CRYPTA
Cours 3 - Signatures numériques

Damien Vergnaud

Sorbonne Université – CNRS – IUF

Contents

Why “Provable Security” ?

Once a cryptosystem is described, how can we prove its security?

by trying to exhibit an attack

- attack found
  $\Rrightarrow$ system insecure!
- attack not found
  $\Rrightarrow$?

by proving that no attack exists under some assumptions

- attack found
  $\Rrightarrow$ false assumption

- **"Textbook" cryptosystems cannot be used as such**
- **Praticitioners need formatting rules to ensure operability.**
  $\rightsquigarrow$ Paddings are used in practice : heuristic security
- **Provable security is needed in upcoming systems.**
  This is no longer just theory.

Why "Provable Security" ?

Once a cryptosystem is described, how can we prove its security?

by trying to exhibit an attack

- attack found
  $\Rrightarrow$ system insecure!
- attack not found
  $\Rrightarrow$?

by proving that no attack exists under some assumptions

- attack found
  $\Rrightarrow$ false assumption

- **"Textbook" cryptosystems cannot be used as such**
- **Pratictioners need formatting rules to ensure operability.**
  $\rightsquigarrow$ Paddings are used in practice : heuristic security
- **Provable security is needed in upcoming systems.**
  This is no longer just theory.

Why "Provable Security" ?

Once a cryptosystem is described, how can we prove its security?

by trying to exhibit an attack

- attack found
  $\Rrightarrow$ system insecure!
- attack not found
  $\Rrightarrow$?

by proving that no attack exists under some assumptions

- attack found
  $\Rrightarrow$ false assumption

- **"Textbook" cryptosystems cannot be used as such**
- **Pratictioners need formatting rules to ensure operability.**
  $\rightsquigarrow$ Paddings are used in practice : heuristic security
- **Provable security is needed in upcoming systems.**
  This is no longer just theory.

Who is the bad guy?

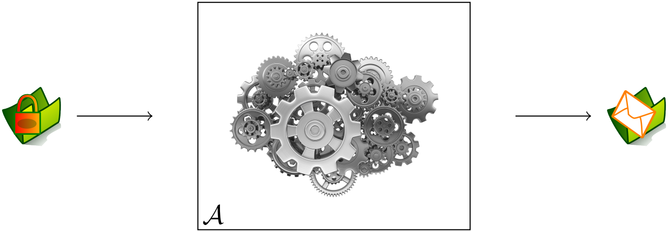
We are protecting ourselves from the evil Eve, who

- is a probabilistic polynomial time Turing machine (PPTM) **(Church-Turing thesis)**
- knows all the algorithms **(Kerckoff's principles)**
- has full access to communication media

Proof by reduction

$\mathcal{A}$ adversary against e.g. one-wayness

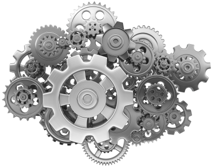
Proof by reduction



$\mathcal{A}$ adversary against e.g. one-wayness

Proof by reduction

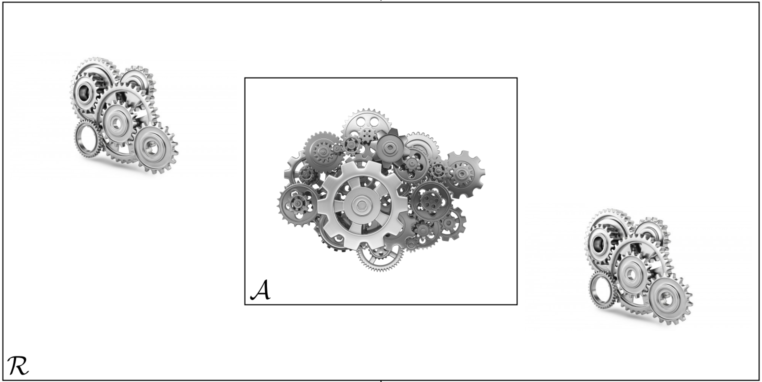
Instance $\mathcal{I}$ of a problem $\mathcal{P}$



$\mathcal{A}$

Proof by reduction

Instance $\mathcal{I}$ of a problem $\mathcal{P}$

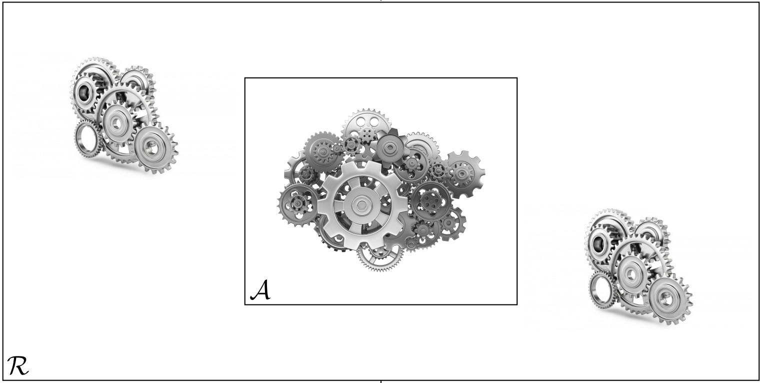


Solution of $\mathcal{I}$

Proof by reduction

Instance $\mathcal{I}$ of a problem $\mathcal{P}$

$\downarrow$



$\downarrow$

Solution of $\mathcal{I}$

$\mathcal{P}$ intractable $\rightarrow$ contradiction

The Methodology of “Provable Security”

1. Define goal of adversary
2. Define security model
3. Define complexity assumptions
4. Provide a proof by reduction
5. Check proof
6. Interpret proof

The Methodology of "Provable Security"

1. Define goal of adversary
2. Define security model
3. Define complexity assumptions
4. Provide a proof by reduction
5. Check proof
6. Interpret proof

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/signing key.
- Anyone can verify Alice's signature, since everyone can obtain her public/verification 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/signing key.
- Anyone can verify Alice's signature, since everyone can obtain her public/verification 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/signing key.
- Anyone can verify Alice's signature, since everyone can obtain her public/verification key.
- the verifier is convinced that only Alice could have produced the signature
  - **only Alice knows her private key!**

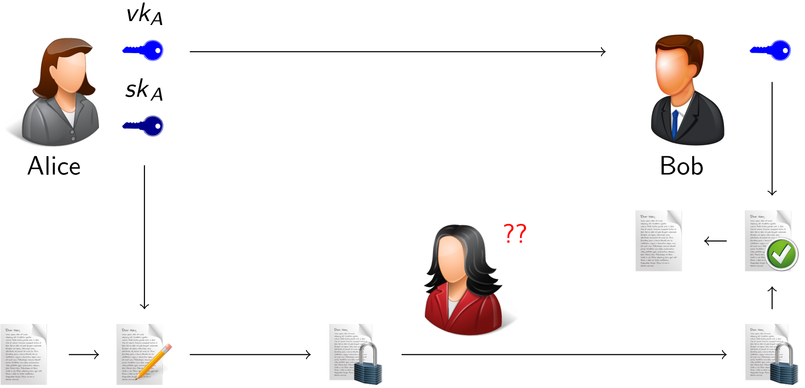
Digital Signatures

- 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/signing key.
- Anyone can verify Alice's signature, since everyone can obtain her public/verification 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 — known by everybody (including Bob)
- a secret key — 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

An digital signature scheme is a triple of algorithms $(\mathcal{K}, \mathcal{S}, \mathcal{V})$ where

- $\mathcal{K}$ is a probabilistic **key generation algorithm** which returns random pairs of secret and verification keys (sk, vk) depending on the security parameter κ,
- $\mathcal{S}$ is a (probabilistic) **signature algorithm** which takes on input a signing key sk and a *message* $m \in \mathcal{M}$, runs on a random tape $u \in \mathcal{U}$ and returns $s \in S$,
- $\mathcal{V}$ is a deterministic **verification algorithm** which takes on input a verification key vk, a message m and $s \in S$ and outputs a bit in $\{0, 1\}$. If $\mathcal{V}_{vk}(m, s) = 1$, then s is a *signature* on m for vk.

If $(sk, vk) \leftarrow \mathcal{K}$, then $\mathcal{V}_{vk}(m, \mathcal{S}_{sk}(m, u)) = 1$ for all $(m, u) \in \mathcal{M} \times \mathcal{U}$.

Signature Schemes

An digital signature scheme is a triple of algorithms $(\mathcal{K}, \mathcal{S}, \mathcal{V})$ where

- $\mathcal{K}$ is a probabilistic **key generation algorithm** which returns random pairs of secret and verification keys (sk, vk) depending on the security parameter κ,
- $\mathcal{S}$ is a (probabilistic) **signature algorithm** which takes on input a signing key sk and a *message* $m \in \mathcal{M}$, runs on a random tape $u \in \mathcal{U}$ and returns $s \in S$,
- $\mathcal{V}$ is a deterministic **verification algorithm** which takes on input a verification key vk, a message m and $s \in S$ and outputs a bit in $\{0, 1\}$. If $\mathcal{V}_{vk}(m, s) = 1$, then s is a *signature* on m for vk.

If $(sk, vk) \leftarrow \mathcal{K}$, then $\mathcal{V}_{vk}(m, \mathcal{S}_{sk}(m, u)) = 1$ for all $(m, u) \in \mathcal{M} \times \mathcal{U}$.

Signature Schemes

An digital signature scheme is a triple of algorithms $(\mathcal{K}, \mathcal{S}, \mathcal{V})$ where

- $\mathcal{K}$ is a probabilistic **key generation algorithm** which returns random pairs of secret and verification keys (sk, vk) depending on the security parameter κ,
- $\mathcal{S}$ is a (probabilistic) **signature algorithm** which takes on input a signing key sk and a *message* $m \in \mathcal{M}$, runs on a random tape $u \in \mathcal{U}$ and returns $s \in S$,
- $\mathcal{V}$ is a deterministic **verification algorithm** which takes on input a verification key vk, a message m and $s \in S$ and outputs a bit in $\{0, 1\}$. If $\mathcal{V}_{vk}(m, s) = 1$, then s is a *signature* on m for vk.

If $(sk, vk) \leftarrow \mathcal{K}$, then $\mathcal{V}_{vk}(m, \mathcal{S}_{sk}(m, u)) = 1$ for all $(m, u) \in \mathcal{M} \times \mathcal{U}$.

Security Goals

[Unbreakability] the attacker recovers the secret key *sk* from the verification key *vk* (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

Chosen-Message Security

Goldwasser, Micali, Rivest (1988)
A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks.
SIAM J. Comput. 17(2) pp. 281-308.

Formally, an signature scheme is said to be (q, τ, ε)-secure if for any adversary $\mathcal{A}$ with running time upper-bounded by τ,

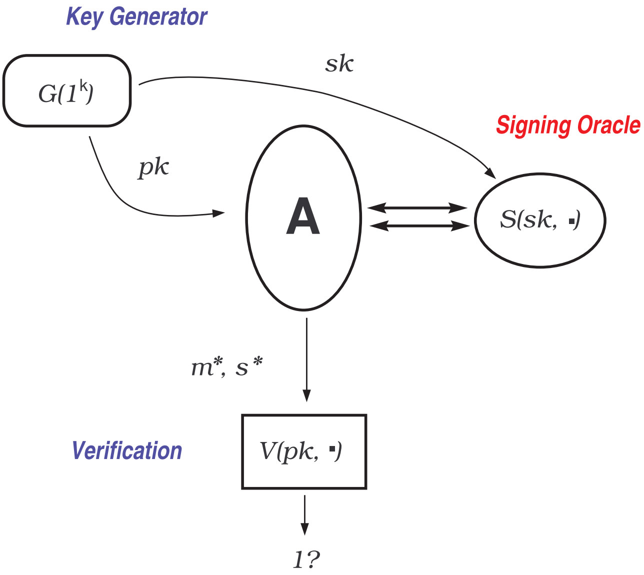
$$\mathsf{Succ}^{\mathsf{EUF-CMA}}(\mathcal{A}) = \Pr\left[\begin{array}{l} (sk, vk) \leftarrow \mathcal{K}(1^{\mathsf{k}}), \\ (m^*, s^*) \leftarrow \mathcal{A}^{\mathcal{S}(sk,\cdot)}(vk), \\ \mathcal{V}(vk, m^*, s^*) = 1 \end{array}\right] < \varepsilon ,$$

where the probability is taken over all random choices.

