cryptosystem given assumptions on its ingredients.

Lamport signatures

L. Lamport
Constructing digital signatures from a one-way function
Technical Report SRI-CSL-98, SRI International Computer Science Laboratory, Oct. 1979.

- a Lamport signature or **Lamport one-time signature scheme** is a method for constructing efficient digital signatures.
- Lamport signatures can be built from any cryptographically secure **one-way** function; usually a **cryptographic hash function** is used.
- Unfortunately each Lamport key can only be used to sign a single message.
- However, we will see how a single key could be used for **many** messages, making this a fairly efficient digital signature scheme.

How to sign **one** bit **just once** ?

$$\boxed{\mathcal{M} = \{0,1\}}$$

- **Key generation:**
  - Generate $f : X \longrightarrow Y$ a **one-way function**.
  - Select two random elements $x_0, x_1 \in X$.
  - Compute their images $y_i = f(x_i)$ for $i \in \{0,1\}$.

  Public key $= (y_0, y_1)$ which can be published.
  Private key $= (x_0, x_1)$ which needs to be kept secret
- **Signature:** if Alice wants to sign a bit b, she does the following:
  - Use her private key (x_0, x_1) to send the signature $s = x_b$ to Bob.
- **Verification:** to check the validity of s on b, Bob does the following:
  - Obtain Alice's authentic public key (y_0, y_1).
  - Check whether $f(s) = y_b$.

How to sign **one** bit **just once** ?

$\boxed{\mathcal{M} = \{0, 1\}}$

- **Key generation:**
  - Generate $f : X \longrightarrow Y$ a **one-way function**.
  - Select two random elements $x_0, x_1 \in X$.
  - Compute their images $y_i = f(x_i)$ for $i \in \{0, 1\}$.

  Public key $= (y_0, y_1)$ which can be published.
  Private key $= (x_0, x_1)$ which needs to be kept secret

- **Signature:** if Alice wants to sign a bit b, she does the following:
  - Use her private key (x_0, x_1) to send the signature $s = x_b$ to Bob.
- **Verification:** to check the validity of s on b, Bob does the following:
  - Obtain Alice's authentic public key (y_0, y_1).
  - Check whether $f(s) = y_b$.

How to sign **one** bit **just once** ?

$$\boxed{\mathcal{M} = \{0,1\}}$$

- **Key generation:**
  - Generate $f : X \longrightarrow Y$ a **one-way function**.
  - Select two random elements $x_0, x_1 \in X$.
  - Compute their images $y_i = f(x_i)$ for $i \in \{0,1\}$.

  Public key $= (y_0, y_1)$ which can be published.
  Private key $= (x_0, x_1)$ which needs to be kept secret

- **Signature:** if Alice wants to sign a bit b, she does the following:
  - Use her private key (x_0, x_1) to send the signature $s = x_b$ to Bob.
- **Verification:** to check the validity of s on b, Bob does the following:
  - Obtain Alice's authentic public key (y_0, y_1).
  - Check whether $f(s) = y_b$.

How to sign k bits **just once** ?

$\boxed{\mathcal{M} = \{0,1\}^k}$

- **Key generation:**
  - Generate $f : X \longrightarrow$ a **one-way function**.
  - Select $2k$ random elements $x_{0,1}, x_{1,1}, \ldots, x_{0,k}, x_{1,k} \in X$.
  - Compute their images $y_{i,j} = f(x_{i,j})$ for $i \in \{0,1\}$ and $j \in [\![1,k]\!]$.

  Public key $= (y_{0,1}, y_{1,1}, \ldots, y_{0,k}, y_{1,k})$ which can be published.
  Private key $= (x_{0,1}, x_{1,1}, \ldots, x_{0,k}, x_{1,k})$ which needs to be kept secret
- **Signature:** if Alice wants to sign $m = m_1 \ldots m_k$, she does the following:
  - Use her private key $(x_{0,1}, x_{1,1}, \ldots, x_{0,k}, x_{1,k})$ to send the signature $s = (x_{m_1,1}, x_{m_1,2}, \ldots, x_{m_k,k})$ to Bob.
- **Verification:** to check the validity of $s = (s_1, \ldots, s_k)$ on m, Bob does the following:
  - Obtain Alice's authentic public key $(y_{0,1}, y_{1,1}, \ldots, y_{0,k}, y_{1,k})$.
  - Check whether $f(s_i) = y_{m_b,i}$ for all $i \in [\![1,k]\!]$.

How to sign k bits **just once** ?

$\boxed{\mathcal{M} = \{0,1\}^k}$

- **Key generation:**
  - Generate $f : X \longrightarrow$ a **one-way function**.
  - Select $2k$ random elements $x_{0,1}, x_{1,1}, \ldots, x_{0,k}, x_{1,k} \in X$.
  - Compute their images $y_{i,j} = f(x_{i,j})$ for $i \in \{0,1\}$ and $j \in [\![1,k]\!]$.

