

Cryptographie M1

Lecture 4

Ludovic Perret

(slides from C. Bouillaguet and Damien Vergnaud)

Sorbonne Université

2023 – 2024

Contents

1 Digital signatures

- Security Notions for Digital Signatures
- Construction from a Trapdoor Permutation

2 One-time signatures

- Lamport signatures
- Generalizations

3 Fiat-Shamir heuristic and variants

- The Fiat-Shamir heuristic
- 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!**

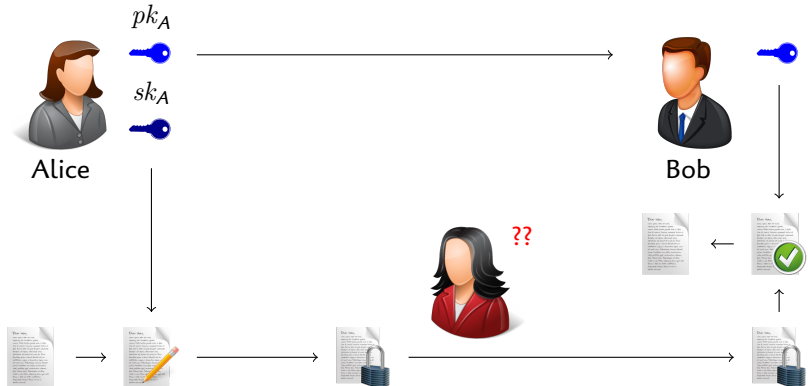
Digital signature schemes

Digital signatures: Alice owns two “keys”

- a **public** key
- a **secret** key

known by everybody (including Bob)

known by Alice only



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

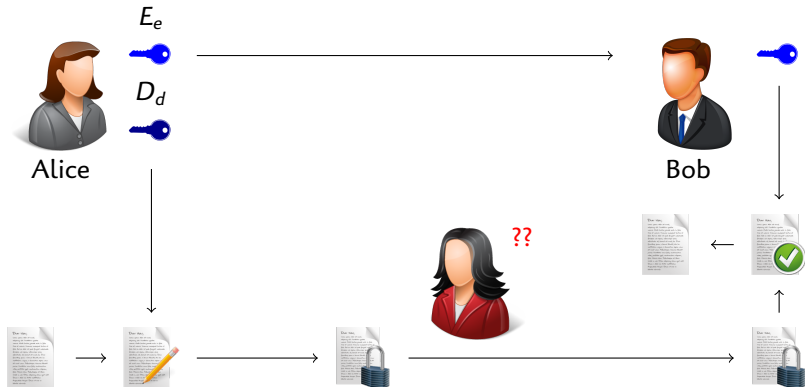
- [**Unbreakability**] the attacker recovers the secret key sk from the public key pk (or an equivalent key if any). This goal is denoted **UB**. Implicitly appeared with public-key cryptography.
- [**Universal Unforgeability**] the attacker, without necessarily having recovered sk , can produce a valid 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 **EUUF**.

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 ...

Trapdoor permutation $\{E : X \rightarrow X\}_{(e,d)}$



Warning: in general it's not that simple. We will explain it later.

... 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 she 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 $e \cdot d \equiv 1 \pmod{\varphi(N)}$.

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 $s = m^d \bmod N$.
 - 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 $m = s^e \bmod N$.

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 $s = m^d \bmod N$.
 - 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 $m = s^e \bmod N$.

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 $m_2 = m/m_1 \pmod{N}$,
- 2 assume Eve asks Alice to sign m_1 and m_2 and receives $S(m_1)$ and $S(m_2)$,
- 3 Eve computes $S(m) = S(m_1) S(m_2) \pmod{N}$ 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 $m_2 = m/m_1 \pmod{N}$,
- 2 assume Eve asks Alice to sign m_1 and m_2 and receives $S(m_1)$ and $S(m_2)$,
- 3 Eve computes $S(m) = S(m_1) S(m_2) \pmod{N}$ 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 $H(m) = E_e(s)$.

