Cryptography

Lecture 24

Arkady Yerukhimovich

November 20, 2024

Outline

Lecture 23 Review

- Defining digital signatures
- Applications of signatures
- RSA digital signature

Outline

One-Way Functions

One-Way Function

A function $f : \{0.1\}^* \to \{0,1\}^*$ is *one-way* if

One-Way Functions

One-Way Function

A function $f : \{0.1\}^* \rightarrow \{0,1\}^*$ is *one-way* if

- Easy to compute: Can compute $f(x)$ in poly time for all x

One-Way Functions

One-Way Function

A function $f : \{0.1\}^* \to \{0,1\}^*$ is *one-way* if

- Easy to compute: Can compute $f(x)$ in poly time for all x
- Hard to invert: $\forall$ PPT $\mathcal{A}$, $\Pr[\mathsf{Invert}_{\mathcal{A},f}(n) = 1] \leq \mathsf{negl}(n)$

One-Way Functions

One-Way Function

A function $f : \{0.1\}^* \to \{0,1\}^*$ is *one-way* if

- Easy to compute: Can compute $f(x)$ in poly time for all x
- Hard to invert: $\forall$ PPT $\mathcal{A}$, $\Pr[\mathsf{Invert}_{\mathcal{A},f}(n) = 1] \leq \mathsf{negl}(n)$

$\mathsf{Invert}_{\mathcal{A},f}(n)$ is the following game between $\mathcal{A}$ and a challenger:

$\mathsf{Invert}_{\mathcal{A},f}(n)$

One-Way Functions

One-Way Function

A function $f : \{0.1\}^* \to \{0,1\}^*$ is *one-way* if

- Easy to compute: Can compute $f(x)$ in poly time for all x
- Hard to invert: $\forall$ PPT $\mathcal{A}$, $\Pr[\mathsf{Invert}_{\mathcal{A},f}(n) = 1] \leq \mathsf{negl}(n)$

$\mathsf{Invert}_{\mathcal{A},f}(n)$ is the following game between $\mathcal{A}$ and a challenger:

$\mathsf{Invert}_{\mathcal{A},f}(n)$

- Challenger chooses $x \leftarrow \{0,1\}^n$ and computes $y = f(x)$

One-Way Functions

One-Way Function

A function $f : \{0.1\}^* \rightarrow \{0,1\}^*$ is *one-way* if

- Easy to compute: Can compute $f(x)$ in poly time for all x
- Hard to invert: $\forall$ PPT $\mathcal{A}$, $\Pr[\text{Invert}_{\mathcal{A},f}(n) = 1] \leq \text{negl}(n)$

$\text{Invert}_{\mathcal{A},f}(n)$ is the following game between $\mathcal{A}$ and a challenger:

$\text{Invert}_{\mathcal{A},f}(n)$

- Challenger chooses $x \leftarrow \{0,1\}^n$ and computes $y = f(x)$
- $\mathcal{A}$ gets y and outputs x'

One-Way Functions

One-Way Function

A function $f : \{0.1\}^* \to \{0,1\}^*$ is *one-way* if

- Easy to compute: Can compute $f(x)$ in poly time for all x
- Hard to invert: $\forall$ PPT $\mathcal{A}$, $\Pr[\mathsf{Invert}_{\mathcal{A},f}(n) = 1] \leq \mathsf{negl}(n)$

$\mathsf{Invert}_{\mathcal{A},f}(n)$ is the following game between $\mathcal{A}$ and a challenger:

$\mathsf{Invert}_{\mathcal{A},f}(n)$

- Challenger chooses $x \leftarrow \{0,1\}^n$ and computes $y = f(x)$
- $\mathcal{A}$ gets y and outputs x'
- We say that $\mathsf{Invert}_{\mathcal{A},f}(n) = 1$ (i.e., $\mathcal{A}$ wins) if $f(x') = y$.

One-Way Functions

One-Way Function

A function $f : \{0.1\}^* \rightarrow \{0,1\}^*$ is *one-way* if

- Easy to compute: Can compute $f(x)$ in poly time for all x
- Hard to invert: $\forall$ PPT $\mathcal{A}$, $\Pr[\mathsf{Invert}_{\mathcal{A},f}(n) = 1] \leq \mathsf{negl}(n)$

$\mathsf{Invert}_{\mathcal{A},f}(n)$ is the following game between $\mathcal{A}$ and a challenger:

$\mathsf{Invert}_{\mathcal{A},f}(n)$

- Challenger chooses $x \leftarrow \{0,1\}^n$ and computes $y = f(x)$
- $\mathcal{A}$ gets y and outputs x'
- We say that $\mathsf{Invert}_{\mathcal{A},f}(n) = 1$ (i.e., $\mathcal{A}$ wins) if $f(x') = y$.

Observations:

- $\mathcal{A}$ does not necessarily have to recover x to win

One-Way Functions

One-Way Function

A function $f : \{0.1\}^* \to \{0,1\}^*$ is *one-way* if

- Easy to compute: Can compute $f(x)$ in poly time for all x
- Hard to invert: $\forall$ PPT $\mathcal{A}$, $\Pr[\mathsf{Invert}_{\mathcal{A},f}(n) = 1] \leq \mathsf{negl}(n)$

$\mathsf{Invert}_{\mathcal{A},f}(n)$ is the following game between $\mathcal{A}$ and a challenger:

$\mathsf{Invert}_{\mathcal{A},f}(n)$

- Challenger chooses $x \leftarrow \{0,1\}^n$ and computes $y = f(x)$
- $\mathcal{A}$ gets y and outputs x'
- We say that $\mathsf{Invert}_{\mathcal{A},f}(n) = 1$ (i.e., $\mathcal{A}$ wins) if $f(x') = y$.

Observations:

- $\mathcal{A}$ does not necessarily have to recover x to win
- One-way functions are the most basic private-key primitives

One-Way Function

A function $f : \{0.1\}^* \to \{0,1\}^*$ is *one-way* if

- Easy to compute: Can compute $f(x)$ in poly time for all x
- Hard to invert: $\forall$ PPT $\mathcal{A}$, $\Pr[\mathsf{Invert}_{\mathcal{A},f}(n) = 1] \leq \mathsf{negl}(n)$

$\mathsf{Invert}_{\mathcal{A},f}(n)$ is the following game between $\mathcal{A}$ and a challenger:

$\mathsf{Invert}_{\mathcal{A},f}(n)$

- Challenger chooses $x \leftarrow \{0,1\}^n$ and computes $y = f(x)$
- $\mathcal{A}$ gets y and outputs x'
- We say that $\mathsf{Invert}_{\mathcal{A},f}(n) = 1$ (i.e., $\mathcal{A}$ wins) if $f(x') = y$.