  Public key $= (y_{0,1}, y_{1,1}, \ldots, y_{0,k}, y_{1,k})$ which can be published.
  Private key $= (x_{0,1}, x_{1,1}, \ldots, x_{0,k}, x_{1,k})$ which needs to be kept secret
- **Signature:** if Alice wants to sign $m = m_1 \ldots m_k$, she does the following:
  - Use her private key $(x_{0,1}, x_{1,1}, \ldots, x_{0,k}, x_{1,k})$ to send the signature $s = (x_{m_1,1}, x_{m_1,2}, \ldots, x_{m_k,k})$ to Bob.
- **Verification:** to check the validity of $s = (s_1, \ldots, s_k)$ on m, Bob does the following:
  - Obtain Alice's authentic public key $(y_{0,1}, y_{1,1}, \ldots, y_{0,k}, y_{1,k})$.
  - Check whether $f(s_i) = y_{m_b,i}$ for all $i \in [\![1,k]\!]$.

How to sign k bits **just once** ?

$\boxed{\mathcal{M} = \{0,1\}^k}$

- **Key generation:**
  - Generate $f : X \longrightarrow$ a **one-way function**.
  - Select $2k$ random elements $x_{0,1}, x_{1,1}, \ldots, x_{0,k}, x_{1,k} \in X$.
  - Compute their images $y_{i,j} = f(x_{i,j})$ for $i \in \{0,1\}$ and $j \in [\![1,k]\!]$.

  Public key $= (y_{0,1}, y_{1,1}, \ldots, y_{0,k}, y_{1,k})$ which can be published.
  Private key $= (x_{0,1}, x_{1,1}, \ldots, x_{0,k}, x_{1,k})$ which needs to be kept secret
- **Signature:** if Alice wants to sign $m = m_1 \ldots m_k$, she does the following:
  - Use her private key $(x_{0,1}, x_{1,1}, \ldots, x_{0,k}, x_{1,k})$ to send the signature $s = (x_{m_1,1}, x_{m_1,2}, \ldots, x_{m_k,k})$ to Bob.
- **Verification:** to check the validity of $s = (s_1, \ldots, s_k)$ on m, Bob does the following:
  - Obtain Alice's authentic public key $(y_{0,1}, y_{1,1}, \ldots, y_{0,k}, y_{1,k})$.
  - Check whether $f(s_i) = y_{m_b,i}$ for all $i \in [\![1,k]\!]$.

How to sign k bits **just once** ?

- Lamport's scheme is EUF-CMA secure assuming **only** the one-wayness of f.
- The signature generation is very efficient.

- For a 128-bit security level, $Y = \{0,1\}^{128}$ and the public-key is made of $256 \cdot k$ bits and its generation requires 256 evaluations of the function f.
- The signature is made of k elements from X. If $X = \{0,1\}^{128}$ the signature length is $128 \cdot k$ bits.
- Can sign only one message

Lamport's signatures: variants

- **Short private key.** Instead of creating and storing all the random numbers of the private key a single key of sufficient size can be stored.

  The single key can then be used as the seed for a **cryptographically secure pseudorandom number generator** to create all the random numbers in the private key when needed.

- **Short public key** A Lamport signature can be combined with a hash list, making it possible to only publish a single hash instead of all the hashes in the public key.

- **Hashing the message.**
  - Unlike some other signature schemes the Lamport signature scheme **does not** require that the message m is hashed before it is signed.
  - A system for signing long messages can use a collision resistant hash function h and sign $h(m)$ instead of m.

Lamport's signatures: variants

- **Short private key.** Instead of creating and storing all the random numbers of the private key a single key of sufficient size can be stored.

  The single key can then be used as the seed for a **cryptographically secure pseudorandom number generator** to create all the random numbers in the private key when needed.

- **Short public key** A Lamport signature can be combined with a hash list, making it possible to only publish a single hash instead of all the hashes in the public key.

- **Hashing the message.**
  - Unlike some other signature schemes the Lamport signature scheme **does not** require that the message m is hashed before it is signed.
  - A system for signing long messages can use a collision resistant hash function h and sign $h(m)$ instead of m.

Lamport's signatures: variants

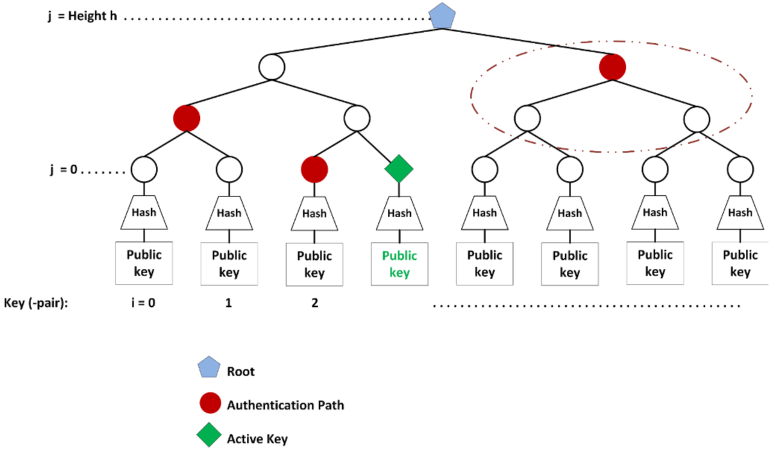
- **Short private key.** Instead of creating and storing all the random numbers of the private key a single key of sufficient size can be stored.