↪ **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 $E = \text{RSA}$, 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 $E = \text{RSA}$, 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 $E = \text{RSA}$, 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 .

↪ **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 .

⇒ 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 .

↪ **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** ?

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

- **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** ?

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

- **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** ?

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

- **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** ?

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

- **Key generation:**

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

Public key = $(y_{0,1}, y_{1,1}, \dots, y_{0,k}, y_{1,k})$ which can be published.

Private key = $(x_{0,1}, x_{1,1}, \dots, x_{0,k}, x_{1,k})$ which needs to be kept secret

- **Signature:** if Alice wants to sign $m = m_1 \dots m_k$, she does the following:

- Use her private key $(x_{0,1}, x_{1,1}, \dots, x_{0,k}, x_{1,k})$ to send the signature $s = (x_{m_1,1}, x_{m_1,2}, \dots, x_{m_k,k})$ to Bob.

- **Verification:** to check the validity of $s = (s_1, \dots, s_k)$ on m , Bob does the following:

- Obtain Alice's authentic public key $(y_{0,1}, y_{1,1}, \dots, y_{0,k}, y_{1,k})$.
- Check whether $f(s_i) = y_{m_b,i}$ for all $i \in \llbracket 1, k \rrbracket$.

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

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

- **Key generation:**

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

Public key = $(y_{0,1}, y_{1,1}, \dots, y_{0,k}, y_{1,k})$ which can be published.

Private key = $(x_{0,1}, x_{1,1}, \dots, x_{0,k}, x_{1,k})$ which needs to be kept secret

- **Signature:** if Alice wants to sign $m = m_1 \dots m_k$, she does the following:

- Use her private key $(x_{0,1}, x_{1,1}, \dots, x_{0,k}, x_{1,k})$ to send the signature $s = (x_{m_1,1}, x_{m_1,2}, \dots, x_{m_k,k})$ to Bob.

- **Verification:** to check the validity of $s = (s_1, \dots, s_k)$ on m , Bob does the following:

- Obtain Alice's authentic public key $(y_{0,1}, y_{1,1}, \dots, y_{0,k}, y_{1,k})$.
- Check whether $f(s_i) = y_{m_b,i}$ for all $i \in \llbracket 1, k \rrbracket$.

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

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

- **Key generation:**

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

Public key = $(y_{0,1}, y_{1,1}, \dots, y_{0,k}, y_{1,k})$ which can be published.

Private key = $(x_{0,1}, x_{1,1}, \dots, x_{0,k}, x_{1,k})$ which needs to be kept secret

- **Signature:** if Alice wants to sign $m = m_1 \dots m_k$, she does the following:

- Use her private key $(x_{0,1}, x_{1,1}, \dots, x_{0,k}, x_{1,k})$ to send the signature $s = (x_{m_1,1}, x_{m_1,2}, \dots, x_{m_k,k})$ to Bob.

- **Verification:** to check the validity of $s = (s_1, \dots, s_k)$ on m , Bob does the following:

- Obtain Alice's authentic public key $(y_{0,1}, y_{1,1}, \dots, y_{0,k}, y_{1,k})$.
- Check whether $f(s_i) = y_{m_b,i}$ for all $i \in \llbracket 1, k \rrbracket$.

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 hash function that generates a n -bit message digest, the ideal preimage resistance on a single hash function invocation implies on the order of 2^n operations and 2^n bits of memory effort to find a preimage under a classical computing model.
- 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 .

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. $G(1^k)$ 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 $y = g^x \bmod p$, the public key is (p, g, y) .

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

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

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. $G(1^k)$ 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 $y = g^x \bmod p$, the public key is (p, g, y) .

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

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

Insecure! cf TD 5. Hash the message first!

The Fiat-Shamir heuristic

Fiat, Shamir (1986)

How to Prove Yourself: Practical Solutions to Identification and Signature Problems.
Advances in Cryptology - Crypto'86, Lect. Notes Comput. Science 263, pp. 186-194.