Observations:

- $\mathcal{A}$ does not necessarily have to recover x to win
- One-way functions are the most basic private-key primitives
- We've seen many examples: CRHFs, PRG, RSA

One-Time Signature Scheme

Let Π be a digital signature scheme. Consider the following game between an adversary $\mathcal{A}$ and a challenger:

$\mathsf{SigForge}_{\mathcal{A},\Pi}(n)$

- Challenger runs $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ and gives pk to $\mathcal{A}$
- $\mathcal{A}$ gets pk and oracle access to $\mathsf{Sign}_{sk}(\cdot)$ and outputs (m, σ)
  - Let Q denote the set of $\mathsf{Sign}_{sk}(\cdot)$ queries made by $\mathcal{A}$
- We say that $\mathsf{SigForge}_{\mathcal{A},\Pi}(n) = 1$ (i.e., $\mathcal{A}$ wins) if $\mathsf{Verify}_{pk}(m, \sigma) = 1$ and $m \notin Q$.

One-Time Signature Scheme

Let Π be a digital signature scheme. Consider the following game between an adversary $\mathcal{A}$ and a challenger:

$\mathsf{SigForge}_{\mathcal{A},\Pi}(n)$

- Challenger runs $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ and gives pk to $\mathcal{A}$
- $\mathcal{A}$ gets pk and oracle access to $\mathsf{Sign}_{sk}(\cdot)$ and outputs (m, σ)
  - Let Q denote the set of $\mathsf{Sign}_{sk}(\cdot)$ queries made by $\mathcal{A}$
- We say that $\mathsf{SigForge}_{\mathcal{A},\Pi}(n) = 1$ (i.e., $\mathcal{A}$ wins) if $\mathsf{Verify}_{pk}(m, \sigma) = 1$ and $m \notin Q$.

One-time signature:

One-Time Signature Scheme

Let Π be a digital signature scheme. Consider the following game between an adversary $\mathcal{A}$ and a challenger:

$\text{SigForge}_{\mathcal{A},\Pi}(n)$

- Challenger runs $(pk, sk) \leftarrow \text{Gen}(1^n)$ and gives pk to $\mathcal{A}$
- $\mathcal{A}$ gets pk and oracle access to $\text{Sign}_{sk}(\cdot)$ and outputs (m, σ)
  - Let Q denote the set of $\text{Sign}_{sk}(\cdot)$ queries made by $\mathcal{A}$
- We say that $\text{SigForge}_{\mathcal{A},\Pi}(n) = 1$ (i.e., $\mathcal{A}$ wins) if $\text{Verify}_{pk}(m, \sigma) = 1$ and $m \notin Q$.

One-time signature:

- In a one-time signature, $\mathcal{A}$ can only query the $\text{Sign}_{sk}(\cdot)$ oracle once.

One-Time Signature Scheme

Let Π be a digital signature scheme. Consider the following game between an adversary $\mathcal{A}$ and a challenger:

$\mathsf{SigForge}_{\mathcal{A},\Pi}(n)$

- Challenger runs $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ and gives pk to $\mathcal{A}$
- $\mathcal{A}$ gets pk and oracle access to $\mathsf{Sign}_{sk}(\cdot)$ and outputs (m, σ)
  - Let Q denote the set of $\mathsf{Sign}_{sk}(\cdot)$ queries made by $\mathcal{A}$
- We say that $\mathsf{SigForge}_{\mathcal{A},\Pi}(n) = 1$ (i.e., $\mathcal{A}$ wins) if $\mathsf{Verify}_{pk}(m, \sigma) = 1$ and $m \notin Q$.

One-time signature:

- In a one-time signature, $\mathcal{A}$ can only query the $\mathsf{Sign}_{sk}(\cdot)$ oracle once.
- Informally: After seeing one signature, $\mathcal{A}$ can't forge another one

One-Time Signature Scheme

Let Π be a digital signature scheme. Consider the following game between an adversary $\mathcal{A}$ and a challenger:

$\mathsf{SigForge}_{\mathcal{A},\Pi}(n)$

- Challenger runs $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ and gives pk to $\mathcal{A}$
- $\mathcal{A}$ gets pk and oracle access to $\mathsf{Sign}_{sk}(\cdot)$ and outputs (m, σ)
  - Let Q denote the set of $\mathsf{Sign}_{sk}(\cdot)$ queries made by $\mathcal{A}$
- We say that $\mathsf{SigForge}_{\mathcal{A},\Pi}(n) = 1$ (i.e., $\mathcal{A}$ wins) if $\mathsf{Verify}_{pk}(m, \sigma) = 1$ and $m \notin Q$.

One-time signature:

- In a one-time signature, $\mathcal{A}$ can only query the $\mathsf{Sign}_{sk}(\cdot)$ oracle once.
- Informally: After seeing one signature, $\mathcal{A}$ can't forge another one
- This is not a very useful notion of security, but we will use it as a building block

Lamport One-Time Signature Scheme

We construct a signature scheme from any OWF as follows:

Lamport One-Time Signature Scheme

We construct a signature scheme from any OWF as follows:

- Gen(1^n): To sign ℓ-bit messages, for $i \in \{1, \ldots, \ell\}$:

Lamport One-Time Signature Scheme

We construct a signature scheme from any OWF as follows:

- $\mathsf{Gen}(1^n)$: To sign ℓ-bit messages, for $i \in \{1, \ldots, \ell\}$:
    - Choose $x_{i,0}, x_{i,1} \leftarrow \{0,1\}^n$

Lamport One-Time Signature Scheme

We construct a signature scheme from any OWF as follows:

- Gen(1^n): To sign ℓ-bit messages, for $i \in \{1, \ldots, \ell\}$:
  - Choose $x_{i,0}, x_{i,1} \leftarrow \{0,1\}^n$
  - Compute $y_{i,0} = f(x_{i,0})$ and $y_{i,1} = f(x_{i,1})$

Lamport One-Time Signature Scheme

We construct a signature scheme from any OWF as follows:

- $\mathsf{Gen}(1^n)$: To sign ℓ-bit messages, for $i \in \{1, \ldots, \ell\}$:
  - Choose $x_{i,0}, x_{i,1} \leftarrow \{0,1\}^n$
  - Compute $y_{i,0} = f(x_{i,0})$ and $y_{i,1} = f(x_{i,1})$
  - Output

$$pk = \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{\ell,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{\ell,1} \end{pmatrix} \qquad sk = \begin{pmatrix} x_{1,0} & x_{2,0} & \cdots & x_{\ell,0} \\ x_{1,1} & x_{2,1} & \cdots & x_{\ell,1} \end{pmatrix}$$

Lamport One-Time Signature Scheme

We construct a signature scheme from any OWF as follows:

- $\mathsf{Gen}(1^n)$: To sign ℓ-bit messages, for $i \in \{1, \ldots, \ell\}$:
  - Choose $x_{i,0}, x_{i,1} \leftarrow \{0,1\}^n$
  - Compute $y_{i,0} = f(x_{i,0})$ and $y_{i,1} = f(x_{i,1})$
  - Output

$$pk = \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{\ell,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{\ell,1} \end{pmatrix} \qquad sk = \begin{pmatrix} x_{1,0} & x_{2,0} & \cdots & x_{\ell,0} \\ x_{1,1} & x_{2,1} & \cdots & x_{\ell,1} \end{pmatrix}$$

- $\mathsf{Sign}_{sk}(m \in \{0,1\}^\ell)$: Output $\sigma = x_{1,m_1}, x_{2,m_2}, \ldots, x_{\ell,m_\ell}$

Lamport One-Time Signature Scheme

We construct a signature scheme from any OWF as follows:

- $\mathsf{Gen}(1^n)$: To sign ℓ-bit messages, for $i \in \{1, \ldots, \ell\}$:
  - Choose $x_{i,0}, x_{i,1} \leftarrow \{0,1\}^n$
  - Compute $y_{i,0} = f(x_{i,0})$ and $y_{i,1} = f(x_{i,1})$
  - Output

$$pk = \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{\ell,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{\ell,1} \end{pmatrix} \qquad sk = \begin{pmatrix} x_{1,0} & x_{2,0} & \cdots & x_{\ell,0} \\ x_{1,1} & x_{2,1} & \cdots & x_{\ell,1} \end{pmatrix}$$

- $\mathsf{Sign}_{sk}(m \in \{0,1\}^\ell)$: Output $\sigma = x_{1,m_1}, x_{2,m_2}, \ldots, x_{\ell,m_\ell}$
- $\mathsf{Verify}_{pk}(m, \sigma)$: Output 1 if $f(x_{i,m_i}) = y_{i,m_i}$ for all $i \in \{1, \ldots, \ell\}$

Lamport One-Time Signature Scheme

We construct a signature scheme from any OWF as follows:

- $\mathsf{Gen}(1^n)$: To sign ℓ-bit messages, for $i \in \{1, \ldots, \ell\}$:
  - Choose $x_{i,0}, x_{i,1} \leftarrow \{0,1\}^n$
  - Compute $y_{i,0} = f(x_{i,0})$ and $y_{i,1} = f(x_{i,1})$
  - Output

$$pk = \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{\ell,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{\ell,1} \end{pmatrix} \qquad sk = \begin{pmatrix} x_{1,0} & x_{2,0} & \cdots & x_{\ell,0} \\ x_{1,1} & x_{2,1} & \cdots & x_{\ell,1} \end{pmatrix}$$

- $\mathsf{Sign}_{sk}(m \in \{0,1\}^\ell)$: Output $\sigma = x_{1,m_1}, x_{2,m_2}, \ldots, x_{\ell,m_\ell}$
- $\mathsf{Verify}_{pk}(m, \sigma)$: Output 1 if $f(x_{i,m_i}) = y_{i,m_i}$ for all $i \in \{1, \ldots, \ell\}$

Security Intuition

Lamport One-Time Signature Scheme

We construct a signature scheme from any OWF as follows:

- $\mathsf{Gen}(1^n)$: To sign ℓ-bit messages, for $i \in \{1, \ldots, \ell\}$:
  - Choose $x_{i,0}, x_{i,1} \leftarrow \{0,1\}^n$
  - Compute $y_{i,0} = f(x_{i,0})$ and $y_{i,1} = f(x_{i,1})$
  - Output

$$pk = \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{\ell,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{\ell,1} \end{pmatrix} \qquad sk = \begin{pmatrix} x_{1,0} & x_{2,0} & \cdots & x_{\ell,0} \\ x_{1,1} & x_{2,1} & \cdots & x_{\ell,1} \end{pmatrix}$$

- $\mathsf{Sign}_{sk}(m \in \{0,1\}^\ell)$: Output $\sigma = x_{1,m_1}, x_{2,m_2}, \ldots, x_{\ell,m_\ell}$
- $\mathsf{Verify}_{pk}(m, \sigma)$: Output 1 if $f(x_{i,m_i}) = y_{i,m_i}$ for all $i \in \{1, \ldots, \ell\}$

Security Intuition

- σ includes one $x_{i,b}$ for each i, this the value of $x_{i,\bar{b}}$ remains secret

Lamport One-Time Signature Scheme

We construct a signature scheme from any OWF as follows:

- $\mathsf{Gen}(1^n)$: To sign ℓ-bit messages, for $i \in \{1, \ldots, \ell\}$:
  - Choose $x_{i,0}, x_{i,1} \leftarrow \{0,1\}^n$
  - Compute $y_{i,0} = f(x_{i,0})$ and $y_{i,1} = f(x_{i,1})$
  - Output

$$pk = \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{\ell,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{\ell,1} \end{pmatrix} \qquad sk = \begin{pmatrix} x_{1,0} & x_{2,0} & \cdots & x_{\ell,0} \\ x_{1,1} & x_{2,1} & \cdots & x_{\ell,1} \end{pmatrix}$$

- $\mathsf{Sign}_{sk}(m \in \{0,1\}^\ell)$: Output $\sigma = x_{1,m_1}, x_{2,m_2}, \ldots, x_{\ell,m_\ell}$
- $\mathsf{Verify}_{pk}(m, \sigma)$: Output 1 if $f(x_{i,m_i}) = y_{i,m_i}$ for all $i \in \{1, \ldots, \ell\}$

Security Intuition

- σ includes one $x_{i,b}$ for each i, this the value of $x_{i,\bar{b}}$ remains secret
- To produce sig on $m' \neq m$, $\mathcal{A}$ needs to recover at least one value $x_{i,b}$ he doesn't know

Lamport One-Time Signature Scheme

We construct a signature scheme from any OWF as follows:

- $\mathsf{Gen}(1^n)$: To sign ℓ-bit messages, for $i \in \{1, \ldots, \ell\}$:
  - Choose $x_{i,0}, x_{i,1} \leftarrow \{0,1\}^n$
  - Compute $y_{i,0} = f(x_{i,0})$ and $y_{i,1} = f(x_{i,1})$
  - Output

$$pk = \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{\ell,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{\ell,1} \end{pmatrix} \qquad sk = \begin{pmatrix} x_{1,0} & x_{2,0} & \cdots & x_{\ell,0} \\ x_{1,1} & x_{2,1} & \cdots & x_{\ell,1} \end{pmatrix}$$

- $\mathsf{Sign}_{sk}(m \in \{0,1\}^\ell)$: Output $\sigma = x_{1,m_1}, x_{2,m_2}, \ldots, x_{\ell,m_\ell}$
- $\mathsf{Verify}_{pk}(m, \sigma)$: Output 1 if $f(x_{i,m_i}) = y_{i,m_i}$ for all $i \in \{1, \ldots, \ell\}$