  The single key can then be used as the seed for a **cryptographically secure pseudorandom number generator** to create all the random numbers in the private key when needed.

- **Short public key** A Lamport signature can be combined with a hash list, making it possible to only publish a single hash instead of all the hashes in the public key.

- **Hashing the message.**
  - Unlike some other signature schemes the Lamport signature scheme **does not** require that the message m is hashed before it is signed.
  - A system for signing long messages can use a collision resistant hash function h and sign $h(m)$ instead of m.

Lamport's signatures: variants



- **Public key for multiple messages.**
  - many keys have to be published if many messages are to be signed.
  - a hash tree can be used on those public keys, publishing the top hash of the hash tree instead.

Textbook ElGamal signatures

ElGamal (1985)
A Public-Key Cryptosystem and a Signature Scheme based on Discrete Logarithms.
IEEE Transactions Information Theory, 31 pp. 469-472.

Key generation. $G(1^{\mathrm{k}})$ randomly selects a k-bit prime p and a generator g of $\mathbb{Z}_p^*$. The secret key is $x \leftarrow \mathbb{Z}_{p-1}$ and setting $y = g^x \mod p$, the public key is (p, g, y).

Signature. To sign a message $m \in \mathbb{Z}_{p-1}$, one generates (r, s) such that $g^m = y^r r^s \mod p$ as follows. Randomly select $k \leftarrow \mathbb{Z}_{p-1}^*$, set $r = g^k \mod p$ and $s = (m - xr)/k \mod p - 1$. Output (r, s).

Verification. Verify that $1 < r < p$ and $g^m = y^r r^s \mod p$.

Insecure! *cf* TD 5. *Hash the message first!*

Textbook ElGamal signatures

ElGamal (1985)
A Public-Key Cryptosystem and a Signature Scheme based on Discrete Logarithms.
IEEE Transactions Information Theory, 31 pp. 469-472.

Key generation. $G(1^{\mathrm{k}})$ randomly selects a k-bit prime p and a generator g of $\mathbb{Z}_p^*$. The secret key is $x \leftarrow \mathbb{Z}_{p-1}$ and setting $y = g^x \mod p$, the public key is (p, g, y).

Signature. To sign a message $m \in \mathbb{Z}_{p-1}$, one generates (r, s) such that $g^m = y^r r^s \mod p$ as follows. Randomly select $k \leftarrow \mathbb{Z}_{p-1}^*$, set $r = g^k \mod p$ and $s = (m - xr)/k \mod p - 1$. Output (r, s).

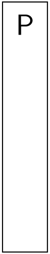
Verification. Verify that $1 < r < p$ and $g^m = y^r r^s \mod p$.

Insecure! *cf* TD 5. *Hash the message first!*

Schnorr's ID Protocol (91)

Let $\mathbb{G} = \langle g \rangle$ be a group of prime order q

Prover P proves to verifier V that he knows the discrete log x of a public group element $y = g^x$. It is a 3-move protocol.



Scenario

P sends $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$

V sends $c \xleftarrow{\$} \mathbb{Z}_q$

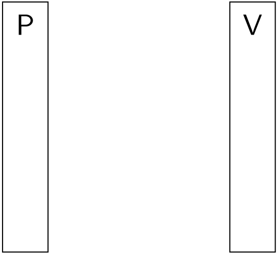
P sends $s = k + cx \bmod q$

V checks whether $g^s \cdot y^{-c} = r$

Schnorr's ID Protocol (91)

Let $\mathbb{G} = \langle g \rangle$ be a group of prime order q

Prover P proves to verifier V that he knows the discrete log x of a public group element $y = g^x$. It is a 3-move protocol.



Scenario

P sends $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$

V sends $c \xleftarrow{\$} \mathbb{Z}_q$

P sends $s = k + cx \bmod q$

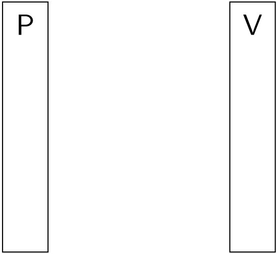
V checks whether $g^s \cdot y^{-c} = r$

Schnorr's ID Protocol (91)

Let $\mathbb{G} = \langle g \rangle$ be a group of prime order q

Prover P proves to verifier V that he knows the discrete log x of a public group element $y = g^x$. It is a 3-move protocol.

$x \xleftarrow{\$} \mathbb{Z}_q$



Scenario

P sends $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$

V sends $c \xleftarrow{\$} \mathbb{Z}_q$

P sends $s = k + cx \bmod q$

V checks whether $g^s \cdot y^{-c} = r$

Schnorr's ID Protocol (91)

Let $\mathbb{G} = \langle g \rangle$ be a group of prime order q

Prover P proves to verifier V that he knows the discrete log x of a public group element $y = g^x$. It is a 3-move protocol.