- Fiat and Shamir presented a (zero-knowledge) identification scheme consisting of 3 messages between a **prover** P and a **verifier** V .
- In a 3-pass (public-coin) identification scheme, these messages are called **commitment**, **challenge** and **response**. The challenge is randomly chosen by V .

Fiat-Shamir Transform: replace the challenge by a hash value taken on scheme parameters and t , thereby removing V . This transforms the protocol by making it *non-interactive*.

The intuition is that any "sufficiently random" hash function should preserve the security of the protocol.

The Fiat-Shamir heuristic

Fiat, Shamir (1986)

How to Prove Yourself: Practical Solutions to Identification and Signature Problems.
Advances in Cryptology - Crypto'86, Lect. Notes Comput. Science 263, pp. 186-194.

- Fiat and Shamir presented a (zero-knowledge) identification scheme consisting of 3 messages between a **prover** P and a **verifier** V .
- In a 3-pass (public-coin) identification scheme, these messages are called **commitment**, **challenge** and **response**. The challenge is randomly chosen by V .

Fiat-Shamir Transform: replace the challenge by a hash value taken on scheme parameters and t , thereby removing V . This transforms the protocol by making it *non-interactive*.

The intuition is that any "sufficiently random" hash function should preserve the security of the protocol.

Fiat-Shamir Identification Protocol

- **One-time setup:**

- Trusted party T publishes an RSA modulus $N = pq$ but keeps primes p and q secret;
- Alice select a secret $s \in \mathbb{Z}_n^*$ and publishes $v = s^2 \bmod n$ as her public key.

- **Protocol: (repeated ℓ times)**

- Alice picks $k \in \mathbb{Z}_n^*$ at random and sends $t = k^2 \bmod n$ (**commitment**);
- Bob picks at random a bit $c \in \{0, 1\}$ and sends it to Alice (**challenge**);
- Alice computes $r = k \cdot s^c \bmod n$ and sends it to Bob (**response**);
- Bob accepts the proof if $r \neq 0$ and $r^2 = t \cdot v^c$.

Fiat Shamir signature

ℓ = **security parameter** (e.g. $\ell = 100$); $H : \{0, 1\}^* \rightarrow \{0, 1\}^\ell$ hash function.

- **Key generation:**

- Alice publishes an RSA modulus $N = pq$ but keeps primes p and q secret;
- Alice select a secret $s \in \mathbb{Z}_n^*$, computes $v = s^2 \bmod n$ and publishes (n, v) as her public key.

- **Signature generation:** message $m \in \{0, 1\}^*$

- Alice picks $k_1, \dots, k_\ell \in \mathbb{Z}_n^*$ at random and computes $t_i = k_i^2 \bmod n$ for $i \in \llbracket 1, \ell \rrbracket$;
- She computes $\vec{c} = (c_1, \dots, c_\ell) = H(m, t_1, \dots, t_\ell)$
- Alice computes $r_i = k_i \cdot s^{c_i} \bmod n$ for $i \in \llbracket 1, \ell \rrbracket$ and the signature is $\sigma = (r_1, \dots, r_\ell, \vec{c})$

- **Verification:**

- Bob computes $\tilde{t}_i = r_i^2 v_i^{-c_i}$ for $i \in \llbracket 1, \ell \rrbracket$ and accepts σ iff $\vec{c} = H(m, \tilde{t}_1, \dots, \tilde{t}_\ell)$.

The Fiat-Shamir Transform

The same heuristic transformation can be applied to any 3-pass identification protocol to construct a signature scheme: the message m becomes a parameter of the hash value $e = H(t, m)$.

Intuitively, it should be hard for a forger to find m and a protocol transcript (t, e, s) for which it is true both that $e = H(t, m)$ and (t, e, s) is a valid transcript w.r.t. a given public key.

So-obtained signature schemes are **more efficient** (while hopefully achieving strong security) than other constructions.

The Fiat-Shamir Transform

The same heuristic transformation can be applied to any 3-pass identification protocol to construct a signature scheme: the message m becomes a parameter of the hash value $e = H(t, m)$.

Intuitively, it should be hard for a forger to find m and a protocol transcript (t, e, s) for which it is true both that $e = H(t, m)$ and (t, e, s) is a valid transcript w.r.t. a given public key.