Security Intuition

- σ includes one $x_{i,b}$ for each i, this the value of $x_{i,\bar{b}}$ remains secret
- To produce sig on $m' \neq m$, $\mathcal{A}$ needs to recover at least one value $x_{i,b}$ he doesn't know
- Hence, $\mathcal{A}$ needs to invert f on the corresponding $y_{i,b}$ in the pk

Lamport OTS – Proof of Security

Assume $\mathcal{A}_c$ breaks OTS security of Lamport, build $\mathcal{A}_r$ that inverts f

Lamport OTS – Proof of Security

Assume $\mathcal{A}_c$ breaks OTS security of Lamport, build $\mathcal{A}_r$ that inverts f

$\mathcal{A}_r(y = f(x))$ – Inverts f on y

- $\mathcal{A}_r$ chooses $i^* \leftarrow \{1, \ldots, \ell\}$, $b^* \leftarrow \{0, 1\}$, and sets $y_{i^*, b^*} = y$

Lamport OTS – Proof of Security

Assume $\mathcal{A}_c$ breaks OTS security of Lamport, build $\mathcal{A}_r$ that inverts f

$\mathcal{A}_r(y = f(x))$ – Inverts f on y

- $\mathcal{A}_r$ chooses $i^* \leftarrow \{1, \ldots, \ell\}$, $b^* \leftarrow \{0, 1\}$, and sets $y_{i^*,b^*} = y$
  - For all other locations, $\mathcal{A}_r$ chooses $x_{i,b} \leftarrow \{0, 1\}^n$ and sets $y_{i,b} = f(x_{i,b})$

Lamport OTS – Proof of Security

Assume $\mathcal{A}_c$ breaks OTS security of Lamport, build $\mathcal{A}_r$ that inverts f

$\mathcal{A}_r(y = f(x))$ – Inverts f on y

- $\mathcal{A}_r$ chooses $i^* \leftarrow \{1, \ldots, \ell\}$, $b^* \leftarrow \{0, 1\}$, and sets $y_{i^*,b^*} = y$
  - For all other locations, $\mathcal{A}_r$ chooses $x_{i,b} \leftarrow \{0, 1\}^n$ and sets $y_{i,b} = f(x_{i,b})$
  - Output $pk = \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{\ell,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{\ell,1} \end{pmatrix}$

Lamport OTS – Proof of Security

Assume $\mathcal{A}_c$ breaks OTS security of Lamport, build $\mathcal{A}_r$ that inverts f

$\mathcal{A}_r(y = f(x))$ – Inverts f on y

- $\mathcal{A}_r$ chooses $i^* \leftarrow \{1, \ldots, \ell\}$, $b^* \leftarrow \{0,1\}$, and sets $y_{i^*,b^*} = y$
  - For all other locations, $\mathcal{A}_r$ chooses $x_{i,b} \leftarrow \{0,1\}^n$ and sets $y_{i,b} = f(x_{i,b})$
  - Output $pk = \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{\ell,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{\ell,1} \end{pmatrix}$
- Run $\mathcal{A}_c$ on pk

Lamport OTS – Proof of Security

Assume $\mathcal{A}_c$ breaks OTS security of Lamport, build $\mathcal{A}_r$ that inverts f

$\mathcal{A}_r(y = f(x))$ – Inverts f on y

- $\mathcal{A}_r$ chooses $i^* \leftarrow \{1, \ldots, \ell\}$, $b^* \leftarrow \{0,1\}$, and sets $y_{i^*,b^*} = y$
  - For all other locations, $\mathcal{A}_r$ chooses $x_{i,b} \leftarrow \{0,1\}^n$ and sets $y_{i,b} = f(x_{i,b})$
  - Output $pk = \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{\ell,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{\ell,1} \end{pmatrix}$
- Run $\mathcal{A}_c$ on pk
- When $\mathcal{A}_c$ requests signature on a message m

Lamport OTS – Proof of Security

Assume $\mathcal{A}_c$ breaks OTS security of Lamport, build $\mathcal{A}_r$ that inverts f

$\mathcal{A}_r(y = f(x))$ – Inverts f on y

- $\mathcal{A}_r$ chooses $i^* \leftarrow \{1, \ldots, \ell\}$, $b^* \leftarrow \{0,1\}$, and sets $y_{i^*,b^*} = y$
  - For all other locations, $\mathcal{A}_r$ chooses $x_{i,b} \leftarrow \{0,1\}^n$ and sets $y_{i,b} = f(x_{i,b})$
  - Output $pk = \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{\ell,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{\ell,1} \end{pmatrix}$
- Run $\mathcal{A}_c$ on pk
- When $\mathcal{A}_c$ requests signature on a message m
  - If $m_{i^*} = b^*$, then $\mathcal{A}_r$ aborts

Lamport OTS – Proof of Security

Assume $\mathcal{A}_c$ breaks OTS security of Lamport, build $\mathcal{A}_r$ that inverts f

$\mathcal{A}_r(y = f(x))$ – Inverts f on y

- $\mathcal{A}_r$ chooses $i^* \leftarrow \{1, \ldots, \ell\}$, $b^* \leftarrow \{0, 1\}$, and sets $y_{i^*,b^*} = y$
  - For all other locations, $\mathcal{A}_r$ chooses $x_{i,b} \leftarrow \{0,1\}^n$ and sets $y_{i,b} = f(x_{i,b})$
  - Output $pk = \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{\ell,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{\ell,1} \end{pmatrix}$
- Run $\mathcal{A}_c$ on pk
- When $\mathcal{A}_c$ requests signature on a message m
  - If $m_{i^*} = b^*$, then $\mathcal{A}_r$ aborts
  - Otherwise, return $\sigma = (x_{1,m_1}, \ldots, x_{\ell,m_\ell})$ – $\mathcal{A}_r$ knows all these x_i's

Lamport OTS – Proof of Security

Assume $\mathcal{A}_c$ breaks OTS security of Lamport, build $\mathcal{A}_r$ that inverts f

$\mathcal{A}_r(y = f(x))$ – Inverts f on y

- $\mathcal{A}_r$ chooses $i^* \leftarrow \{1, \ldots, \ell\}$, $b^* \leftarrow \{0, 1\}$, and sets $y_{i^*,b^*} = y$
  - For all other locations, $\mathcal{A}_r$ chooses $x_{i,b} \leftarrow \{0, 1\}^n$ and sets $y_{i,b} = f(x_{i,b})$
  - Output $pk = \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{\ell,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{\ell,1} \end{pmatrix}$
- Run $\mathcal{A}_c$ on pk
- When $\mathcal{A}_c$ requests signature on a message m
  - If $m_{i^*} = b^*$, then $\mathcal{A}_r$ aborts
  - Otherwise, return $\sigma = (x_{1,m_1}, \ldots, x_{\ell,m_\ell})$ – $\mathcal{A}_r$ knows all these x_i's
- When $\mathcal{A}_c$ outputs a forgery (m', σ)