$x \xleftarrow{\$} \mathbb{Z}_q$

$y = g^x$

P

V

Scenario

P sends $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$

V sends $c \xleftarrow{\$} \mathbb{Z}_q$

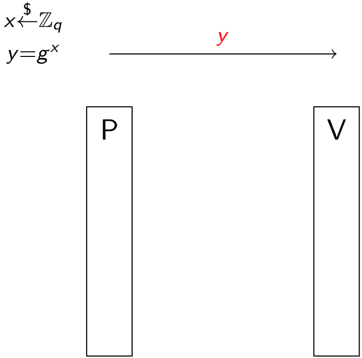
P sends $s = k + cx \bmod q$

V checks whether $g^s \cdot y^{-c} = r$

Schnorr's ID Protocol (91)

Let $\mathbb{G} = \langle g \rangle$ be a group of prime order q

Prover P proves to verifier V that he knows the discrete log x of a public group element $y = g^x$. It is a 3-move protocol.



Scenario

P sends $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$

V sends $c \xleftarrow{\$} \mathbb{Z}_q$

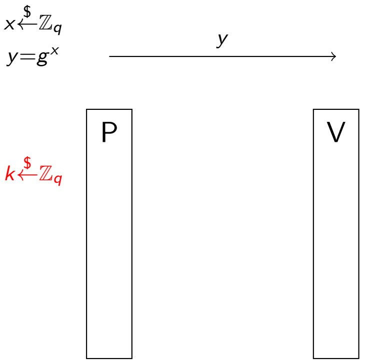
P sends $s = k + cx \bmod q$

V checks whether $g^s \cdot y^{-c} = r$

Schnorr's ID Protocol (91)

Let $\mathbb{G} = \langle g \rangle$ be a group of prime order q

Prover P proves to verifier V that he knows the discrete log x of a public group element $y = g^x$. It is a 3-move protocol.



Scenario

P sends $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$

V sends $c \xleftarrow{\$} \mathbb{Z}_q$

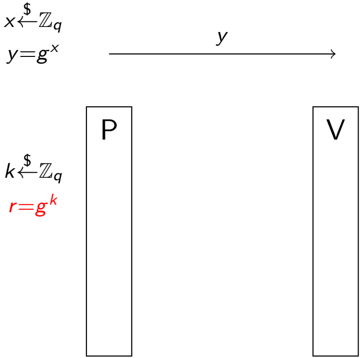
P sends $s = k + cx \bmod q$

V checks whether $g^s \cdot y^{-c} = r$

Schnorr's ID Protocol (91)

Let $\mathbb{G} = \langle g \rangle$ be a group of prime order q

Prover P proves to verifier V that he knows the discrete log x of a public group element $y = g^x$. It is a 3-move protocol.



Scenario

P sends $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$

V sends $c \xleftarrow{\$} \mathbb{Z}_q$

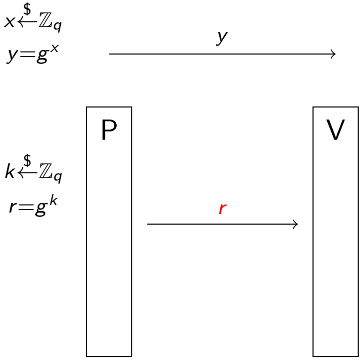
P sends $s = k + cx \bmod q$

V checks whether $g^s \cdot y^{-c} = r$

Schnorr's ID Protocol (91)

Let $\mathbb{G} = \langle g \rangle$ be a group of prime order q

Prover P proves to verifier V that he knows the discrete log x of a public group element $y = g^x$. It is a 3-move protocol.



Scenario

P sends $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$

V sends $c \xleftarrow{\$} \mathbb{Z}_q$

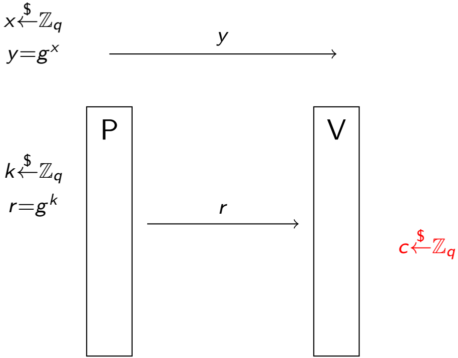
P sends $s = k + cx \bmod q$

V checks whether $g^s \cdot y^{-c} = r$

Schnorr's ID Protocol (91)

Let $\mathbb{G} = \langle g \rangle$ be a group of prime order q

Prover P proves to verifier V that he knows the discrete log x of a public group element $y = g^x$. It is a 3-move protocol.