So-obtained signature schemes are **more efficient** (while hopefully achieving strong security) than other constructions.

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.

P

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.

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.

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

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

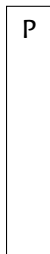
Schnorr's ID Protocol (91)

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.

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

$$y = g^x$$



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.

$$\begin{array}{l} x \xleftarrow{\$} \mathbb{Z}_q \\ y = g^x \end{array} \xrightarrow{\quad y \quad}$$

P

V

Scenario

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

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

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

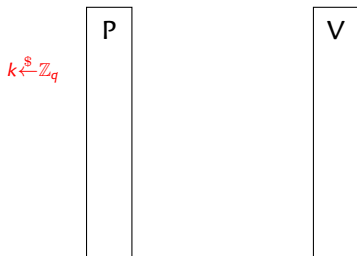
V checks whether $g^s \cdot y^{-c} = r$

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.

$$\begin{array}{l} x \xleftarrow{\$} \mathbb{Z}_q \\ y = g^x \end{array} \xrightarrow{y}$$



Scenario

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

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

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

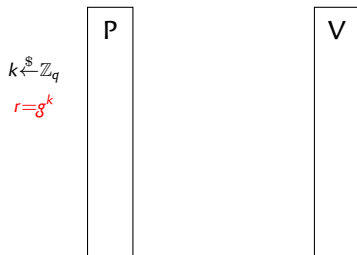
V checks whether $g^s \cdot y^{-c} = r$

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.

$$\begin{array}{l} x \xleftarrow{\$} \mathbb{Z}_q \\ y = g^x \end{array} \xrightarrow{y}$$



Scenario

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

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

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

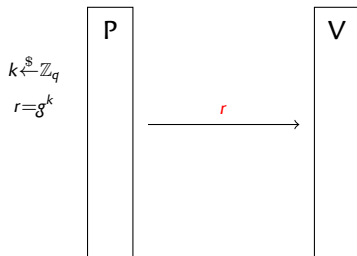
V checks whether $g^s \cdot y^{-c} = r$

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.

$$\begin{array}{l} x \xleftarrow{\$} \mathbb{Z}_q \\ y = g^x \end{array} \quad \xrightarrow{\quad y \quad}$$



Scenario

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

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

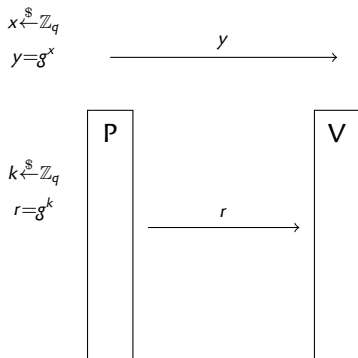
P sends $s = k + cx \bmod q$

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 $s = k + cx \bmod q$

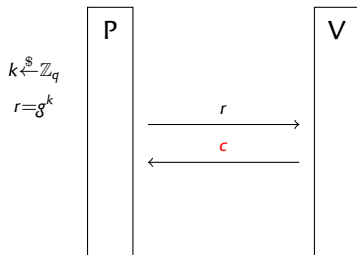
V checks whether $g^s \cdot y^{-c} = r$

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.

$$\begin{array}{l} x \xleftarrow{\$} \mathbb{Z}_q \\ y = g^x \end{array} \quad \xrightarrow{\quad y \quad}$$