Lamport OTS – Proof of Security

Assume $\mathcal{A}_c$ breaks OTS security of Lamport, build $\mathcal{A}_r$ that inverts f

$\mathcal{A}_r(y = f(x))$ – Inverts f on y

- $\mathcal{A}_r$ chooses $i^* \leftarrow \{1, \ldots, \ell\}$, $b^* \leftarrow \{0,1\}$, and sets $y_{i^*,b^*} = y$
  - For all other locations, $\mathcal{A}_r$ chooses $x_{i,b} \leftarrow \{0,1\}^n$ and sets $y_{i,b} = f(x_{i,b})$
  - Output $pk = \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{\ell,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{\ell,1} \end{pmatrix}$
- Run $\mathcal{A}_c$ on pk
- When $\mathcal{A}_c$ requests signature on a message m
  - If $m_{i^*} = b^*$, then $\mathcal{A}_r$ aborts
  - Otherwise, return $\sigma = (x_{1,m_1}, \ldots, x_{\ell,m_\ell})$ – $\mathcal{A}_r$ knows all these x_i's
- When $\mathcal{A}_c$ outputs a forgery (m', σ)
  - Check if $m'_{i^*} = b^*$, if so output x_{i^*,b^*} (from σ) as inverse

Lamport OTS – Proof of Security

Assume $\mathcal{A}_c$ breaks OTS security of Lamport, build $\mathcal{A}_r$ that inverts f

$\mathcal{A}_r(y = f(x))$ – Inverts f on y

- $\mathcal{A}_r$ chooses $i^* \leftarrow \{1, \ldots, \ell\}$, $b^* \leftarrow \{0, 1\}$, and sets $y_{i^*,b^*} = y$
  - For all other locations, $\mathcal{A}_r$ chooses $x_{i,b} \leftarrow \{0,1\}^n$ and sets $y_{i,b} = f(x_{i,b})$
  - Output $pk = \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{\ell,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{\ell,1} \end{pmatrix}$
- Run $\mathcal{A}_c$ on pk
- When $\mathcal{A}_c$ requests signature on a message m
  - If $m_{i^*} = b^*$, then $\mathcal{A}_r$ aborts
  - Otherwise, return $\sigma = (x_{1,m_1}, \ldots, x_{\ell,m_\ell})$ – $\mathcal{A}_r$ knows all these x_i's
- When $\mathcal{A}_c$ outputs a forgery (m', σ)
  - Check if $m'_{i^*} = b^*$, if so output x_{i^*,b^*} (from σ) as inverse

Analysis: (i^*, b^*) is random to $\mathcal{A}_c$, so $\Pr[m'_{i^*} \neq m_{i^*} \wedge m'_{i^*} = b^*] \geq \frac{1}{2\ell}$

Lamport OTS – Proof of Security

Assume $\mathcal{A}_c$ breaks OTS security of Lamport, build $\mathcal{A}_r$ that inverts f

$\mathcal{A}_r(y = f(x))$ – Inverts f on y

- $\mathcal{A}_r$ chooses $i^* \leftarrow \{1, \ldots, \ell\}$, $b^* \leftarrow \{0, 1\}$, and sets $y_{i^*,b^*} = y$
  - For all other locations, $\mathcal{A}_r$ chooses $x_{i,b} \leftarrow \{0,1\}^n$ and sets $y_{i,b} = f(x_{i,b})$
  - Output $pk = \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{\ell,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{\ell,1} \end{pmatrix}$
- Run $\mathcal{A}_c$ on pk
- When $\mathcal{A}_c$ requests signature on a message m
  - If $m_{i^*} = b^*$, then $\mathcal{A}_r$ aborts
  - Otherwise, return $\sigma = (x_{1,m_1}, \ldots, x_{\ell,m_\ell})$ – $\mathcal{A}_r$ knows all these x_i's
- When $\mathcal{A}_c$ outputs a forgery (m', σ)
  - Check if $m'_{i^*} = b^*$, if so output x_{i^*,b^*} (from σ) as inverse

Analysis: (i^*, b^*) is random to $\mathcal{A}_c$, so $\Pr[m'_{i^*} \neq m_{i^*} \wedge m'_{i^*} = b^*] \geq \frac{1}{2\ell}$

$$\Pr[\mathsf{Invert}_{\mathcal{A}_r,f} = 1] \geq \frac{1}{2\ell} \cdot \Pr[\mathsf{SigForge}_{\mathcal{A},\Pi}(n) = 1] \geq 1/\mathrm{poly}(n)$$

Improving on Lamport

Weaknesses of Lamport:

Improving on Lamport

Weaknesses of Lamport:

- Can only sign ℓ-bit messages

Improving on Lamport

Weaknesses of Lamport:

- Can only sign ℓ-bit messages
- Can only sign 1 message

Improving on Lamport

Weaknesses of Lamport:

- Can only sign ℓ-bit messages
- Can only sign 1 message

Signing Arbitrary Length Messages:

Improving on Lamport

Weaknesses of Lamport:

- Can only sign ℓ-bit messages
- Can only sign 1 message

Signing Arbitrary Length Messages:

Hash-and-Sign

Improving on Lamport

Weaknesses of Lamport:

- Can only sign ℓ-bit messages
- Can only sign 1 message

Signing Arbitrary Length Messages:

Hash-and-Sign

- Same idea as in Hash-and-MAC

Improving on Lamport

Weaknesses of Lamport:

- Can only sign ℓ-bit messages
- Can only sign 1 message

Signing Arbitrary Length Messages:

Hash-and-Sign

- Same idea as in Hash-and-MAC
- To sign $m \in \{0,1\}^*$, instead sign $H^s(m)$

Improving on Lamport

Weaknesses of Lamport:

- Can only sign ℓ-bit messages
- Can only sign 1 message

Signing Arbitrary Length Messages:

Hash-and-Sign

- Same idea as in Hash-and-MAC
- To sign $m \in \{0,1\}^*$, instead sign $H^s(m)$
- Now, only need to sign ℓ-bit hash outputs

Improving on Lamport

Weaknesses of Lamport:

- Can only sign ℓ-bit messages
- Can only sign 1 message

Signing Arbitrary Length Messages:

Hash-and-Sign

- Same idea as in Hash-and-MAC
- To sign $m \in \{0,1\}^*$, instead sign $H^s(m)$
- Now, only need to sign ℓ-bit hash outputs
- Include hash key s in public key

Improving on Lamport

Weaknesses of Lamport:

- Can only sign ℓ-bit messages
- Can only sign 1 message

Signing Arbitrary Length Messages:

Hash-and-Sign

- Same idea as in Hash-and-MAC
- To sign $m \in \{0,1\}^*$, instead sign $H^s(m)$
- Now, only need to sign ℓ-bit hash outputs
- Include hash key s in public key
- Secure if H is CRHF

Going Beyond One Signature

Try 1 – Many Keys

Going Beyond One Signature

Try 1 – Many Keys

- Gen generates t key pairs (pk_i, sk_i)

Going Beyond One Signature

Try 1 – Many Keys

- Gen generates t key pairs (pk_i, sk_i)
- Each key pair is used once

Going Beyond One Signature

Try 1 – Many Keys

- Gen generates t key pairs (pk_i, sk_i)
- Each key pair is used once
- Sign stores which keys have already been used to prevent reuse

Going Beyond One Signature

Try 1 – Many Keys

- Gen generates t key pairs (pk_i, sk_i)
- Each key pair is used once
- Sign stores which keys have already been used to prevent reuse

Limitations:

Going Beyond One Signature

Try 1 – Many Keys

- Gen generates t key pairs (pk_i, sk_i)
- Each key pair is used once
- Sign stores which keys have already been used to prevent reuse

Limitations:

- $|pk|$ is large – grows with t

Going Beyond One Signature

Try 1 – Many Keys

- Gen generates t key pairs (pk_i, sk_i)
- Each key pair is used once
- Sign stores which keys have already been used to prevent reuse

Limitations:

- $|pk|$ is large – grows with t
- number of possible signatures, t, bounded at Gen time

Going Beyond One Signature

Try 1 – Many Keys

- Gen generates t key pairs (pk_i, sk_i)
- Each key pair is used once
- Sign stores which keys have already been used to prevent reuse

Limitations:

- $|pk|$ is large – grows with t
- number of possible signatures, t, bounded at Gen time
- Signature is *stateful* – problematic if state is reset

Going Beyond One Signature

Try 2 – Chain-Based Signatures

Going Beyond One Signature

Try 2 – Chain-Based Signatures

- Generate key pairs (pk_i, sk_i) as we go

pk_1

Going Beyond One Signature

Try 2 – Chain-Based Signatures

- Generate key pairs (pk_i, sk_i) as we go

pk_1

$$\overbrace{m_1||pk_2}$$

Going Beyond One Signature

Try 2 – Chain-Based Signatures

- Generate key pairs (pk_i, sk_i) as we go

$$pk_1$$

$$\overbrace{m_1||pk_2}$$

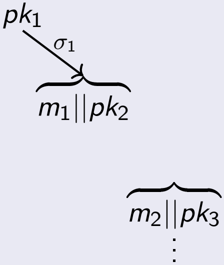
$$\overbrace{m_2||pk_3}$$

$$\vdots$$

Going Beyond One Signature

Try 2 – Chain-Based Signatures

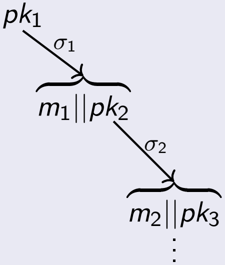
- Generate key pairs (pk_i, sk_i) as we go
- Use sk_i to sign $m_i||pk_{i+1}$



Going Beyond One Signature

Try 2 – Chain-Based Signatures

- Generate key pairs (pk_i, sk_i) as we go
- Use sk_i to sign $m_i||pk_{i+1}$



Going Beyond One Signature

Try 2 – Chain-Based Signatures

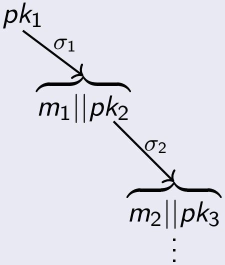
- Generate key pairs (pk_i, sk_i) as we go
- Use sk_i to sign $m_i||pk_{i+1}$
- σ includes all sigs (and pks) in chain

$$pk_1 \xrightarrow{\sigma_1} m_1||pk_2 \xrightarrow{\sigma_2} m_2||pk_3 \quad \vdots$$

Going Beyond One Signature

Try 2 – Chain-Based Signatures

- Generate key pairs (pk_i, sk_i) as we go
- Use sk_i to sign $m_i||pk_{i+1}$
- σ includes all sigs (and pks) in chain
- pk includes just original pk_1



Going Beyond One Signature

Try 2 – Chain-Based Signatures

- Generate key pairs (pk_i, sk_i) as we go
- Use sk_i to sign $m_i||pk_{i+1}$
- σ includes all sigs (and pks) in chain
- pk includes just original pk_1
- Verify verifies all signatures in chain starting at the top

$$\begin{array}{l} pk_1 \\ \quad \searrow^{\sigma_1} \\ \quad \overbrace{m_1||pk_2} \\ \qquad \searrow^{\sigma_2} \\ \qquad \overbrace{m_2||pk_3} \\ \qquad\quad \vdots \end{array}$$

Going Beyond One Signature

Try 2 – Chain-Based Signatures

- Generate key pairs (pk_i, sk_i) as we go
- Use sk_i to sign $m_i||pk_{i+1}$
- σ includes all sigs (and pks) in chain
- pk includes just original pk_1
- Verify verifies all signatures in chain starting at the top

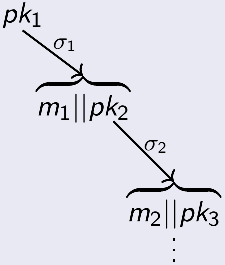
$$pk_1 \xrightarrow{\sigma_1} m_1||pk_2 \xrightarrow{\sigma_2} m_2||pk_3 \;\vdots$$

Pros:

Going Beyond One Signature

Try 2 – Chain-Based Signatures

- Generate key pairs (pk_i, sk_i) as we go
- Use sk_i to sign $m_i || pk_{i+1}$
- σ includes all sigs (and pks) in chain
- pk includes just original pk_1
- Verify verifies all signatures in chain starting at the top



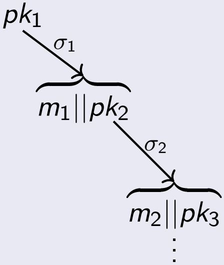
Pros:

- Every pk only used once – OTS security is sufficient

Going Beyond One Signature

Try 2 – Chain-Based Signatures

- Generate key pairs (pk_i, sk_i) as we go
- Use sk_i to sign $m_i||pk_{i+1}$
- σ includes all sigs (and pks) in chain
- pk includes just original pk_1
- Verify verifies all signatures in chain starting at the top



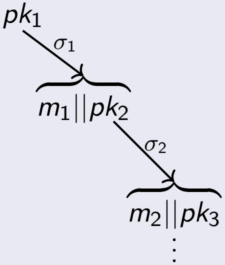
Pros:

- Every pk only used once – OTS security is sufficient
- No bound on number of signatures that can be issued