Scenario

P sends $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$

V sends $c \xleftarrow{\$} \mathbb{Z}_q$

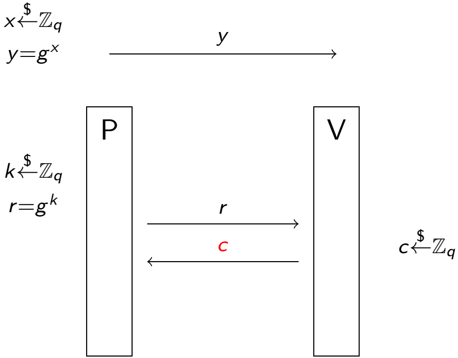
P sends $s = k + cx \bmod q$

V checks whether $g^s \cdot y^{-c} = r$

Schnorr's ID Protocol (91)

Let $\mathbb{G} = \langle g \rangle$ be a group of prime order q

Prover P proves to verifier V that he knows the discrete log x of a public group element $y = g^x$. It is a 3-move protocol.



Scenario

P sends $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$

V sends $c \xleftarrow{\$} \mathbb{Z}_q$

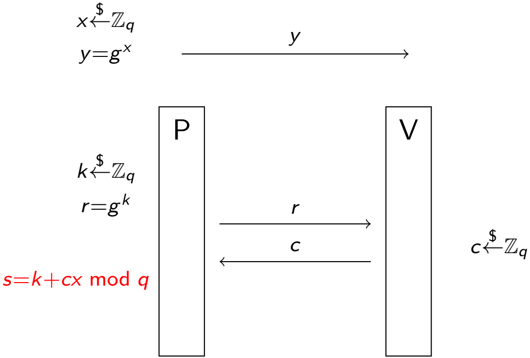
P sends $s = k + cx \bmod q$

V checks whether $g^s \cdot y^{-c} = r$

Schnorr's ID Protocol (91)

Let $\mathbb{G} = \langle g \rangle$ be a group of prime order q

Prover P proves to verifier V that he knows the discrete log x of a public group element $y = g^x$. It is a 3-move protocol.



Scenario

P sends $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$

V sends $c \xleftarrow{\$} \mathbb{Z}_q$

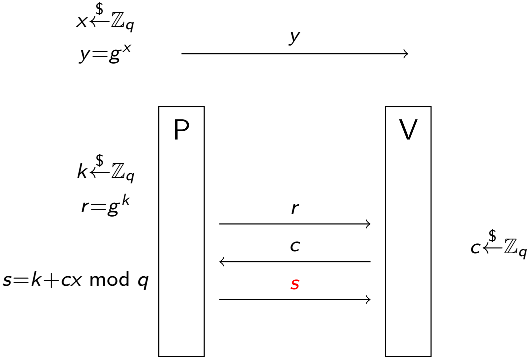
P sends $s = k + cx \bmod q$

V checks whether $g^s \cdot y^{-c} = r$

Schnorr's ID Protocol (91)

Let $\mathbb{G} = \langle g \rangle$ be a group of prime order q

Prover P proves to verifier V that he knows the discrete log x of a public group element $y = g^x$. It is a 3-move protocol.



Scenario

P sends $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$

V sends $c \xleftarrow{\$} \mathbb{Z}_q$

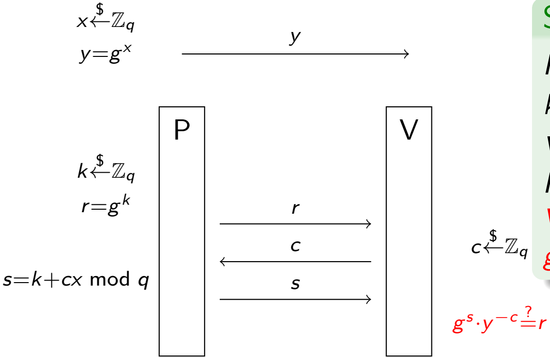
P sends $s = k + cx \bmod q$

V checks whether $g^s \cdot y^{-c} = r$

Schnorr's ID Protocol (91)

Let $\mathbb{G} = \langle g \rangle$ be a group of prime order q

Prover P proves to verifier V that he knows the discrete log x of a public group element $y = g^x$. It is a 3-move protocol.



Scenario

P sends $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$

V sends $c \xleftarrow{\$} \mathbb{Z}_q$

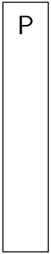
P sends $s = k + cx \bmod q$

V checks whether $g^s \cdot y^{-c} = r$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $H : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.



Signing and Verifying

$\textsc{Sign}$

P computes $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = H(m, r)$
P computes $s = k + cx \bmod q$
P sends $\sigma = (s, c)$

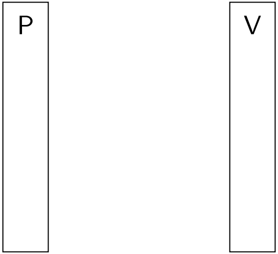
$\textsc{Ver}$

V checks whether $H(m, g^s \cdot y^{-c}) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $H : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.



Signing and Verifying

$\textsc{Sign}$

P computes $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = H(m, r)$
P computes $s = k + cx \bmod q$
P sends $\sigma = (s, c)$

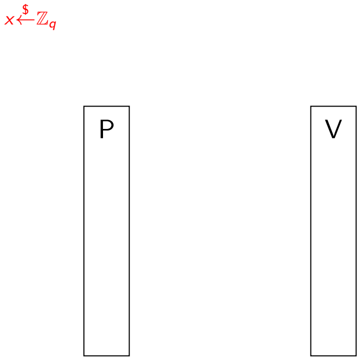
$\textsc{Ver}$

V checks whether $H(m, g^s \cdot y^{-c}) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $H : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.