Scenario

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

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

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

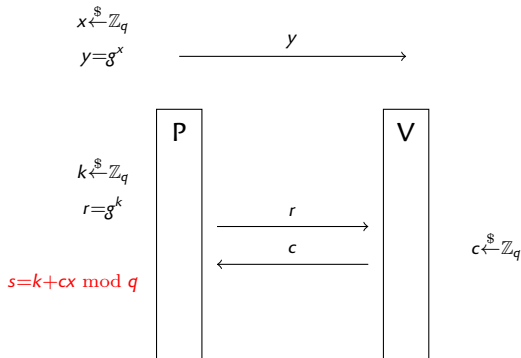
V checks whether $g^s \cdot y^{-c} = r$

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

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

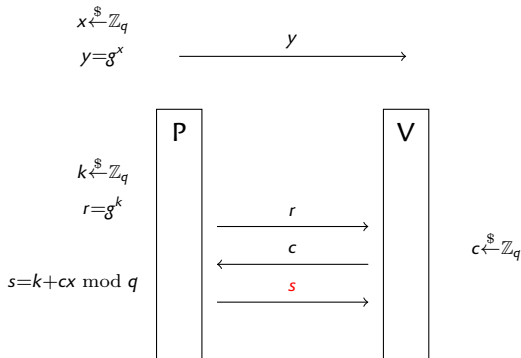
P sends $s = k + cx \bmod q$

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

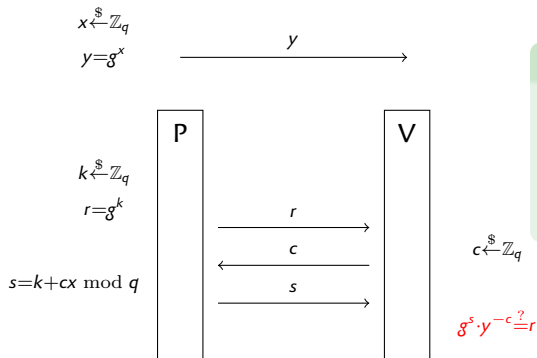
P sends $s = k + cx \pmod q$

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 $s = k + cx \pmod q$

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

Schnorr Signatures (via the Fiat-Shamir Transform)

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

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\text{GEN}, \text{SIGN}, \text{VER})$ defined as follows.

P

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 $H : \{0, 1\}^* \mapsto \mathbb{Z}_q$

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\text{GEN}, \text{SIGN}, \text{VER})$ defined as follows.

P

V

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 $H : \{0, 1\}^* \mapsto \mathbb{Z}_q$

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\text{GEN}, \text{SIGN}, \text{VER})$ defined as follows.

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

P

V

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 $H : \{0, 1\}^* \mapsto \mathbb{Z}_q$

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\text{GEN}, \text{SIGN}, \text{VER})$ defined as follows.

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

$$y = g^x$$

P

V

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 $\sigma = (s, c)$

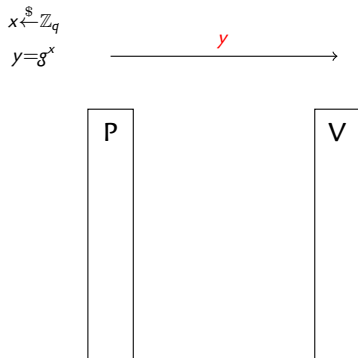
VER

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

Schnorr Signatures (via the Fiat-Shamir Transform)

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

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\text{GEN}, \text{SIGN}, \text{VER})$ 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 $\sigma = (s, c)$

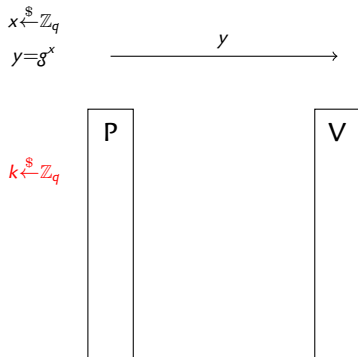
VER

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

Schnorr Signatures (via the Fiat-Shamir Transform)

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

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\text{GEN}, \text{SIGN}, \text{VER})$ 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 $\sigma = (s, c)$

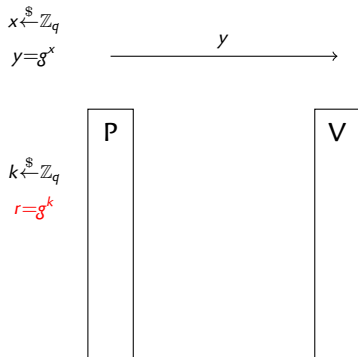
VER

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

Schnorr Signatures (via the Fiat-Shamir Transform)