The notation $\mathcal{A}^{\mathcal{S}(sk,\cdot)}$ means that the adversary has access to a signing oracle throughout the game, but at most q times.

The message m^* output by $\mathcal{A}$ was **never** requested to the signing oracle...

EUF-CMA: Playing the Game



Outline

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.

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

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

- **Key generation:**
  - Consider $f : X \longrightarrow Y$ a **one-way function**.
    e.g.
    $$\begin{array}{cccc} f: & \mathbb{Z}_q & \longrightarrow & \mathbb{G} \\ & x & \longmapsto & f(x) = g^x \end{array}$$
  - Select two random elements $x_0, x_1 \in X$.
  - Compute their images $y_i = f(x_i)$ for $i \in \{0, 1\}$.

  Verification key $vk = (y_0, y_1)$ which can be published.
  Signing key $sk = (x_0, x_1)$ which needs to be kept secret

- **Signature:** if Alice wants to sign a bit b, she does the following:
  - Use her signing 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 verification 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:**
  - Consider $f : X \longrightarrow Y$ a **one-way function**.
    e.g.
    $$\begin{array}{rccc} f: & \mathbb{Z}_q & \longrightarrow & \mathbb{G} \\ & x & \longmapsto & f(x) = g^x \end{array}$$
  - Select two random elements $x_0, x_1 \in X$.
  - Compute their images $y_i = f(x_i)$ for $i \in \{0, 1\}$.

  Verification key $vk = (y_0, y_1)$ which can be published.
  Signing key $sk = (x_0, x_1)$ which needs to be kept secret
- **Signature:** if Alice wants to sign a bit b, she does the following:
  - Use her signing 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 verification 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:**
  - Consider $f : X \longrightarrow Y$ a **one-way function**.
    e.g.
    $$\begin{array}{rccc} f: & \mathbb{Z}_q & \longrightarrow & \mathbb{G} \\ & x & \longmapsto & f(x) = g^x \end{array}$$
  - Select two random elements $x_0, x_1 \in X$.
  - Compute their images $y_i = f(x_i)$ for $i \in \{0, 1\}$.

  Verification key $vk = (y_0, y_1)$ which can be published.
  Signing key $sk = (x_0, x_1)$ which needs to be kept secret
- **Signature:** if Alice wants to sign a bit b, she does the following:
  - Use her signing 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 verification 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 $f : X \longrightarrow Y$ 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 $j \in [\![1,k]\!]$.

  Verification key $vk = (y_{0,1}, y_{1,1}, \ldots, y_{0,k}, y_{1,k})$ which can be published.
  Signing key $sk = (x_{0,1}, x_{1,1}, \ldots, x_{0,k}, x_{1,k})$ which needs to be kept secret

- **Signature:** if Alice wants to sign $m = m_1 \ldots m_k$, she does the following:
  - Use her signing 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 verification 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 $f : X \longrightarrow Y$ 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 $j \in [\![1,k]\!]$.

  Verification key $vk = (y_{0,1}, y_{1,1}, \ldots, y_{0,k}, y_{1,k})$ which can be published.
  Signing key $sk = (x_{0,1}, x_{1,1}, \ldots, x_{0,k}, x_{1,k})$ which needs to be kept secret
- **Signature:** if Alice wants to sign $m = m_1 \ldots m_k$, she does the following:
  - Use her signing 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 verification 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 $f : X \longrightarrow Y$ 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 $j \in [\![1,k]\!]$.

  Verification key $vk = (y_{0,1}, y_{1,1}, \ldots, y_{0,k}, y_{1,k})$ which can be published.
  Signing key $sk = (x_{0,1}, x_{1,1}, \ldots, x_{0,k}, x_{1,k})$ which needs to be kept secret

- **Signature:** if Alice wants to sign $m = m_1 \ldots m_k$, she does the following:
  - Use her signing 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 verification 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** ?

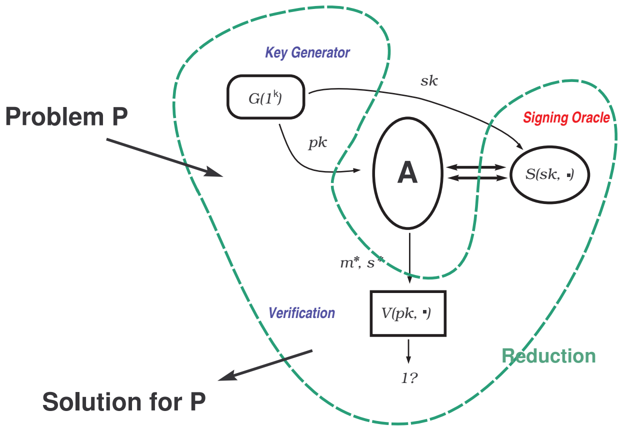
Theorem

If f is (τ, ε)-one way then Lamport's signature scheme (for k-bit messages) is $(1, \tau', 2k \cdot \varepsilon)$-EUF-CMA secure, with $\tau' = \tau + (2k-1)\mathcal{T}_{\mathsf{Eval}}$.

- In other words: If there is an Adversary $\mathcal{A}$ that chooses
  - a message $m \in \{0,1\}^k$ for Alice to legitimately authenticate
  - forges a message $m' \neq m$ with probability at least ε

  Then there is an Adversary $\mathcal{B}$ that can break the one-wayness of the function f with probability at least $\varepsilon/2k$ operates in time roughly the same as $\mathcal{A}$

Simulating the Attacker's Environment



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

Proof. $\mathcal{B}$ gets as input the description of f and $y^\star \in Y$.

- $\mathcal{B}$ picks as input an index $(i^\star, j^\star) \in \{0,1\} \times [\![1,k]\!]$
- $\mathcal{B}$ selects $2k-1$ random elements $x_{0,1}, \dots, \widehat{x_{i^\star,j^\star}}, \dots, x_{1,k} \in X$.
- $\mathcal{B}$ computes their images $y_{i,j} = f(x_{i,j}) = \mathrm{Eval}(x_{i,j})$ for $(i,j) \in \{0,1\} \times [\![1,k]\!] \setminus \{(i^\star, j^\star)\}$.
- $\mathcal{B}$ sets $y_{i^\star,j^\star} = y^\star$
- $\mathcal{B}$ executes $\mathcal{A}$ on the verification key $(y_{0,1}, y_{1,1}, \dots, y_{0,k}, y_{1,k})$
- At some point $\mathcal{A}$ query **one** message $m = m_1 \dots m_k$ to the signature oracle
  - If $m_{j^\star} = i^\star$ then $\mathcal{B}$ aborts the simulation (probability $1/2$),
  - otherwise $\mathcal{B}$ outputs a valid signature on m thanks to its knowledge of $x_{0,1}, \dots, \widehat{x_{i^\star,j^\star}}, \dots, x_{1,k}$.
- Eventually, $\mathcal{A}$ outputs a signature s' on a message $m' \neq m$ and $\mathcal{B}$ outputs $s'_{j^\star}$. The message m' differs from m in at least one position. If it is the $j^\star$-th position (probability $1/k$) and if the signature is valid (probability ε) we have $f(s'_{j^\star}) = y_{i^\star,j^\star} = y^\star$.

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

Proof. $\mathcal{B}$ gets as input the description of f and $y^\star \in Y$.

- $\mathcal{B}$ picks as input an index $(i^\star, j^\star) \in \{0,1\} \times [\![1,k]\!]$
- $\mathcal{B}$ selects $2k-1$ random elements $x_{0,1}, \dots, \widehat{x_{i^\star,j^\star}}, \dots, x_{1,k} \in X$.
- $\mathcal{B}$ computes their images $y_{i,j} = f(x_{i,j}) = \mathrm{Eval}(x_{i,j})$ for $(i,j) \in \{0,1\} \times [\![1,k]\!] \setminus \{(i^\star, j^\star)\}$.
- $\mathcal{B}$ sets $y_{i^\star,j^\star} = y^\star$
- $\mathcal{B}$ executes $\mathcal{A}$ on the verification key $(y_{0,1}, y_{1,1}, \dots, y_{0,k}, y_{1,k})$
- At some point $\mathcal{A}$ query **one** message $m = m_1 \dots m_k$ to the signature oracle
  - If $m_{j^\star} = i^\star$ then $\mathcal{B}$ aborts the simulation (probability 1/2),
  - otherwise $\mathcal{B}$ outputs a valid signature on m thanks to its knowledge of $x_{0,1}, \dots, \widehat{x_{i^\star,j^\star}}, \dots, x_{1,k}$.
- Eventually, $\mathcal{A}$ outputs a signature s' on a message $m' \neq m$ and $\mathcal{B}$ outputs $s'_{j^\star}$. The message m' differs from m in at least one position. If it is the $j^\star$-th position (probability $1/k$) and if the signature is valid (probability ε) we have $f(s'_{j^\star}) = y_{i^\star,j^\star} = y^\star$.

How to sign k bits just once ?

Proof. $\mathcal{B}$ gets as input the description of f and $y^\star \in Y$.

- $\mathcal{B}$ picks as input an index $(i^\star, j^\star) \in \{0,1\} \times [\![1,k]\!]$
- $\mathcal{B}$ selects $2k-1$ random elements $x_{0,1}, \dots, \widehat{x_{i^\star,j^\star}}, \dots, x_{1,k} \in X$.
- $\mathcal{B}$ computes their images $y_{i,j} = f(x_{i,j}) = \mathsf{Eval}(x_{i,j})$ for $(i,j) \in \{0,1\} \times [\![1,k]\!] \setminus \{(i^\star, j^\star)\}$.
- $\mathcal{B}$ sets $y_{i^\star,j^\star} = y^\star$
- $\mathcal{B}$ executes $\mathcal{A}$ on the verification key $(y_{0,1}, y_{1,1}, \dots, y_{0,k}, y_{1,k})$
- At some point $\mathcal{A}$ query **one** message $m = m_1 \dots m_k$ to the signature oracle
  - If $m_{j^\star} = i^\star$ then $\mathcal{B}$ aborts the simulation (probability $1/2$),
  - otherwise $\mathcal{B}$ outputs a valid signature on m thanks to its knowledge of $x_{0,1}, \dots, \widehat{x_{i^\star,j^\star}}, \dots, x_{1,k}$.
- Eventually, $\mathcal{A}$ outputs a signature s' on a message $m' \neq m$ and $\mathcal{B}$ outputs $s'_{j^\star}$. The message m' differs from m in at least one position. If it is the $j^\star$-th position (probability $1/k$) and if the signature is valid (probability ε) we have $f(s'_{j^\star}) = y_{i^\star,j^\star} = y^\star$.

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

Proof. $\mathcal{B}$ gets as input the description of f and $y^\star \in Y$.