Signing and Verifying

$\textsc{Sign}$

P computes $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = H(m, r)$
P computes $s = k + cx \bmod q$
P sends $\sigma = (s, c)$

$\textsc{Ver}$

V checks whether $H(m, g^s \cdot y^{-c}) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $H : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.

$x \xleftarrow{\$} \mathbb{Z}_q$

$y = g^x$

P

V

Signing and Verifying

$\textsc{Sign}$

P computes $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = H(m, r)$
P computes $s = k + cx \bmod q$
P sends $\sigma = (s, c)$

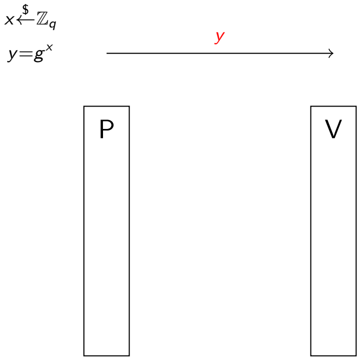
$\textsc{Ver}$

V checks whether $H(m, g^s \cdot y^{-c}) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $H : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.



Signing and Verifying

Sign

P computes $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = H(m, r)$
P computes $s = k + cx \bmod q$
P sends $\sigma = (s, c)$

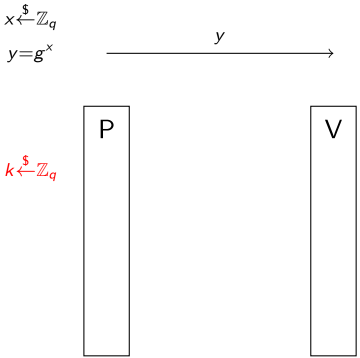
Ver

V checks whether $H(m, g^s \cdot y^{-c}) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $H : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.



Signing and Verifying

$\textsc{Sign}$

P computes $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = H(m, r)$
P computes $s = k + cx \bmod q$
P sends $\sigma = (s, c)$

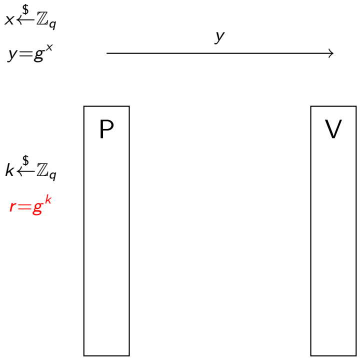
$\textsc{Ver}$

V checks whether $H(m, g^s \cdot y^{-c}) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $H : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.



Signing and Verifying

$\textsc{Sign}$

P computes $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = H(m, r)$
P computes $s = k + cx \bmod q$
P sends $\sigma = (s, c)$

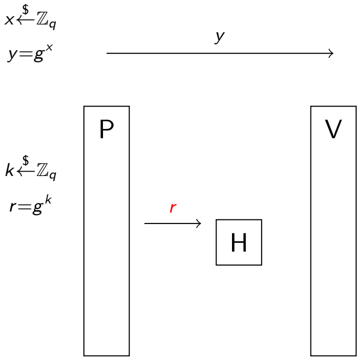
$\textsc{Ver}$

V checks whether $H(m, g^s \cdot y^{-c}) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $H : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.



Signing and Verifying

$\textsc{Sign}$

P computes $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = H(m, r)$
P computes $s = k + cx \bmod q$
P sends $\sigma = (s, c)$

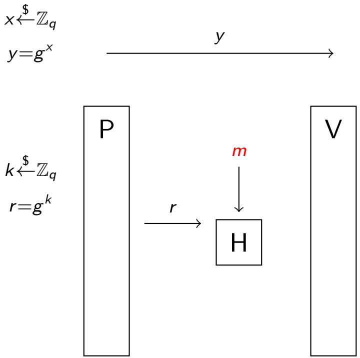
$\textsc{Ver}$

V checks whether $H(m, g^s \cdot y^{-c}) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $H : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.



Signing and Verifying

$\textsc{Sign}$

P computes $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = H(m, r)$
P computes $s = k + cx \bmod q$
P sends $\sigma = (s, c)$

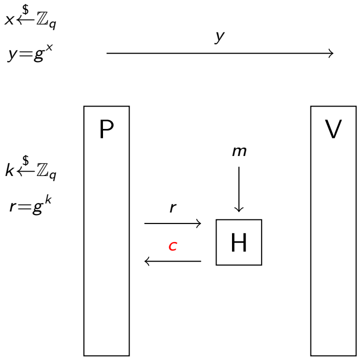
$\textsc{Ver}$

V checks whether $H(m, g^s \cdot y^{-c}) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $H : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.