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

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\text{GEN}, \text{SIGN}, \text{VER})$ 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 $\sigma = (s, c)$

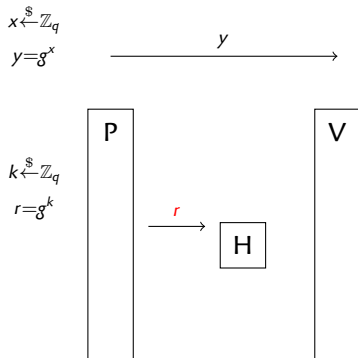
VER

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

Schnorr Signatures (via the Fiat-Shamir Transform)

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

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\text{GEN}, \text{SIGN}, \text{VER})$ 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 $\sigma = (s, c)$

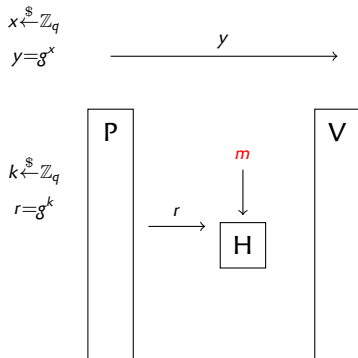
VER

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

Schnorr Signatures (via the Fiat-Shamir Transform)

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

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\text{GEN}, \text{SIGN}, \text{VER})$ 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 $\sigma = (s, c)$

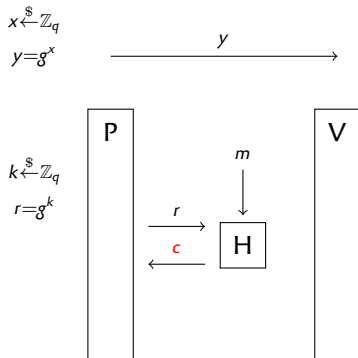
VER

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

Schnorr Signatures (via the Fiat-Shamir Transform)

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

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\text{GEN}, \text{SIGN}, \text{VER})$ 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 $\sigma = (s, c)$

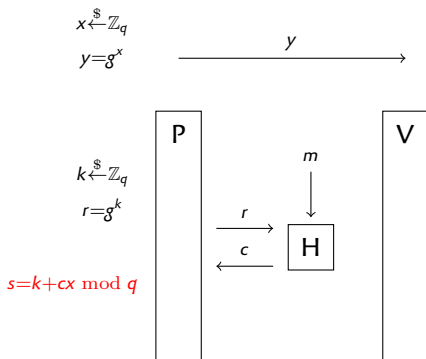
VER

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

Schnorr Signatures (via the Fiat-Shamir Transform)

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

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\text{GEN}, \text{SIGN}, \text{VER})$ 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 $\sigma = (s, c)$

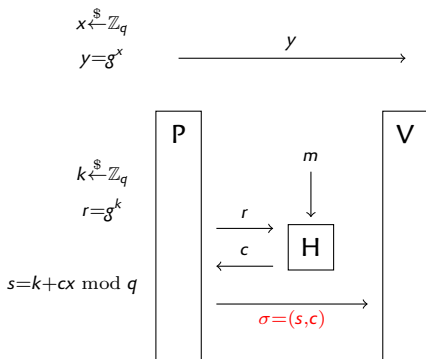
VER

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

Schnorr Signatures (via the Fiat-Shamir Transform)

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

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\text{GEN}, \text{SIGN}, \text{VER})$ defined as follows.



Signing and Verifying

SIGN

P computes $r = g^k$ where $k \leftarrow \mathbb{Z}_q$

P computes $c = H(m, r)$

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

P sends $\sigma = (s, c)$

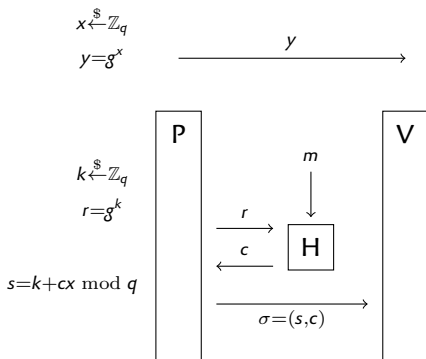
VER

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

Schnorr Signatures (via the Fiat-Shamir Transform)