Going Beyond One Signature

Try 2 – Chain-Based Signatures

- Generate key pairs (pk_i, sk_i) as we go
- Use sk_i to sign $m_i||pk_{i+1}$
- σ includes all sigs (and pks) in chain
- pk includes just original pk_1
- Verify verifies all signatures in chain starting at the top



Pros:

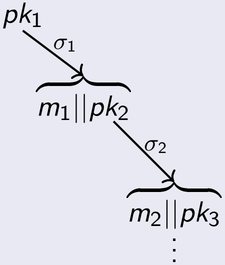
- Every pk only used once – OTS security is sufficient
- No bound on number of signatures that can be issued

Limitations:

Going Beyond One Signature

Try 2 – Chain-Based Signatures

- Generate key pairs (pk_i, sk_i) as we go
- Use sk_i to sign $m_i || pk_{i+1}$
- σ includes all sigs (and pks) in chain
- pk includes just original pk_1
- Verify verifies all signatures in chain starting at the top



Pros:

- Every pk only used once – OTS security is sufficient
- No bound on number of signatures that can be issued

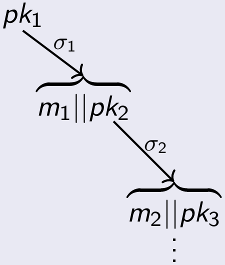
Limitations:

- $|\sigma|$ grows with the number of signatures issued

Going Beyond One Signature

Try 2 – Chain-Based Signatures

- Generate key pairs (pk_i, sk_i) as we go
- Use sk_i to sign $m_i || pk_{i+1}$
- σ includes all sigs (and pks) in chain
- pk includes just original pk_1
- Verify verifies all signatures in chain starting at the top



Pros:

- Every pk only used once – OTS security is sufficient
- No bound on number of signatures that can be issued

Limitations:

- $|\sigma|$ grows with the number of signatures issued
- Still have to store state

Going Beyond One Signature

Try 3 – Tree-Based Signatures

Going Beyond One Signature

Try 3 – Tree-Based Signatures

- Use a different pk for each possible message

Going Beyond One Signature

Try 3 – Tree-Based Signatures

- Use a different pk for each possible message
- Use depth ℓ tree to encode pk's

Going Beyond One Signature

Try 3 – Tree-Based Signatures

- Use a different pk for each possible message
- Use depth ℓ tree to encode pk's

pk_ϵ

Try 3 – Tree-Based Signatures

- Use a different *pk* for each possible message
- Use depth ℓ tree to encode *pk*'s

$$pk_\epsilon$$
$$\downarrow$$
$$pk_0 || pk_1$$

Going Beyond One Signature

Try 3 – Tree-Based Signatures

- Use a different pk for each possible message
- Use depth ℓ tree to encode pk's

$$
\begin{tikzcd}
pk_\epsilon \arrow[d] & \\
pk_0 || pk_1 \arrow[dr] & \\
& pk_{10} || pk_{11}
\end{tikzcd}
$$

Going Beyond One Signature

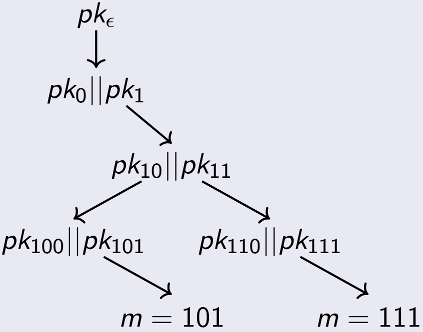
Try 3 – Tree-Based Signatures

- Use a different pk for each possible message
- Use depth ℓ tree to encode pk's

$$\begin{tikzcd}
& pk_\epsilon \arrow[d] & & \\
& pk_0||pk_1 \arrow[dr] & & \\
& & pk_{10}||pk_{11} \arrow[dl] \arrow[dr] & \\
& pk_{100}||pk_{101} & & pk_{110}||pk_{111}
\end{tikzcd}$$

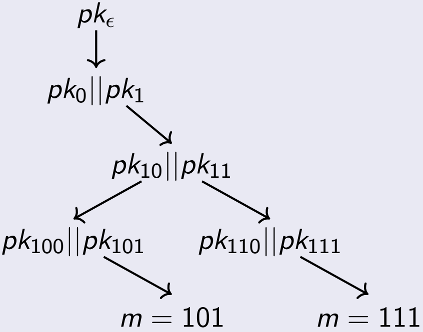
Try 3 – Tree-Based Signatures

- Use a different *pk* for each possible message
- Use depth ℓ tree to encode *pk*'s



Try 3 – Tree-Based Signatures

- Use a different pk for each possible message
- Use depth ℓ tree to encode pk's
- pk_i signs next node on path to m



Going Beyond One Signature

Try 3 – Tree-Based Signatures

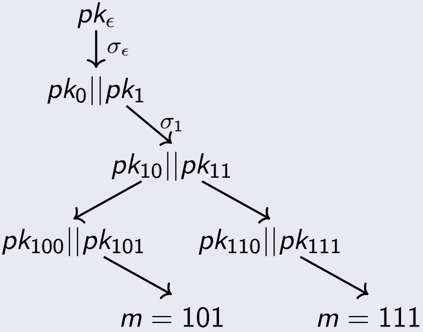
- Use a different pk for each possible message
- Use depth ℓ tree to encode pk's
- pk_i signs next node on path to m

$$\begin{tikzcd}
& pk_\epsilon \arrow[d, "\sigma_\epsilon"] & & & \\
& pk_0||pk_1 \arrow[dr] & & & \\
& & pk_{10}||pk_{11} \arrow[dll] \arrow[dr] & & \\
pk_{100}||pk_{101} \arrow[drr] & & & pk_{110}||pk_{111} \arrow[dr] & \\
& & m = 101 & & m = 111
\end{tikzcd}$$

Going Beyond One Signature

Try 3 – Tree-Based Signatures

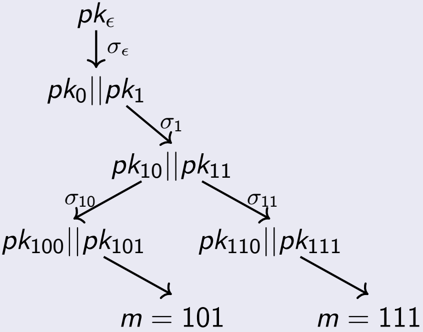
- Use a different pk for each possible message
- Use depth ℓ tree to encode pk's
- pk_i signs next node on path to m



Going Beyond One Signature

Try 3 – Tree-Based Signatures

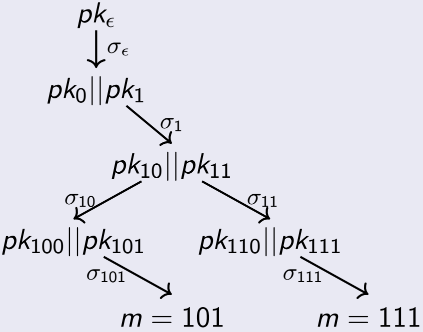
- Use a different pk for each possible message
- Use depth ℓ tree to encode pk's
- pk_i signs next node on path to m