Signing and Verifying

$\textsc{Sign}$

P computes $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = H(m, r)$
P computes $s = k + cx \bmod q$
P sends $\sigma = (s, c)$

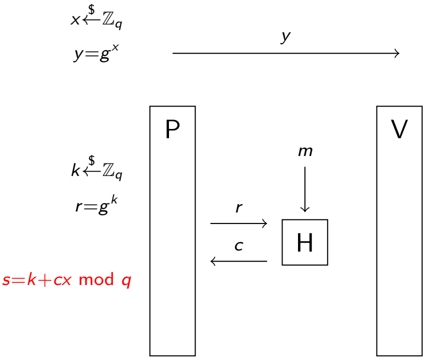
$\textsc{Ver}$

V checks whether $H(m, g^s \cdot y^{-c}) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $H : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.



Signing and Verifying

$\textsc{Sign}$

P computes $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = H(m, r)$
P computes $s = k + cx \bmod q$
P sends $\sigma = (s, c)$

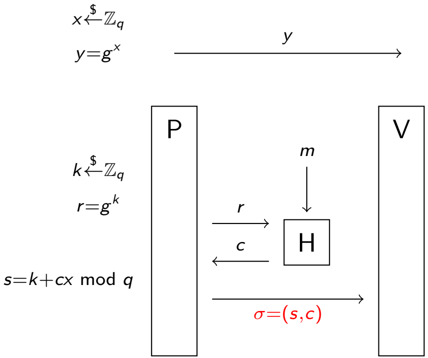
$\textsc{Ver}$

V checks whether $H(m, g^s \cdot y^{-c}) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $H : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.



Signing and Verifying

$\textsc{Sign}$

P computes $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = H(m, r)$
P computes $s = k + cx \bmod q$
P sends $\sigma = (s, c)$

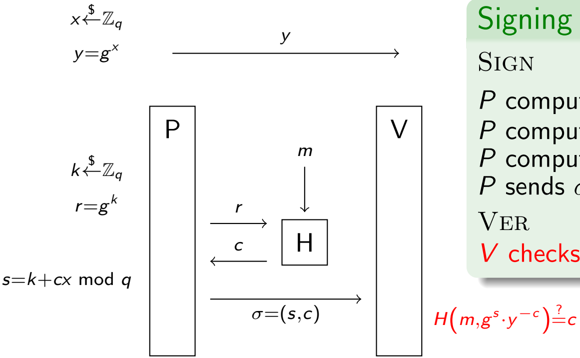
$\textsc{Ver}$

V checks whether $H(m, g^s \cdot y^{-c}) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $H : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.



Signing and Verifying

$\textsc{Sign}$

P computes $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = H(m, r)$
P computes $s = k + cx \bmod q$
P sends $\sigma = (s, c)$

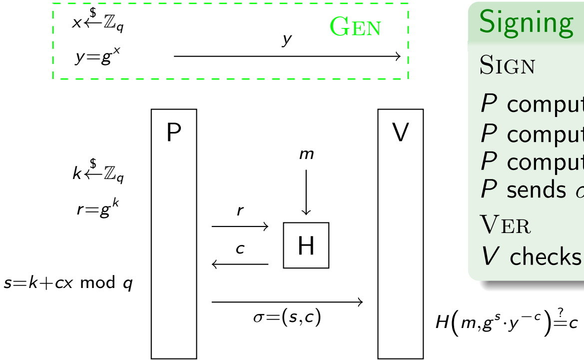
$\textsc{Ver}$

V checks whether $H\left(m, g^s \cdot y^{-c}\right) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $H : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.



Signing and Verifying

Sign

P computes $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = H(m, r)$
P computes $s = k + cx \bmod q$
P sends $\sigma = (s, c)$

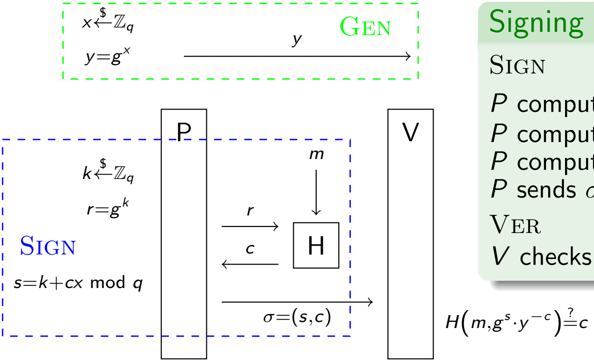
Ver

V checks whether $H(m, g^s \cdot y^{-c}) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $H : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.



Signing and Verifying

SIGN

P computes $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = H(m, r)$
P computes $s = k + cx \bmod q$
P sends $\sigma = (s, c)$

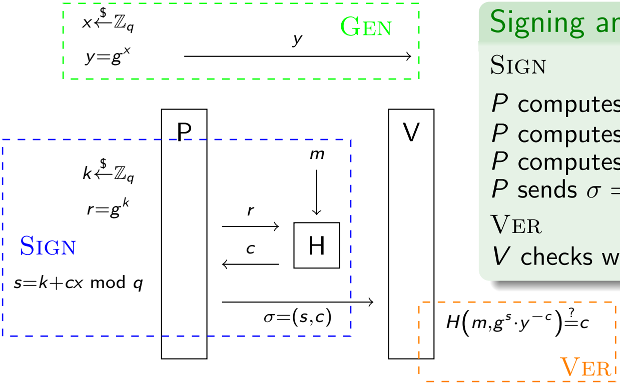
VER

V checks whether $H(m, g^s \cdot y^{-c}) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $H : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.