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

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\text{GEN}, \text{SIGN}, \text{VER})$ 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

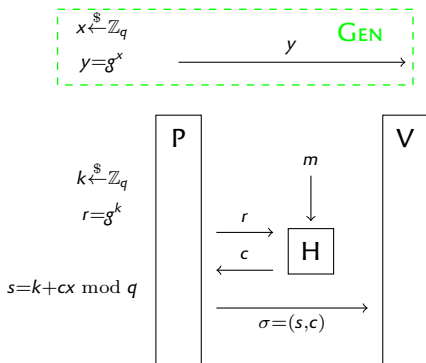
V checks whether $H(m, g^s \cdot y^{-c}) = c$

$$H(m, g^s \cdot y^{-c}) \stackrel{?}{=} c$$

Schnorr Signatures (via the Fiat-Shamir Transform)

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

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\text{GEN}, \text{SIGN}, \text{VER})$ 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 $s = k + cx \text{ mod } q$

P sends $\sigma = (s, c)$

VER

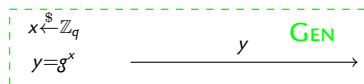
V checks whether $H(m, g^s \cdot y^{-c}) = c$

$$H(m, g^s \cdot y^{-c}) \stackrel{?}{=} c$$

Schnorr Signatures (via the Fiat-Shamir Transform)

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

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\text{GEN}, \text{SIGN}, \text{VER})$ defined as follows.



Signing and Verifying

SIGN

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

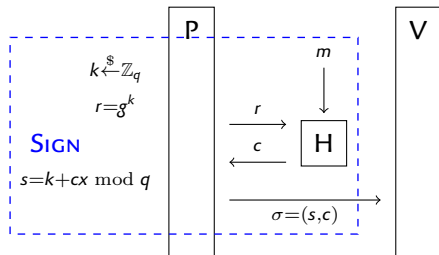
P computes $c = H(m, r)$

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

P sends $\sigma = (s, c)$

VER

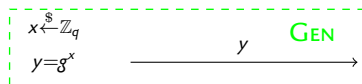
V checks whether $H(m, g^s \cdot y^{-c}) \stackrel{?}{=} c$



Schnorr Signatures (via the Fiat-Shamir Transform)

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

Schnorr's signature scheme Σ_H is a tuple of probabilistic algorithms $\Sigma_H = (\text{GEN}, \text{SIGN}, \text{VER})$ defined as follows.



Signing and Verifying

SIGN

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

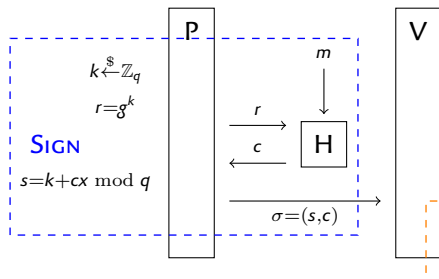
P computes $c = H(m, r)$

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

P sends $\sigma = (s, c)$

VER

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



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. $G(1^k)$ 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 $(p, g, y = g^x \bmod p)$.

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

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

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

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

Output (r, s) .

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

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

Digital Signature Algorithm (DSA)

Hashed ElGamal signature scheme

Key generation. $G(1^k)$ 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 $(p, g, y = g^x \bmod p)$ and a hash function \mathcal{H} .

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

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

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

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

Output (r, s) .

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

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

Digital Signature Algorithm (DSA)

Hashed ElGamal signature scheme with Schnorr's trick

Key generation. $G(1^k)$ 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 $(p, q, g, y = g^x \bmod p)$ and a hash function \mathcal{H} .

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

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

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

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

Output (r, s) .

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

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

Digital Signature Algorithm (DSA)

DSA

Key generation. $G(1^k)$ 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 $(p, q, g, y = g^x \bmod p)$ and a hash function \mathcal{H} .

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

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

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

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

Output (r, s) .

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

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