- $\mathcal{B}$ picks as input an index $(i^\star, j^\star) \in \{0,1\} \times [\![1, k]\!]$
- $\mathcal{B}$ selects $2k-1$ random elements $x_{0,1}, \dots, \widehat{x_{i^\star,j^\star}}, \dots, x_{1,k} \in X$.
- $\mathcal{B}$ computes their images $y_{i,j} = f(x_{i,j}) = \mathrm{Eval}(x_{i,j})$ for $(i,j) \in \{0,1\} \times [\![1, k]\!] \setminus \{(i^\star, j^\star)\}$.
- $\mathcal{B}$ sets $y_{i^\star,j^\star} = y^\star$
- $\mathcal{B}$ executes $\mathcal{A}$ on the verification key $(y_{0,1}, y_{1,1}, \dots, y_{0,k}, y_{1,k})$
- At some point $\mathcal{A}$ query **one** message $m = m_1 \dots m_k$ to the signature oracle
  - If $m_{j^\star} = i^\star$ then $\mathcal{B}$ aborts the simulation (probability $1/2$),
  - otherwise $\mathcal{B}$ outputs a valid signature on m thanks to its knowledge of $x_{0,1}, \dots, \widehat{x_{i^\star,j^\star}}, \dots, x_{1,k}$.
- Eventually, $\mathcal{A}$ outputs a signature s' on a message $m' \neq m$ and $\mathcal{B}$ outputs $s'_{j^\star}$. The message m' differs from m in at least one position. If it is the $j^\star$-th position (probability $1/k$) and if the signature is valid (probability ε) we have $f(s'_{j^\star}) = y_{i^\star,j^\star} = y^\star$.

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

Proof. $\mathcal{B}$ gets as input the description of f and $y^\star \in Y$.

- $\mathcal{B}$ picks as input an index $(i^\star, j^\star) \in \{0,1\} \times [\![1,k]\!]$
- $\mathcal{B}$ selects $2k-1$ random elements $x_{0,1}, \dots, \widehat{x_{i^\star,j^\star}}, \dots, x_{1,k} \in X$.
- $\mathcal{B}$ computes their images $y_{i,j} = f(x_{i,j}) = \mathrm{Eval}(x_{i,j})$ for $(i,j) \in \{0,1\} \times [\![1,k]\!] \setminus \{(i^\star, j^\star)\}$.
- $\mathcal{B}$ sets $y_{i^\star,j^\star} = y^\star$
- $\mathcal{B}$ executes $\mathcal{A}$ on the verification key $(y_{0,1}, y_{1,1}, \dots, y_{0,k}, y_{1,k})$
- At some point $\mathcal{A}$ query **one** message $m = m_1 \dots m_k$ to the signature oracle
  - If $m_{j^\star} = i^\star$ then $\mathcal{B}$ aborts the simulation (probability $1/2$),
  - otherwise $\mathcal{B}$ outputs a valid signature on m thanks to its knowledge of $x_{0,1}, \dots, \widehat{x_{i^\star,j^\star}}, \dots, x_{1,k}$.
- Eventually, $\mathcal{A}$ outputs a signature s' on a message $m' \neq m$ and $\mathcal{B}$ outputs $s'_{j^\star}$. The message m' differs from m in at least one position. If it is the $j^\star$-th position (probability $1/k$) and if the signature is valid (probability ε) we have $f(s'_{j^\star}) = y_{i^\star,j^\star} = y^\star$.

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

Proof. $\mathcal{B}$ gets as input the description of f and $y^\star \in Y$.

- $\mathcal{B}$ picks as input an index $(i^\star, j^\star) \in \{0,1\} \times [\![1,k]\!]$
- $\mathcal{B}$ selects $2k-1$ random elements $x_{0,1}, \dots, \widehat{x_{i^\star,j^\star}}, \dots, x_{1,k} \in X$.
- $\mathcal{B}$ computes their images $y_{i,j} = f(x_{i,j}) = \mathsf{Eval}(x_{i,j})$ for $(i,j) \in \{0,1\} \times [\![1,k]\!] \setminus \{(i^\star, j^\star)\}$.
- $\mathcal{B}$ sets $y_{i^\star,j^\star} = y^\star$
- $\mathcal{B}$ executes $\mathcal{A}$ on the verification key $(y_{0,1}, y_{1,1}, \dots, y_{0,k}, y_{1,k})$
- At some point $\mathcal{A}$ query **one** message $m = m_1 \dots m_k$ to the signature oracle
  - If $m_{j^\star} = i^\star$ then $\mathcal{B}$ aborts the simulation (probability $1/2$),
  - otherwise $\mathcal{B}$ outputs a valid signature on m thanks to its knowledge of $x_{0,1}, \dots, \widehat{x_{i^\star,j^\star}}, \dots, x_{1,k}$.
- Eventually, $\mathcal{A}$ outputs a signature s' on a message $m' \neq m$ and $\mathcal{B}$ outputs $s'_{j^\star}$. The message m' differs from m in at least one position. If it is the $j^\star$-th position (probability $1/k$) and if the signature is valid (probability ε) we have $f(s'_{j^\star}) = y_{i^\star,j^\star} = y^\star$.

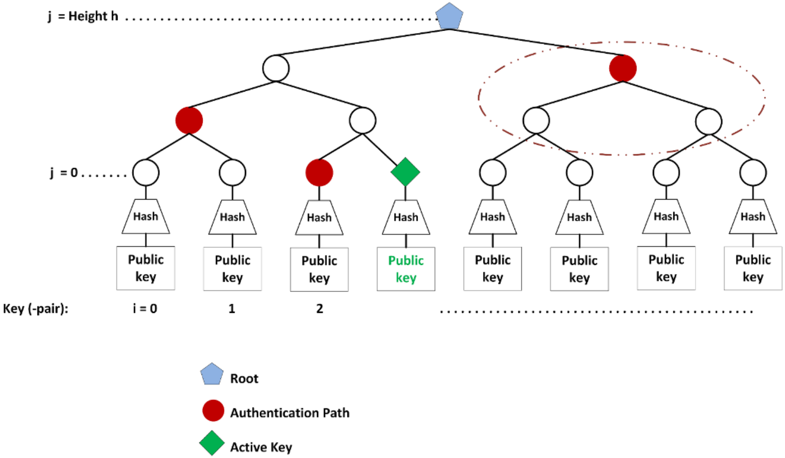
$\square$

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 (generic) groups of of prime order q of n-bits, solving the discrete logarithm problem requires $2^{n/2}$ operations.
- For a 128-bit security level, we need to have a group order q of (at least) 256 bits and for an ideal $\mathbb{G}$ (an elliptic curve?), elements in $\mathbb{G}$ can be represented with 256 bits.
  The verification key is made of $256^2 = 65536$ bits and its generation requires 256 exponentiation in $\mathbb{G}$.
- The signature is made of k elements from $\mathbb{Z}_q$. The signature length is at least $256 \cdot k$ bits.
- Can sign only one message

Lamport's signatures: several messages



Groth's one-time signatures

Groth (2006)
Simulation-sound NIZK proofs for a practical language and constant size group signatures.
Advances in Cryptology - Asiacrypt 2006: pp. 444–459

Key generation: generate $vk = (X = g^x, Y = g^y, Z = g^z)$ where $x, y, z \xleftarrow{\$} \mathbb{Z}_p^*$

Sign: to sign $m \in \mathbb{Z}_p^*$, select $r \xleftarrow{\$} \mathbb{Z}_p^*$, compute $s = (1 - mx - yr)/z \in \mathbb{Z}_p^*$, and output $\sigma = (r, s)$.

Verify: given $\sigma \in (\mathbb{Z}_p^*)^2$, check

$$X^m Y^r Z^s = g.$$

Groth's one-time signatures

Theorem

If the discrete logarithm assumption holds in $\mathbb{G}$ then Groth's signature scheme is one-time EUF-CMA secure.

Proof idea: given a DL instance $(g, h) \in \mathbb{G}$, one sets $X = g^{a_1} h^{b_1}$, $Y = g^{a_2} h^{b_2}$, $Z = g^{a_3}$ where $a, b, c \xleftarrow{\$} \mathbb{Z}_p^*$. On signature query on m, one compute $r = -mb_1/b_2 \bmod p$ and $s = (1 - ma_1 - r_2)/a_3 \bmod p$.

Thanks to the adversary's forgery, one can retrieve the discrete logarithm of h in base g by solving a linear system. □

Outline

Preliminaries: Injective function into the prime integers

- Before any description, we will assume the existence of a function Ψ with the following properties:
  - given k, Ψ maps any string from $\{0,1\}^k$ into the **set of the prime integers**,
  - Ψ is also designed to be **easy to compute** and **injective**.

- The following is a natural candidate:

$$\Psi : \begin{array}{rcl} \{0,1\}^k & \longrightarrow & \text{Primes} \\ m & \longmapsto & \texttt{nextprime}(m \cdot 2^\kappa) \end{array}$$

where κ is suitably chosen to guarantee the existence of a prime in any set $[m \cdot 2^\kappa, (m+1) \cdot 2^\kappa[$, for $m < 2^k$.

- Note that the deterministic property of nextprime is not mandatory, one just needs it to be **injective**.

Gennaro-Halevi-Rabin signatures

Gennaro-Halevi-Rabin (GHR, E'99)

1. Generate a safe RSA modulus $n = pq$ with $p = 2p' + 1$, $q = 2q' + 1$.
   Randomly select $s \in \mathbb{Z}_n^*$.
   Let $\Psi : \{0,1\}^\ell \mapsto \text{Primes} \geq 3$ be as above
   Publish (n, s). Keep (p, q) private.
2. To sign $m \in \{0,1\}^\ell$, compute $\sigma = s^{1/\Psi(m)} \mod n$.
3. Given (m, σ), check whether $\sigma^{\Psi(m)} = s \mod n$.

"Under the Strong RSA assumption, the GHR signature scheme is existentially unforgeable under an adaptive chosen message attack in the standard model".

Gennaro-Halevi-Rabin signatures

Gennaro-Halevi-Rabin (GHR, E'99)

1. Generate a safe RSA modulus $n = pq$ with $p = 2p' + 1$, $q = 2q' + 1$.
   Randomly select $s \in \mathbb{Z}_n^*$.
   Let $\Psi : \{0,1\}^\ell \mapsto$ Primes ≥ 3 be as above
   Publish (n, s). Keep (p, q) private.
2. To sign $m \in \{0,1\}^\ell$, compute $\sigma = s^{1/\Psi(m)} \mod n$.
3. Given (m, σ), check whether $\sigma^{\Psi(m)} = s \mod n$.

> *"Under the Strong RSA assumption, the GHR signature scheme is existentially unforgeable under an adaptive chosen message attack in the standard model".*

Gennaro-Halevi-Rabin signatures

Gennaro-Halevi-Rabin (GHR, E'99)

1. Generate a safe RSA modulus $n = pq$ with $p = 2p' + 1$, $q = 2q' + 1$.
   Randomly select $s \in \mathbb{Z}_n^*$.
   Let $\Psi : \{0,1\}^\ell \mapsto$ Primes ≥ 3 be as above
   Publish (n, s). Keep (p, q) private.
2. To sign $m \in \{0,1\}^\ell$, compute $\sigma = s^{1/\Psi(m)} \mod n$.
3. Given (m, σ), check whether $\sigma^{\Psi(m)} = s \mod n$.

"Under the Strong RSA assumption, the GHR signature scheme is existentially unforgeable under an adaptive chosen message attack in the standard model".

Gennaro-Halevi-Rabin signatures

Gennaro-Halevi-Rabin (GHR, E'99)

1. Generate a safe RSA modulus $n = pq$ with $p = 2p' + 1$, $q = 2q' + 1$.
   Randomly select $s \in \mathbb{Z}_n^*$.
   Let $\Psi : \{0,1\}^\ell \mapsto$ Primes ≥ 3 be as above
   Publish (n, s). Keep (p, q) private.
2. To sign $m \in \{0,1\}^\ell$, compute $\sigma = s^{1/\Psi(m)} \mod n$.
3. Given (m, σ), check whether $\sigma^{\Psi(m)} = s \mod n$.