Signing and Verifying

SIGN

P computes $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = H(m, r)$
P computes $s = k + cx \bmod q$
P sends $\sigma = (s, c)$

VER

V checks whether $H(m, g^s \cdot y^{-c}) = c$

Digital Signature Algorithm (DSA)

- The Digital Signature Algorithm (DSA) is a United States Federal Government standard or FIPS for digital signatures.
- It was proposed by the National Institute of Standards and Technology (NIST) in **August 1991** for use in their Digital Signature Standard (DSS), specified in FIPS 186, adopted in **1993**.
- DSA makes use of a cryptographic hash function $\mathcal{H}$.
  In the original DSS, $\mathcal{H}$ was always SHA, but stronger hash functions from the SHA family are also in use.
- The original DSS constrained the key length to be a multiple of 64 between 512 and 1024 (inclusive).

Digital Signature Algorithm (DSA)

Textbook ElGamal signature scheme

Key generation. $G(1^{\mathrm{k}})$ randomly selects a k-bit prime p and a generator g of $\mathbb{Z}_p^*$.

The secret key is $x \leftarrow \mathbb{Z}_{p-1}$
The public key is $(p, g, y = g^x \mod p)$.

Signature. To sign a message $m \in \mathbb{Z}_{p-1}$, one generates (r, s) such that

$$g^m = y^r r^s \mod p$$

as follows. Randomly select $k \leftarrow \mathbb{Z}_{p-1}^*$, set $r = g^k \mod p$ and

$$s = (m - xr)/k \mod p - 1.$$

Output (r, s).

Verification. Verify that $1 < r < p$ and

$$g^m = y^r r^s \mod p$$

.

Digital Signature Algorithm (DSA)

Hashed ElGamal signature scheme

Key generation. $G(1^{\mathrm{k}})$ randomly selects a k-bit prime p and a generator g of $\mathbb{Z}_p^*$.

The secret key is $x \leftarrow \mathbb{Z}_{p-1}$

The public key is $(p, g, y = g^x \mod p)$ and a hash function $\mathcal{H}$.

Signature. To sign a message $m \in \mathbb{Z}_{p-1}$, one generates (r, s) such that

$$g^{\mathcal{H}(m)} = y^r r^s \mod p$$

as follows. Randomly select $k \leftarrow \mathbb{Z}_{p-1}^*$, set $r = g^k \mod p$ and

$$s = (\mathcal{H}(m) - xr)/k \mod p - 1.$$

Output (r, s).

Verification. Verify that $1 < r < p$ and

$$g^{\mathcal{H}(m)} = y^r r^s \mod p$$

.

Digital Signature Algorithm (DSA)

Hashed ElGamal signature scheme with Schnorr's trick

Key generation. $G(1^{\mathrm{k}})$ randomly selects a k-bit prime p and a generator g of $\mathbb{G} \subset \mathbb{Z}_p^*$ of prime order q.

The secret key is $x \leftarrow \mathbb{Z}_q$

The public key is $(p, q, g, y = g^x \mod p)$ and a hash function $\mathcal{H}$.

Signature. To sign a message $m \in \mathbb{Z}_{p-1}$, one generates (r, s) such that

$$g^{\mathcal{H}(m)} = y^r r^s \mod p$$

as follows. Randomly select $k \leftarrow \mathbb{Z}_q^*$, set $r = g^k \mod p$ and

$$s = (\mathcal{H}(m) - xr)/k \mod q.$$

Output (r, s).

Verification. Verify that $1 < r < q$ and

$$g^{\mathcal{H}(m)} = y^r r^s \mod p$$

.

Digital Signature Algorithm (DSA)

DSA

Key generation. $G(1^{\mathrm{k}})$ randomly selects a k-bit prime p and a generator g of $\mathbb{G} \subset \mathbb{Z}_p^*$ of prime order q.

The secret key is $x \leftarrow \mathbb{Z}_q$

The public key is $(p, q, g, y = g^x \mod p)$ and a hash function $\mathcal{H}$.

Signature. To sign a message $m \in \mathbb{Z}_{p-1}$, one generates (r, s) such that

$$g^{\mathcal{H}(m)} = y^r r^s \mod p$$

as follows. Randomly select $k \leftarrow \mathbb{Z}_q^*$, set $r = g^k \mod p$ and

$$s = (\mathcal{H}(m) + xr)/k \mod q.$$

Output (r, s).

Verification. Verify that $1 < r < q$, calculate $w = s^{-1} \mod q$, $u_1 = \mathcal{H}(m) \cdot w \mod q$, $u_2 = r \cdot w \mod q$ and check whether

$$g^{u_1} y^{u_2} \mod p \mod q = r.$$