"Under the Strong RSA assumption, the GHR signature scheme is existentially unforgeable under an adaptive chosen message attack in the standard model".

The Flexible RSA Problem

Flexible RSA Problem (Barić, Pfitzmann, E'97): let $n = pq$ be an RSA modulus and $z \in \mathbb{Z}_n^*$. Find x and y such that

$$x^y = z \mod n$$

with $(x, y) \neq (z, 1)$.

An algorithm $\mathcal{R}$ is said to $(\tau_{\mathcal{R}}, \varepsilon_{\mathcal{R}})$-solve the FRSA problem if in at most $\tau_{\mathcal{R}}$ operations,

$$\Pr\left[n \leftarrow RSA(1^{\mathrm{k}}), z \leftarrow \mathbb{Z}_n^*, (x, y) \leftarrow \mathcal{R}(n, z), x^y = z \bmod n\right] \geq \varepsilon_{\mathcal{R}} ,$$

where the probability is taken over the distribution of (n, z) and over $\mathcal{R}$'s random tapes.

Strong RSA Assumption: for any $(\tau_{\mathcal{R}}, \varepsilon_{\mathcal{R}})$-solver,

$$\tau_{\mathcal{R}} \leq \mathrm{poly}(\mathrm{k}) \quad \Rightarrow \quad \varepsilon_{\mathcal{R}} = \mathrm{negl}(\mathrm{k}) .$$

GHR signatures, short message variant

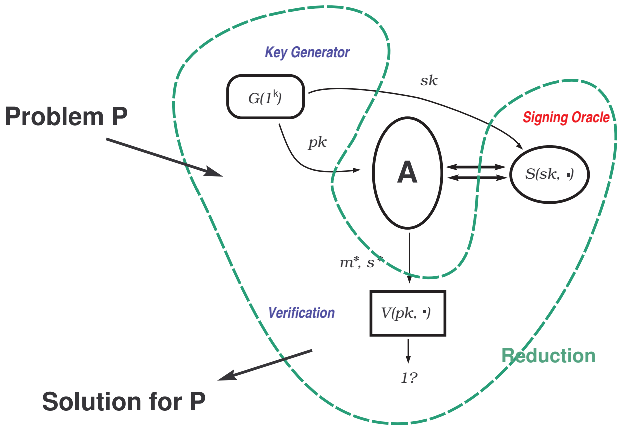
Gennaro-Halevi-Rabin (GHR, E'99), short message variant (GHR-s).

1. Generate a safe RSA modulus $n = pq$ with $p = 2p' + 1$, $q = 2q' + 1$.
   Randomly select $s \in \mathbb{Z}_n^*$.
   Let $\Psi : \{0,1\}^\ell \mapsto$ Primes ≥ 3 be as above ($\ell = 30$).
   Publish (n, s). Keep (p, q) private.

2. To sign $m \in \{0,1\}^\ell$, compute $\sigma = s^{1/\Psi(m)} \mod n$.

3. Given (m, σ), check whether $\sigma^{\Psi(m)} = s \mod n$.

Simulating the Attacker's Environment



Proving FRSA $\Leftarrow$ EUF-CMA(GHRs)

Our reduction $\mathcal{R}$ will behave as follows.

- $\mathcal{R}$ is given $n \leftarrow RSA(1^{\mathsf{k}})$ and $z \leftarrow \mathbb{Z}_n^*$, as well as an attacker $\mathcal{A}$ that $(q, \tau_{\mathcal{A}}, \varepsilon_{\mathcal{A}})$-solves EUF-CMA(GHRs),
- $\mathcal{R}$ simulates G and transmits vk to $\mathcal{A}$,
- $\mathcal{R}$ receives signature queries from $\mathcal{A}$: she will have to simulate a signing oracle wrt vk at most q times,
- $\mathcal{A}$ outputs a forgery (m, σ) for GHRs with probability $\varepsilon_{\mathcal{A}}$,
- $\mathcal{R}$ outputs non-trivial (x, y) such that $x^y = z \bmod n$.

$\mathcal{R}$ will provide a perfect simulation and $(\tau_{\mathcal{R}}, \varepsilon_{\mathcal{R}})$-solve FRSA with

$$\varepsilon_{\mathcal{R}} \geq \frac{\varepsilon_{\mathcal{A}}}{2^{\ell}} \quad \text{and} \quad \tau_{\mathcal{R}} \leq \tau_{\mathcal{A}} + \mathrm{poly}(2^{\ell}, \mathsf{k}) \; .$$

Simulation of Oracles

Simulation of G; for each and every message $m_i \in \{0,1\}^\ell$, compute $\Psi(m_i)$. Set $E = \prod \Psi(m_i)$, and select $Z \leftarrow \mathbb{Z}_n^*$ uniformly at random. Compute $s = z^E \mod n$ and send the GHRs verification key (n, s) to $\mathcal{A}$.

Since $n \leftarrow RSA(1^k)$ (external to $\mathcal{R}$) and $z \leftarrow \mathbb{Z}_n^*$ are random choices, and $x \mapsto x^E$ is one-to-one, (n, s) is perfectly indistinguishable from a random GHRs public key ($n \leftarrow RSA(1^k), s \leftarrow \mathbb{Z}_n^*$).

Simulation of S; when $\mathcal{A}$ requests the signature of some m_i, send $\sigma_i = z^{E/\Psi(m_i)} \mod n$.

Knowing Z and E, it is easy to extract a $\Psi(m_i)$-th root of s for any m_i. $\mathcal{A}$'s queries can be answered with perfectly valid signatures.

Simulation of Oracles

Simulation of G; for each and every message $m_i \in \{0,1\}^\ell$, compute $\Psi(m_i)$. Set $E = \prod \Psi(m_i)$, and select $Z \leftarrow \mathbb{Z}_n^*$ uniformly at random. Compute $s = z^E \mod n$ and send the GHRs verification key (n, s) to $\mathcal{A}$.

Since $n \leftarrow RSA(1^k)$ (external to $\mathcal{R}$) and $z \leftarrow \mathbb{Z}_n^*$ are random choices, and $x \mapsto x^E$ is one-to-one, (n, s) is perfectly indistinguishable from a random GHRs public key $(n \leftarrow RSA(1^k), s \leftarrow \mathbb{Z}_n^*)$.

Simulation of S; when $\mathcal{A}$ requests the signature of some m_i, send $\sigma_i = z^{E/\Psi(m_i)} \mod n$.

Knowing Z and E, it is easy to extract a $\Psi(m_i)$-th root of s for any m_i. $\mathcal{A}$'s queries can be answered with perfectly valid signatures.

Simulation of Oracles (Cont'd)

Simulation of V; trivial.

The simulation of the attacker's environment is perfect. So $\Pr[\mathcal{A} \text{ forges}] \geq \varepsilon_{\mathcal{A}}$.

So what? How will $\mathcal{R}$ answer the external query (n, z)?

Indeed, the forgery output by $\mathcal{A}$ with probability $\varepsilon_{\mathcal{A}}$ will be $s = z^{E/\Psi(m)}$. $\mathcal{R}$ could have computed that forgery by itself!

$\mathcal{R}$ must link her simulation of $\mathcal{A}$'s environment to its own query z (without destroying its perfectness too much).

Besides, the forgery must help $\mathcal{R}$ to get a good response (x, y).

Simulation of Oracles (Cont'd)

Simulation of V; trivial.

The simulation of the attacker's environment is perfect. So $\Pr[\mathcal{A} \text{ forges}] \geq \varepsilon_{\mathcal{A}}$.

So what? How will $\mathcal{R}$ answer the external query (n, z)?

Indeed, the forgery output by $\mathcal{A}$ with probability $\varepsilon_{\mathcal{A}}$ will be $s = z^{E/\Psi(m)}$. $\mathcal{R}$ could have computed that forgery by itself!

$\mathcal{R}$ must link her simulation of $\mathcal{A}$'s environment to its own query z (without destroying its perfectness too much).

Besides, the forgery must help $\mathcal{R}$ to get a good response (x, y).

Simulation of Oracles

Simulation of G;

- choose $i \in 1, \ldots 2^{\ell}$ uniformly at random
- for each $m_j \in \{0,1\}^{\ell}$, compute $\Psi(m_j)$. Set $E = \prod_{j \neq i} \Psi(m_j)$
- set $s = z^E \mod n$ and send the GHRs verification key (n, s) to $\mathcal{A}$.

Since $n \leftarrow RSA(1^k)$ and $z \leftarrow \mathbb{Z}_n^*$ are random choices (both external to $\mathcal{R}$), and $x \mapsto x^E$ is one-to-one, (n, s) is perfectly indistinguishable from a random GHRs verification key $(n \leftarrow RSA(1^k), s \leftarrow \mathbb{Z}_n^*)$.

That simulation of G is also a perfect one.

Simulation of Oracles

Simulation of G;

- choose $i \in 1, \ldots 2^\ell$ uniformly at random
- for each $m_j \in \{0,1\}^\ell$, compute $\Psi(m_j)$. Set $E = \prod_{j \neq i} \Psi(m_j)$
- set $s = z^E \mod n$ and send the GHRs verification key (n, s) to $\mathcal{A}$.

> *Since $n \leftarrow RSA(1^{\mathrm{k}})$ and $z \leftarrow \mathbb{Z}_n^*$ are random choices (both external to $\mathcal{R}$), and $x \mapsto x^E$ is one-to-one, (n, s) is perfectly indistinguishable from a random GHRs verification key $(n \leftarrow RSA(1^{\mathrm{k}}), s \leftarrow \mathbb{Z}_n^*)$.*

That simulation of G is also a perfect one.

Simulation of Oracles

Simulation of S; when $\mathcal{A}$ requests the signature of some m_j,

- if $j \neq i$, send $\sigma_j = z^{E/\Psi(m_j)} \mod n$
- if $j = i$, abort the experiment.

$\mathcal{A}$'s queries can be answered with perfectly valid signatures except when the query is m_i.

Since i is chosen in $[1, 2^\ell]$ independently from the attacker's view,

$$\Pr[\text{perfect simulation}] = \Pr[m_i \notin \texttt{Queries}(\mathcal{A}, S)] \geq \frac{2^\ell - q}{2^\ell} = 1 - \frac{q}{2^\ell} .$$

Simulation of Oracles

Simulation of S; when $\mathcal{A}$ requests the signature of some m_j,

- if $j \neq i$, send $\sigma_j = z^{E/\Psi(m_j)} \mod n$
- if $j = i$, abort the experiment.

$\mathcal{A}$'s queries can be answered with perfectly valid signatures except when the query is m_i.

Since i is chosen in $[1, 2^\ell]$ independently from the attacker's view,

$$\Pr\left[\text{perfect simulation}\right] = \Pr\left[m_i \notin \texttt{Queries}(\mathcal{A}, S)\right] \geq \frac{2^\ell - q}{2^\ell} = 1 - \frac{q}{2^\ell} \ .$$

Simulation of Oracles

How does that help? Assume that at the end of the game, $\mathcal{A}$ outputs (m_i, σ) as a forgery. Then,

$$\sigma^{\Psi(m_i)} = s = z^E \pmod{n}$$

But $\Psi(m_i)$ and E are coprime so $\mathcal{R}$ easily computes a and b with $a \cdot \Psi(m_i) + b \cdot E = 1$. Finally

$$z = z^{a\Psi(m_i)} \cdot z^{bE} = z^{a\Psi(m_i)} \cdot \sigma^{b\Psi(m_i)} = \left(z^a \sigma^b\right)^{\Psi(m_i)} \pmod{n}.$$

$\mathcal{R}$ then sets $x = z^a \sigma^b$ and $y = \Psi(m_i)$ and outputs a genuine solution (x, y).

Simulation of Oracles

How does that help? Assume that at the end of the game, $\mathcal{A}$ outputs (m_i, σ) as a forgery. Then,

$$\sigma^{\Psi(m_i)} = s = z^E \pmod{n}$$

But $\Psi(m_i)$ and E are coprime so $\mathcal{R}$ easily computes a and b with $a \cdot \Psi(m_i) + b \cdot E = 1$. Finally

$$z = z^{a\Psi(m_i)} \cdot z^{bE} = z^{a\Psi(m_i)} \cdot \sigma^{b\Psi(m_i)} = \left(z^a \sigma^b\right)^{\Psi(m_i)} \pmod{n}.$$

$\mathcal{R}$ then sets $x = z^a \sigma^b$ and $y = \Psi(m_i)$ and outputs a genuine solution (x, y).

Putting It All Together

- $\mathcal{R}$ perfectly simulates the scheme's oracles with probability $1 - q/2^\ell$,
- $\mathcal{A}$ then outputs (m, σ) with probability at least $\varepsilon_\mathcal{A}$ after time $\tau_\mathcal{A}$,
- since i is independent from $\mathcal{A}$, the event $m = m_i$ occurs with probability $1/(2^\ell - q)$,
- $\mathcal{R}$ then outputs a solution (x, y) for $\mathsf{FRSA}[n, z]$ with probability one.

Summing up, $\mathcal{R}$ succeeds with probability

$$\varepsilon_\mathcal{R} \geq \left(1 - \frac{q}{2^\ell}\right) \cdot \varepsilon_\mathcal{A} \cdot \frac{1}{2^\ell - q} = \frac{\varepsilon_\mathcal{A}}{2^\ell}.$$

Putting It All Together

- $\mathcal{R}$ simulates G in time at most

$$2^{\ell}\tau_{\Psi} + \mathcal{O}\left(2^{\ell}\mathsf{k}^2\right) + \mathcal{O}\left(\mathsf{k}^3\right) = \mathsf{poly}(2^{\ell}, \mathsf{k}) \ ,$$

- $\mathcal{R}$ simulates S in time $q \cdot \mathsf{poly}(2^{\ell}, \mathsf{k})$,
- $\mathcal{R}$ simulates V in time $\mathsf{poly}(2^{\ell}, \mathsf{k})$,
- $\mathcal{R}$ computes (x, y) in time $\mathsf{poly}(\mathsf{k}^3)$,
- and $\mathcal{R}$ lets $\mathcal{A}$ run as long as necessary, which takes time $\tau_{\mathcal{A}}$.

Summing up, our reduction $\mathcal{R}$ runs in time

$$\tau_{\mathcal{R}} \leq \tau_{\mathcal{A}} + q \cdot \mathsf{poly}(2^{\ell}, \mathsf{k}).$$

Towards EUF-CMA Security for arbitrary messages

- In the previous proof, we did not use the fact that the messages are **short** but only the fact that the set of messages to be signed are known at the beginning of the simulation.

- Therefore, the same proof yields that GHR scheme is EUF-KMA secure for **arbitrary length messages**.

- Unfortunately, in this case the GHR signature scheme has also loose security reduction to the flexible-RSA problem

Can we make if EUF-CMA secure for arbitrary length messages ?
Can we have a tight security reduction ?

Outline

What Are Ideal Assumptions?

Providing reductions is rarely as easy as just seen. We often need to idealize our view of primitive objects in order to simplify the proof.

- ideal random hash functions $\rightrightarrows$ random oracle model,
- ideal symmetric encryption $\rightrightarrows$ ideal cipher model,
- ideal group $\rightrightarrows$ generic group model.

A reduction is easier between a given problem and a generic adversary!

Do people buy these proofs? NO: there exist theoretical schemes secure in the ROM which are insecure in the standard model; YES: it is a moral proof that spots design errors anyway...

Diffie and Hellman's First Step...

Trapdoor-based signatures:

- Alice generates and publishes some one-way trapdoor function E, while she keeps $D = E^{-1}$ private,
- to sign message m, Alice computes $s = D(m)$ and sends the pair (m, s) to Bob,
- to verify the signature, Bob computes $m' = E(s)$ and checks whether $m' = m$.

The intuition behind this system is that only Alice can compute $D(m)$ so forgery should be hard since function E is supposedly one-way.

- subject to **existential forgery**
- (for RSA) subject to **universal forgery under chosen-message attack**

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 message m, Alice computes $s = D_d(H(m))$ and sends the pair (m, s) to Bob,
- to verify the signature, Bob checks whether $H(m) = E_e(s)$.

This technique is called the Hash-then-Invert paradigm. H is now a part (subroutine) of the scheme.
It must map messages to elements of E's domain, say X.

What features of H are **sufficient** to prevent **all** attacks?

Proofs in the Random Oracle Model

- The Random Oracle Model:
  - the hash function is replaced by an oracle which outputs a random value for each new input.
  - no entity, scheme ingredient or adversary, can compute the hash function by itself, it must query the oracle.

- Proof in the Random Oracle Model
  - a proof in that model does not imply that the scheme is secure in the real world (Canetti, Goldreich and Halevi, STOC' 98).
  - widely believed to be an acceptable engineering principle to design provably secure schemes.

Signature Schemes in the ROM

In the ROM, the hash function(s) included into the system description are externalized as random oracles (wlog, there is only one random oracle H).

- whenever $S^H(sk, m)$ needs $H(\omega)$, ω is sent to the oracle H and some value $H(\omega)$ is returned (black-box subroutine)
- works the same way for $V^H(pk, m, s)$
- if H is called twice with input ω, the same response $H(\omega)$ is returned
- before calling H with query ω, $H(\omega)$ can be any value in the output space $\mathcal{H}$ of H, so it's guessable with probability $1/|\mathcal{H}|$ whatever past queries happen to be.

How to View Random Oracles

The random oracle H is selected uniformly at random among all functions $\{0,1\}^* \mapsto \mathcal{H}$ for each new experiment involving the scheme.
Therefore, the values returned by H are independent and decorrelated from the keys (pk, sk).

The best way of looking at H is under the form of a coin tosser with memory, that defines itself as time goes on:

- at the beginning of the experiment, H is completely undefined
- when H is called with query ω for the first time, H selects $H(\omega)$ uniformly at random over $\mathcal{H}$ and memorizes the pair $(\omega, H(\omega))$ in a database
- for each query ω, H first searches for (ω, h) in its database. If found, h is returned.

CM Attacks in the ROM

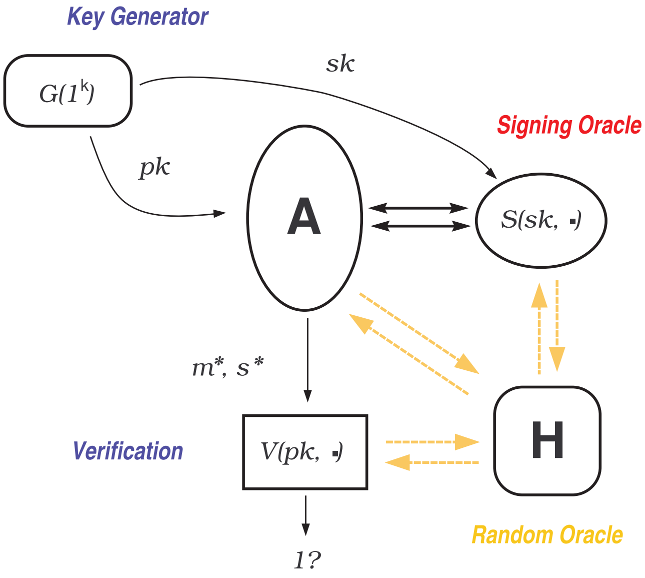
What happens to EUF-CMA in the Random Oracle Model?

A signature scheme is said to be $(q_H, q_{\mathrm{sig}}, \tau, \varepsilon)$-secure if for any adversary $\mathcal{A}$ with running time upper-bounded by τ,

$$\mathsf{Succ}^{\mathsf{EUF-CMA}}(\mathcal{A}) = \Pr\left[\begin{array}{l} (sk, vk) \leftarrow G(1^{\mathrm{k}}), \\ H \leftarrow \mathrm{Functions}\left(\{0,1\}* \mapsto \mathcal{H}\right), \\ (m^*, s^*) \leftarrow \mathcal{A}^{S^H(sk,\cdot),H}(pk), \\ V^H(pk, m^*, s^*) = 1 \end{array}\right] < \varepsilon \ ,$$

where the probability is taken over all random choices, including the one of H. The adversary has access to the signing oracle and the random oracle throughout the game, but at most q_{sig} and q_H times respectively.

EUF-CMA Attackers In the ROM



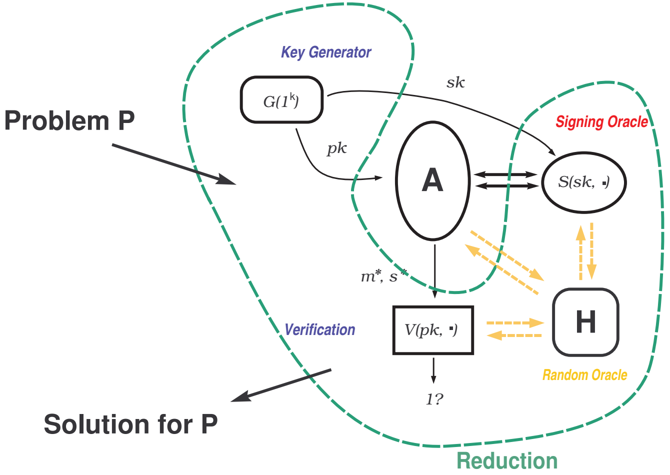
Reductions in the ROM

In the standard model, a reduction algorithm $\mathcal{R}$ has to simulate the attacker's environment *i.e.* provide access to oracles G, S and V with the same distributions of inputs and outputs.

> *In the Random Oracle Model, $\mathcal{R}$ must simulate oracles G, S^H and V^H* **as well as oracle H itself**. *Distributions have to be identical (or indistinguishable) from the ones the attacker expects.*

Most of times, this is easy...

Simulating the Attacker's Environment



Example: RSA $\overset{\text{ro}}{\Longleftarrow}$ EUF-CMA(FDH-RSA)

Full Domain Hash-RSA was proven secure in the Random Oracle Model in 1996 by Bellare and Rogaway:

$$m \longrightarrow H(m) \longrightarrow s = H(m)^d \mod n\,.$$

The hash function $H : \{0,1\}^* \mapsto \mathbb{Z}_n^* = \mathcal{H}$ has maximal output size here.

FDH-RSA is existentially unforgeable under chosen-message attacks in the Random Oracle Model, assuming that inverting RSA is hard.

Exercise: build a reduction from RSA to EUF-CMA(FDH-RSA) in the Random Oracle Model.

The RSA Problem

Rivest-Shamir-Adleman (1978): let $n = pq$ be an RSA modulus, $e \in \mathbb{Z}^*_{\phi(n)}$ and $y \in \mathbb{Z}^*_n$. Find x such that

$$x^e = y \mod n \ .$$

An algorithm $\mathcal{R}$ is said to $(\tau_{\mathcal{R}}, \varepsilon_{\mathcal{R}})$-solve RSA if in at most $\tau_{\mathcal{R}}$ operations,

$$\Pr\left[(n, e) \leftarrow RSA(1^{\mathrm{k}}), y \leftarrow \mathbb{Z}^*_n, x \leftarrow \mathcal{R}(n, e, y), x^e = y \bmod n\right] \geq \varepsilon_{\mathcal{R}} \ ,$$

where the probability is taken over the distribution of (n, e, y) and over $\mathcal{R}$'s random tapes.

The RSA Assumption: for any $(\tau_{\mathcal{R}}, \varepsilon_{\mathcal{R}})$-solver,

$$\tau_{\mathcal{R}} \leq \mathrm{poly}(\mathrm{k}) \quad \Rrightarrow \quad \varepsilon_{\mathcal{R}} = \mathrm{negl}(\mathrm{k}) \ .$$

Proving RSA $\stackrel{\mathrm{ro}}{\Leftarrow}$ EUF-CMA(FDH-RSA)

Every reduction $\mathcal{R}$ behaves as follows.

- $\mathcal{R}$ is given $(n, e) \leftarrow RSA(1^{\mathrm{k}})$ and $y \leftarrow \mathbb{Z}_n^*$, as well as an attacker $\mathcal{A}$ that $(q_H, q_{\mathrm{sig}}, \tau_{\mathcal{A}}, \varepsilon_{\mathcal{A}})$-solves EUF-CMA(FDH-RSA),
- $\mathcal{R}$ simulates G and transmits some verification key vk to $\mathcal{A}$,
- $\mathcal{R}$ receives signature queries from $\mathcal{A}$: it will have to simulate a signing oracle wrt vk at most q_{sig} times,
- $\mathcal{R}$ receives queries for H from $\mathcal{A}$: it will have to simulate H at most q_H times,
- $\mathcal{A}$ outputs a forgery (m, s) for FDH-RSA
- $\mathcal{R}$ simulates a verification of the forgery which is valid with probability $\varepsilon_{\mathcal{A}}$,
- $\mathcal{R}$ outputs x such that $x^e = y \bmod n$.

Proving RSA $\Leftarrow^{\mathrm{ro}}$ EUF-CMA(FDH-RSA)

Every reduction $\mathcal{R}$ behaves as follows.

- $\mathcal{R}$ is given $(n, e) \leftarrow RSA(1^{\mathsf{k}})$ and $y \leftarrow \mathbb{Z}_n^*$, as well as an attacker $\mathcal{A}$ that $(q_H, q_{\mathrm{sig}}, \tau_{\mathcal{A}}, \varepsilon_{\mathcal{A}})$-solves EUF-CMA(FDH-RSA),
- $\mathcal{R}$ simulates G and transmits some verification key vk to $\mathcal{A}$,
- $\mathcal{R}$ receives signature queries from $\mathcal{A}$: it will have to simulate a signing oracle wrt vk at most q_{sig} times,
- $\mathcal{R}$ receives queries for H from $\mathcal{A}$: it will have to simulate H at most q_H times,
- $\mathcal{A}$ outputs a forgery (m, s) for FDH-RSA
- $\mathcal{R}$ simulates a verification of the forgery which is valid with probability $\varepsilon_{\mathcal{A}}$,
- $\mathcal{R}$ outputs x such that $x^e = y \bmod n$.

Proving RSA $\overset{\mathrm{ro}}{\Leftarrow}$ EUF-CMA(FDH-RSA)

Every reduction $\mathcal{R}$ behaves as follows.

- $\mathcal{R}$ is given $(n, e) \leftarrow RSA(1^{\mathsf{k}})$ and $y \leftarrow \mathbb{Z}_n^*$, as well as an attacker $\mathcal{A}$ that $(q_H, q_{\mathrm{sig}}, \tau_{\mathcal{A}}, \varepsilon_{\mathcal{A}})$-solves EUF-CMA(FDH-RSA),
- $\mathcal{R}$ simulates G and transmits some verification key vk to $\mathcal{A}$,
- $\mathcal{R}$ receives signature queries from $\mathcal{A}$: it will have to simulate a signing oracle wrt vk at most q_{sig} times,
- $\mathcal{R}$ receives queries for H from $\mathcal{A}$: it will have to simulate H at most q_H times,
- $\mathcal{A}$ outputs a forgery (m, s) for FDH-RSA
- $\mathcal{R}$ simulates a verification of the forgery which is valid with probability $\varepsilon_{\mathcal{A}}$,
- $\mathcal{R}$ outputs x such that $x^e = y \bmod n$.

Simulation of Oracles

Our reduction $\mathcal{R}$ will provide a perfect simulation and $(\tau_{\mathcal{R}}, \varepsilon_{\mathcal{R}})$-solve RSA with

$$\varepsilon_{\mathcal{R}} \geq \frac{\varepsilon_{\mathcal{A}}}{q_H + q_{\mathrm{sig}} + 1} \quad \text{and} \quad \tau_{\mathcal{R}} \leq \tau_{\mathcal{A}} + (q_H + q_{\mathrm{sig}} + 1) \cdot \mathrm{poly(k)} \; .$$

Simulation of G; The reduction $\mathcal{R}$

- chooses $i \in [1, \dots, q_H + q_{\mathrm{sig}} + 1]$ uniformly at random
- sets $j = 1$ and $\mathrm{Hist}\,[H] = \emptyset$
- sends the FDH-RSA verification key (n, e) to $\mathcal{A}$.

The simulation of G is perfect.

Simulation of Oracles

Simulation of H; when $\mathcal{A}$ queries H with message m,

- $\mathcal{R}$ checks in Hist $[H]$ if m was queried in the past. If $H(m)$ is already defined to a value h, the same value h is returned.
- if $j \neq i$, $\mathcal{R}$
  - picks r_j at random
  - defines and returns $H(m) = r_j^e \pmod{n}$ omitted) to $\mathcal{A}$
  - memorizes (m, r_j, r_j^e) in Hist $[H]$
  - increments $j = j + 1$
- if $j = i$, $\mathcal{R}$
  - defines and returns $H(m) = y$ to $\mathcal{A}$
  - memorizes $(m, \perp, y)$ in Hist $[H]$

The simulation of H is perfect because $y \leftarrow \mathbb{Z}_n^*$ and $r_j \leftarrow \mathbb{Z}_n^*$ are pairwise independent and uniformly distributed over $\mathcal{H} = \mathbb{Z}_n^*$.

Simulation of Oracles

Simulation of S^H; when $\mathcal{A}$ requests the signature of some message m, $\mathcal{R}$

- invokes its own simulation of H to compute $H(m)$
- searches for the unique (m, α, β) in $\mathrm{Hist}\,[H]$
- if $(\alpha, \beta) = (\perp, y)$, $\mathcal{R}$ aborts
- otherwise $(\alpha, \beta) = (r_j, r_j^e)$ for some j and $\mathcal{R}$ returns r_j.

$\mathcal{A}$'s queries can be answered with perfectly valid signatures unless $\mathcal{R}$ aborts.

Simulation of Oracles

Simulation of V^H; Given (m, s), $\mathcal{R}$

- invokes its own simulation of H to get $H(m)$
- outputs 1 if $H(m)^e = s \pmod{n}$ or 0 otherwise

Final Outcome: assume that at the end of the game, $\mathcal{A}$ outputs (m^*, s^*) as a forgery. Then,

- $\mathcal{R}$ simulates V^H to verify if (m^*, s^*) is a valid forgery,
- if (m^*, s^*) is invalid, $\mathcal{R}$ aborts
- if $H(m^*) \neq y$, $\mathcal{R}$ aborts
- $\mathcal{R}$ sets $x = s^*$
- $\mathcal{R}$ outputs x

Synthesis

- $\mathcal{R}$ perfectly simulates the scheme's oracles H and V^H
- at each simulation of S^H, $\mathcal{R}$ may abort with probability

  $$1/(q_H + q_{\text{sig}} + 1)$$

  because **the choice of i is independent from $\mathcal{A}$**
- $\mathcal{R}$ aborts before answering the q_{sig} queries to S^H with probability $\Pr[\mathcal{R} \text{ fails}] \leq q_{\text{sig}}/(q_H + q_{\text{sig}} + 1)$
- hence, under probability $1 - \Pr[\mathcal{R} \text{ fails}]$, $\mathcal{R}$ reaches the end of the game
- after time $\tau_{\mathcal{A}}$, $\mathcal{A}$ outputs (m^*, s^*)
- (m^*, s^*) is accepted as a valid forgery by V^H with probability at least $\varepsilon_{\mathcal{A}}$ by definition

Synthesis

- since the choice of i is independent from $\mathcal{A}$

$$\Pr\left[(m^*, \perp, y) \in \mathsf{Hist}\left[H\right]\right] = \frac{1}{q_H + 1}$$

- $\mathcal{R}$ then outputs the solution x of RSA$[n, e, y]$ with probability one.

Summing up, $\mathcal{R}$ succeeds with probability

$$\varepsilon_{\mathcal{R}} \geq \left(1 - \frac{q_{\mathrm{sig}}}{q_H + q_{\mathrm{sig}} + 1}\right) \cdot \varepsilon_{\mathcal{A}} \cdot \frac{1}{q_H + 1} = \frac{\varepsilon_{\mathcal{A}}}{q_H + q_{\mathrm{sig}} + 1}$$

and time bound

$$\tau_{\mathcal{R}} \leq \tau_{\mathcal{A}} + (q_H + q_{\mathrm{sig}} + 1)\, \mathcal{T}_{RSA} \, .$$

Security proof for FDH-RSA

- This security proof is from Bellare and Rogaway (E'96)
- From a forger that breaks FDH with probability ε in time $t = \tau$, we can invert RSA with probability $\varepsilon' = \varepsilon/(q_H + q_{\mathrm{sig}} + 1)$ in time τ' close to τ.
- Conversely, if we assume that it is impossible to invert RSA with probability greater than ε' in time τ', it is impossible to break FDH with probability greater than $\varepsilon = (q_H + q_{\mathrm{sig}} + 1) \cdot \varepsilon'$ in time τ close to τ'.

- The probability ε of breaking FDH can be much larger than ε', the probability of inverting RSA.

Tightness of the Reduction

$$\boxed{\varepsilon_{\mathcal{R}} \geq \frac{\varepsilon_{\mathcal{A}}}{q_H + q_{\mathrm{sig}} + 1} \text{ and } \tau_{\mathcal{R}} \leq \tau_{\mathcal{A}} + (q_H + q_{\mathrm{sig}} + 1)\,\mathcal{T}_{RSA}.}$$

- **Security bound:** 2^{112}
- **Hash queries:** 2^{80}
- **Signing queries:** 2^{30}

Break the scheme within time expected t, invert RSA within time

$$t' \leq (q_H + q_{\mathrm{sig}} + 1)(t + (q_H + q_{\mathrm{sig}})\mathcal{T}_{RSA}) \leq 2^{80} t + 2^{160} \mathcal{T}_{RSA}$$

- RSA 2048 bits $\longrightarrow 2^{214}$ (NFS : 2^{112}) ✖
- RSA 3072 bits $\longrightarrow 2^{215}$ (NFS : 2^{128}) ✖
- RSA 7680 bits $\longrightarrow 2^{217}$ (NFS : 2^{192}) ✖
- RSA 15360 bits $\longrightarrow 2^{219}$ (NFS : 2^{256}) ✔

Graph isomorphism

- In **graph theory**, an isomorphism of graphs G and H is a bijection between the vertex sets of G and H

$$f : V(G) \longrightarrow V(H)$$

  such that any two vertices u and v of G are adjacent in G if and only if $f(u)$ and $f(v)$ are adjacent in H.

- If an isomorphism exists between two graphs, then the graphs are called isomorphic.

- The computational problem of determining whether two finite graphs are isomorphic is referred to as the **graph isomorphism problem**.

- The graph isomorphism problem is a curiosity in computational complexity theory: not known to be in $\mathcal{P}$ nor $\mathcal{NP}$-complete.

Graph isomorphism

- In **graph theory**, an isomorphism of graphs G and H is a bijection between the vertex sets of G and H

$$f : V(G) \longrightarrow V(H)$$

  such that any two vertices u and v of G are adjacent in G if and only if $f(u)$ and $f(v)$ are adjacent in H.

- If an isomorphism exists between two graphs, then the graphs are called isomorphic.

- The computational problem of determining whether two finite graphs are isomorphic is referred to as the **graph isomorphism problem**.

- The graph isomorphism problem is a curiosity in computational complexity theory: not known to be in $\mathcal{P}$ nor $\mathcal{NP}$-complete.

Graph isomorphism

- In **graph theory**, an isomorphism of graphs G and H is a bijection between the vertex sets of G and H

$$f : V(G) \longrightarrow V(H)$$

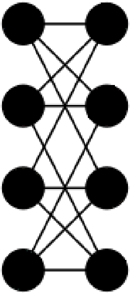
  such that any two vertices u and v of G are adjacent in G if and only if $f(u)$ and $f(v)$ are adjacent in H.

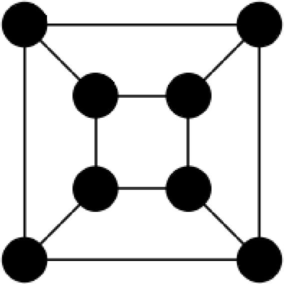
- If an isomorphism exists between two graphs, then the graphs are called isomorphic.

- The computational problem of determining whether two finite graphs are isomorphic is referred to as the **graph isomorphism problem**.

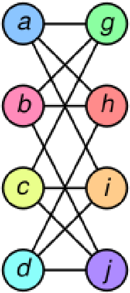
- The graph isomorphism problem is a curiosity in computational complexity theory: not known to be in $\mathcal{P}$ nor $\mathcal{NP}$-complete.

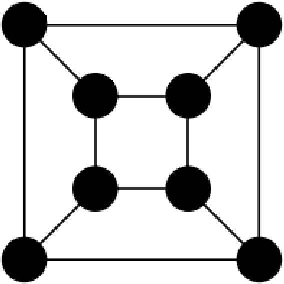
Graph isomorphism



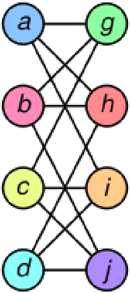


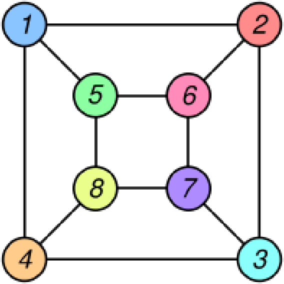
Graph isomorphism





Graph isomorphism





Zero-knowledge interactive proof

- a **zero-knowledge proof or zero-knowledge** protocol is an interactive method for one party to prove to another that a (usually mathematical) statement is true, without revealing anything other than the veracity of the statement.

- A zero-knowledge proof must satisfy three properties:
  1. **Completeness:** if the statement is true, the honest verifier (that is, one following the protocol properly) will be convinced of this fact by an honest prover.
  2. **Soundness:** if the statement is false, no cheating prover can convince the honest verifier that it is true, except with some small probability.
  3. **Zero-knowledge:** if the statement is true, no cheating verifier learns anything other than this fact.

- The first two of these are properties of more general interactive proof systems. The third is what makes the proof zero-knowledge.

Zero-knowledge interactive proof

- a **zero-knowledge proof or zero-knowledge** protocol is an interactive method for one party to prove to another that a (usually mathematical) statement is true, without revealing anything other than the veracity of the statement.
- A zero-knowledge proof must satisfy three properties:
  1. **Completeness:** if the statement is true, the honest verifier (that is, one following the protocol properly) will be convinced of this fact by an honest prover.
  2. **Soundness:** if the statement is false, no cheating prover can convince the honest verifier that it is true, except with some small probability.
  3. **Zero-knowledge:** if the statement is true, no cheating verifier learns anything other than this fact.
- The first two of these are properties of more general interactive proof systems. The third is what makes the proof zero-knowledge.

Zero-knowledge interactive proof

- a **zero-knowledge proof or zero-knowledge** protocol is an interactive method for one party to prove to another that a (usually mathematical) statement is true, without revealing anything other than the veracity of the statement.
- A zero-knowledge proof must satisfy three properties:
  1. **Completeness:** if the statement is true, the honest verifier (that is, one following the protocol properly) will be convinced of this fact by an honest prover.
  2. **Soundness:** if the statement is false, no cheating prover can convince the honest verifier that it is true, except with some small probability.
  3. **Zero-knowledge:** if the statement is true, no cheating verifier learns anything other than this fact.
- The first two of these are properties of more general interactive proof systems. The third is what makes the proof zero-knowledge.

Zero-knowledge interactive proof for Graph Isomorphism

Input: Two graphs G_0 and G_1 each having vertex set $\{1, \ldots, n\}$.
Alice **knows** $\sigma \in \mathfrak{S}_n$ an isomorphism from G_0 to G_1

1. Alice chooses a random permutation $\pi \in \mathfrak{S}_n$,
2. She computes H to be the image of G_0 under π and sends H to Bob,
3. Bob chooses randomly $b \in \{0, 1\}$ and sends it to Alice,
4. Alice sends $\rho = \pi \circ \sigma^b$ to Bob,
5. Bob checks if H is the image of G_b under ρ

Zero-knowledge interactive proof for Graph Isomorphism

Input: Two graphs G_0 and G_1 each having vertex set $\{1, \dots, n\}$.
Alice **knows** $\sigma \in \mathfrak{S}_n$ an isomorphism from G_0 to G_1

1. Alice chooses a random permutation $\pi \in \mathfrak{S}_n$,
2. She computes H to be the image of G_0 under π and sends H to Bob,
3. Bob chooses randomly $b \in \{0, 1\}$ and sends it to Alice,
4. Alice sends $\rho = \pi \circ \sigma^b$ to Bob,
5. Bob checks if H is the image of G_b under ρ

Zero-knowledge interactive proof for Graph Isomorphism

Input: Two graphs G_0 and G_1 each having vertex set $\{1, \ldots, n\}$.
Alice **knows** $\sigma \in \mathfrak{S}_n$ an isomorphism from G_0 to G_1

1. Alice chooses a random permutation $\pi \in \mathfrak{S}_n$,
2. She computes H to be the image of G_0 under π and sends H to Bob,
3. Bob chooses randomly $b \in \{0, 1\}$ and sends it to Alice,
4. Alice sends $\rho = \pi \circ \sigma^b$ to Bob,
5. Bob checks if H is the image of G_b under ρ

Zero-knowledge interactive proof for Graph Isomorphism

Input: Two graphs G_0 and G_1 each having vertex set $\{1, \dots, n\}$.
Alice **knows** $\sigma \in \mathfrak{S}_n$ an isomorphism from G_0 to G_1

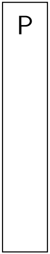
Repeat the following n times

1. Alice chooses a random permutation $\pi \in \mathfrak{S}_n$,
2. She computes H to be the image of G_0 under π and sends H to Bob,
3. Bob chooses randomly $b \in \{0, 1\}$ and sends it to Alice,
4. Alice sends $\rho = \pi \circ \sigma^b$ to Bob,
5. Bob checks if H is the image of G_b under ρ

Schnorr's ID Protocol (1989)

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.



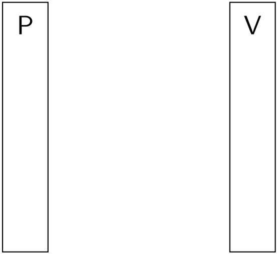
Scenario

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 (1989)

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

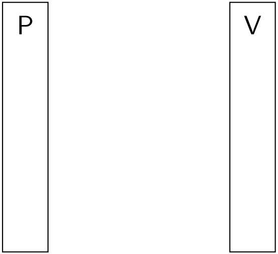
P sends $r = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
V sends $c \xleftarrow{\$} \mathbb{Z}_q$
P sends $s = k + cx \bmod q$
V checks whether $g^s \cdot y^{-c} = r$

Schnorr's ID Protocol (1989)

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

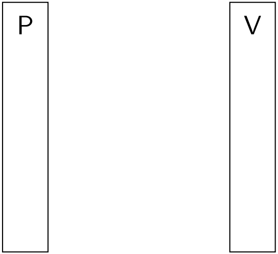
Schnorr's ID Protocol (1989)

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.

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

$y = g^x$



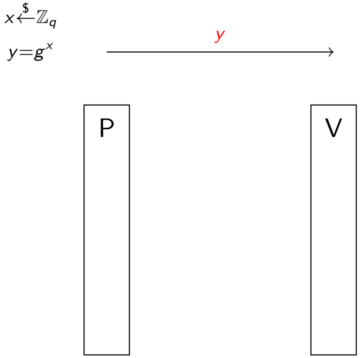
Scenario

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 (1989)

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.



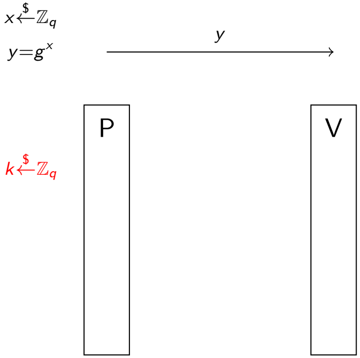
Scenario

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 (1989)

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 (1989)

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.

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

$y = g^x$ $\xrightarrow{\quad y \quad}$

P | V

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

$r = g^k$

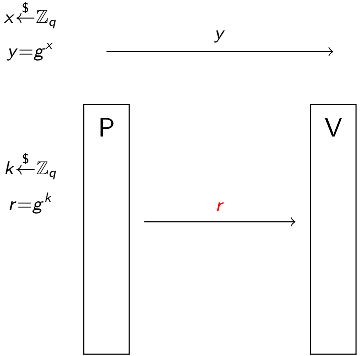
Scenario

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 (1989)

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.



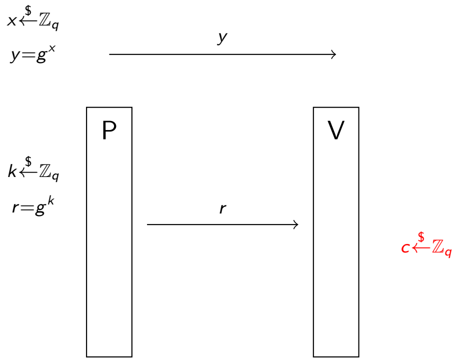
Scenario

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 (1989)

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.



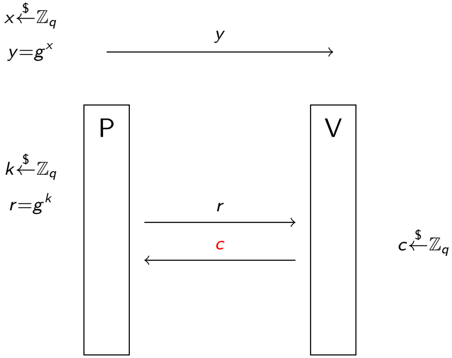
Scenario

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 (1989)

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.



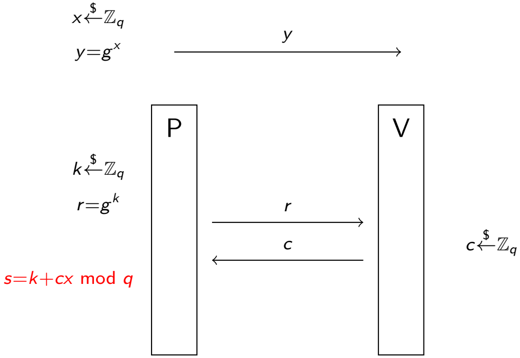
Scenario

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 (1989)

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.



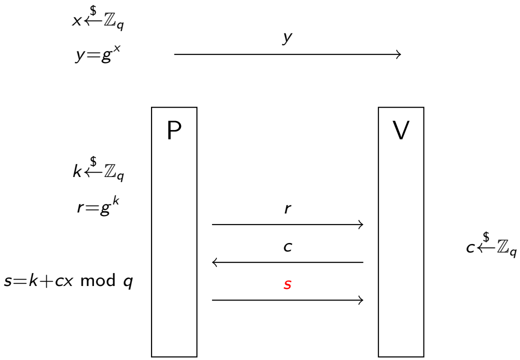
Scenario

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 (1989)

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

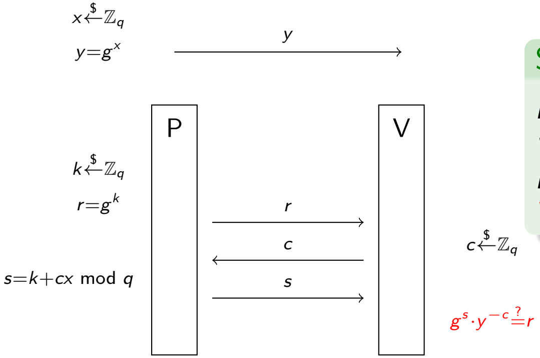
P sends $s = k + cx \bmod q$

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

Schnorr's ID Protocol (1989)

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

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.

- In such a 3-pass identification scheme, the 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.

- In such a 3-pass identification scheme, the 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 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 $e = H(t, m)$.

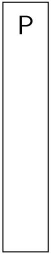
Intuitively, it should be hard for a forger to find m and a protocol transcript (t, e, s) for which it is true both that $e = H(t, m)$ and (t, e, s) is a valid transcript w.r.t. a given public key.

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

Schnorr Signatures (via the Fiat-Shamir Transform)

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

Schnorr's signature scheme $\S_i gma_H$ is a tuple of probabilistic algorithms $\S_i gma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ 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 $\sigma = (s, c)$

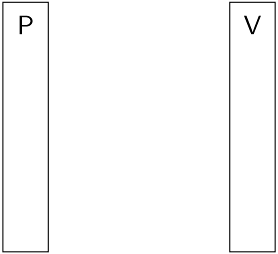
$\textsc{Ver}$

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 $\S_i gma_H$ is a tuple of probabilistic algorithms $\S_i gma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ 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 $\sigma = (s, c)$

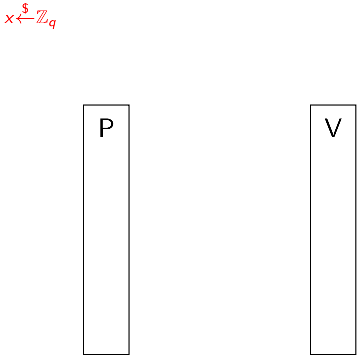
$\textsc{Ver}$

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 $\S_i gma_H$ is a tuple of probabilistic algorithms $\S_i gma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ 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 $H(m, g^s \cdot y^{-c}) = c$

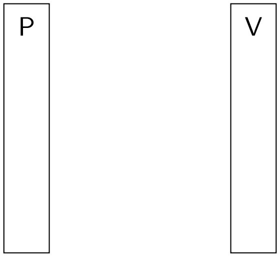
Schnorr Signatures (via the Fiat-Shamir Transform)

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

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

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

$y = g^x$



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 $\sigma = (s, c)$

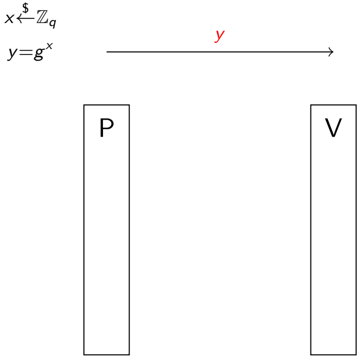
$\textsc{Ver}$

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 $\S_i gma_H$ is a tuple of probabilistic algorithms $\S_i gma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ 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 $\sigma = (s, c)$

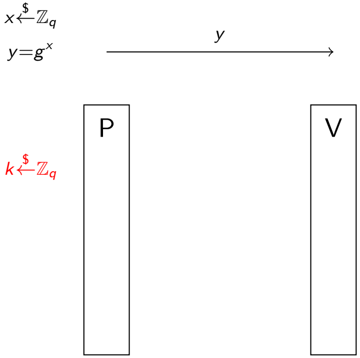
$\textsc{Ver}$

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 $\S_i gma_H$ is a tuple of probabilistic algorithms $\S_i gma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ 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 $\S_i gma_H$ is a tuple of probabilistic algorithms $\S_i gma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.

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

$y = g^x \qquad \xrightarrow{\quad y \quad}$

P   V

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

$r = g^k$

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 $\sigma = (s, c)$

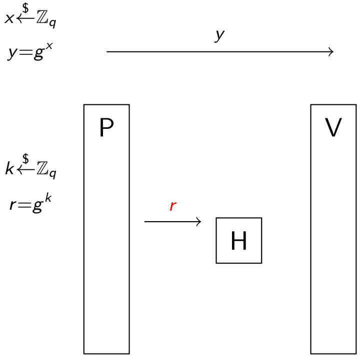
$\textsc{Ver}$

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 $\S_i gma_H$ is a tuple of probabilistic algorithms $\S_i gma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ 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 $\sigma = (s, c)$

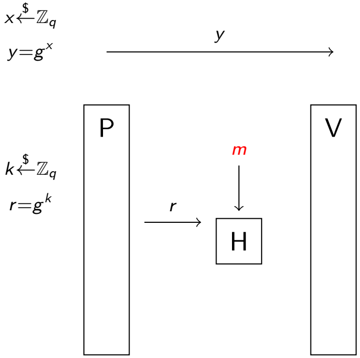
$\textsc{Ver}$

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 $\S_i gma_H$ is a tuple of probabilistic algorithms $\S_i gma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ 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 $\sigma = (s, c)$

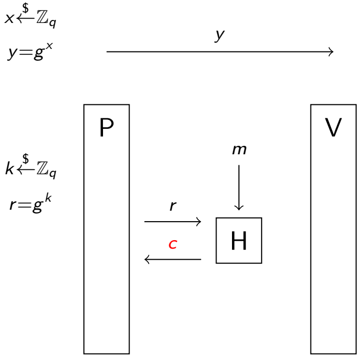
$\textsc{Ver}$

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 $\S_i gma_H$ is a tuple of probabilistic algorithms $\S_i gma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ 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 $\sigma = (s, c)$

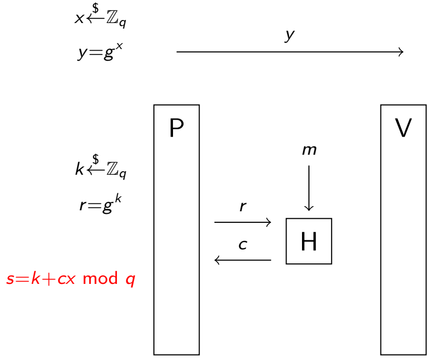
$\textsc{Ver}$

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 $\S_i gma_H$ is a tuple of probabilistic algorithms $\S_i gma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ 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 $\sigma = (s, c)$

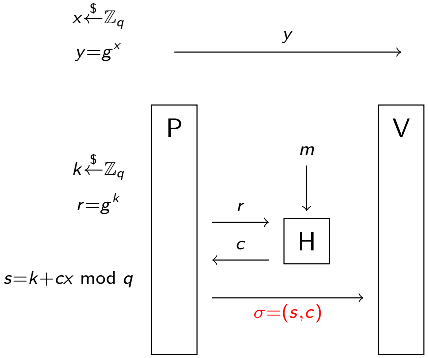
$\textsc{Ver}$

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 $\S_i gma_H$ is a tuple of probabilistic algorithms $\S_i gma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ 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 $\sigma = (s, c)$

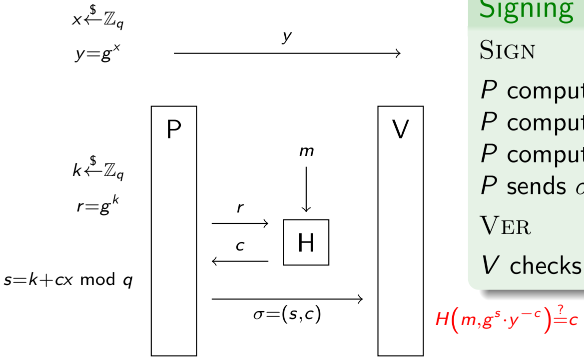
$\textsc{Ver}$

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 $\S_i gma_H$ is a tuple of probabilistic algorithms $\S_i gma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ 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 $\sigma = (s, c)$

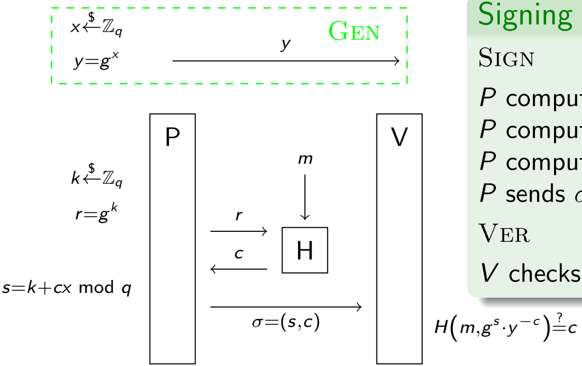
$\textsc{Ver}$

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 $\S_i gma_H$ is a tuple of probabilistic algorithms $\S_i gma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ 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 $\sigma = (s, c)$

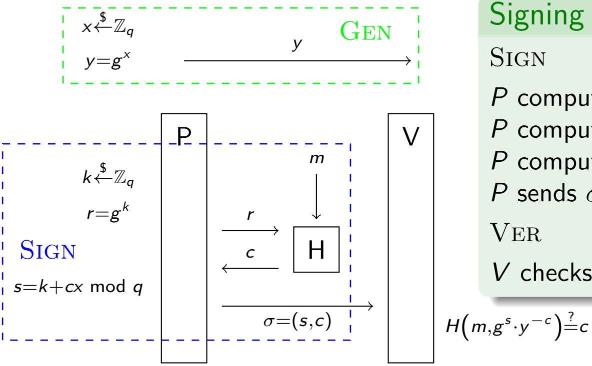
Ver

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 $\S_i gma_H$ is a tuple of probabilistic algorithms $\S_i gma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ 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 $\sigma = (s, c)$

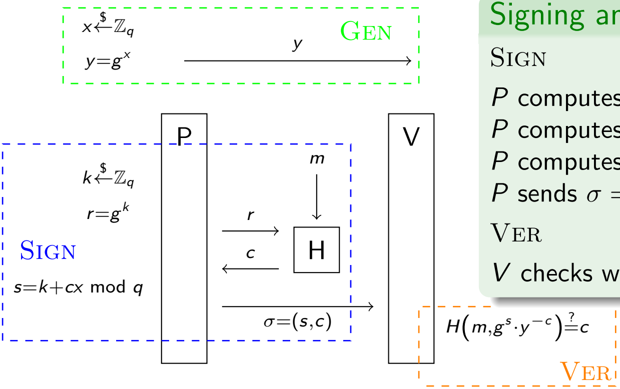
Ver

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 $\S_i gma_H$ is a tuple of probabilistic algorithms $\S_i gma_H = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ 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 $H(m, g^s \cdot y^{-c}) = c$

Security of Schnorr Signatures - Key Only Attacks

Theorem

If there exist a $(0, \tau, \varepsilon)$-EUF-CMA adversary in the ROM (with q_H queries to the RO) against Schnorr's signature scheme (in $\mathbb{G}$), then the discrete logarithm in $\mathbb{G}$ can be solved in expected time $O(\tau \cdot q_H/\varepsilon)$.

Proof Intuition

- run the adversary $\mathcal{A}$ several times in related executions
- the process "forks" at a certain point (modification of the RO)
- hope for two executions of $\mathcal{A}$ with forgery on the same message queried to the RO (but with different hash values)
  $\rightsquigarrow$ extract the discrete logarithm