Going Beyond One Signature

Try 3 – Tree-Based Signatures

- Use a different pk for each possible message
- Use depth ℓ tree to encode pk's
- pk_i signs next node on path to m



Going Beyond One Signature

Try 3 – Tree-Based Signatures

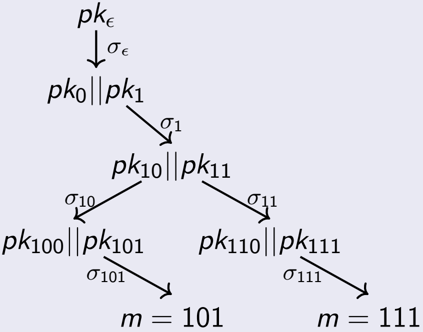
- Use a different pk for each possible message
- Use depth ℓ tree to encode pk's
- pk_i signs next node on path to m
- σ on message m, outputs signatures on path to m

$$
\begin{tikzcd}
 & pk_\epsilon \arrow[d, "\sigma_\epsilon"] & & & \\
 & pk_0||pk_1 \arrow[dr, "\sigma_1"] & & & \\
 & & pk_{10}||pk_{11} \arrow[dl, "\sigma_{10}"'] \arrow[dr, "\sigma_{11}"] & & \\
 & pk_{100}||pk_{101} \arrow[dr, "\sigma_{101}"'] & & pk_{110}||pk_{111} \arrow[dr, "\sigma_{111}"'] & \\
 & & m = 101 & & m = 111
\end{tikzcd}
$$

Going Beyond One Signature

Try 3 – Tree-Based Signatures

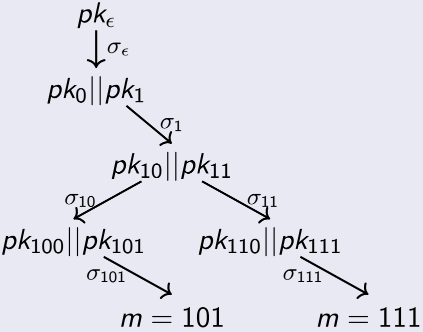
- Use a different pk for each possible message
- Use depth ℓ tree to encode pk's
- pk_i signs next node on path to m
- σ on message m, outputs signatures on path to m
- Only generate keys as needed, otherwise exponential time



Going Beyond One Signature

Try 3 – Tree-Based Signatures

- Use a different pk for each possible message
- Use depth ℓ tree to encode pk's
- pk_i signs next node on path to m
- σ on message m, outputs signatures on path to m
- Only generate keys as needed, otherwise exponential time



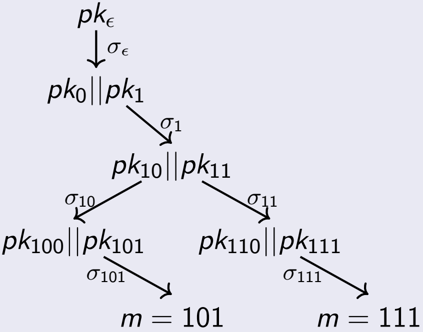
Pros:

- $|\sigma| = O(\ell)$, can sign up to 2^ℓ messages

Going Beyond One Signature

Try 3 – Tree-Based Signatures

- Use a different pk for each possible message
- Use depth ℓ tree to encode pk's
- pk_i signs next node on path to m
- σ on message m, outputs signatures on path to m
- Only generate keys as needed, otherwise exponential time



Pros:

- $|\sigma| = O(\ell)$, can sign up to 2^ℓ messages

Limitations:

- Still stateful – Tree is the state

Making It Stateless

Idea: Use a PRF to generate randomness for Gen at each node

Making It Stateless

Idea: Use a PRF to generate randomness for Gen at each node

- For each node w, store r_w and set $(sk_w, pk_w) = \text{Gen}(r_w)$

Making It Stateless

Idea: Use a PRF to generate randomness for Gen at each node

- For each node w, store r_w and set $(sk_w, pk_w) = \text{Gen}(r_w)$
- Instead of storing r_w, compute it as $r_w = F_k(w)$

Making It Stateless

Idea: Use a PRF to generate randomness for Gen at each node

- For each node w, store r_w and set $(sk_w, pk_w) = \text{Gen}(r_w)$
- Instead of storing r_w, compute it as $r_w = F_k(w)$
- PRF key k is stored as part of global secret key

Making It Stateless

Idea: Use a PRF to generate randomness for Gen at each node

- For each node w, store r_w and set $(sk_w, pk_w) = \text{Gen}(r_w)$
- Instead of storing r_w, compute it as $r_w = F_k(w)$
- PRF key k is stored as part of global secret key

Final Result – Stateless Tree-Based Signatures

Making It Stateless

Idea: Use a PRF to generate randomness for Gen at each node

- For each node w, store r_w and set $(sk_w, pk_w) = \mathrm{Gen}(r_w)$
- Instead of storing r_w, compute it as $r_w = F_k(w)$
- PRF key k is stored as part of global secret key

Final Result – Stateless Tree-Based Signatures

We have constructed a signature scheme that is:

Making It Stateless

Idea: Use a PRF to generate randomness for Gen at each node

- For each node w, store r_w and set $(sk_w, pk_w) = \text{Gen}(r_w)$
- Instead of storing r_w, compute it as $r_w = F_k(w)$
- PRF key k is stored as part of global secret key

Final Result – Stateless Tree-Based Signatures

We have constructed a signature scheme that is:

- Stateless

Making It Stateless

Idea: Use a PRF to generate randomness for Gen at each node

- For each node w, store r_w and set $(sk_w, pk_w) = \text{Gen}(r_w)$
- Instead of storing r_w, compute it as $r_w = F_k(w)$
- PRF key k is stored as part of global secret key

Final Result – Stateless Tree-Based Signatures

We have constructed a signature scheme that is:

- Stateless
- $|\sigma| = \ell$ signatures

Making It Stateless

Idea: Use a PRF to generate randomness for Gen at each node

- For each node w, store r_w and set $(sk_w, pk_w) = \text{Gen}(r_w)$
- Instead of storing r_w, compute it as $r_w = F_k(w)$
- PRF key k is stored as part of global secret key

Final Result – Stateless Tree-Based Signatures

We have constructed a signature scheme that is:

- Stateless
- $|\sigma| = \ell$ signatures
- Can sign arbitrary length m – using Hash-and-MAC

Making It Stateless

Idea: Use a PRF to generate randomness for Gen at each node

- For each node w, store r_w and set $(sk_w, pk_w) = \text{Gen}(r_w)$
- Instead of storing r_w, compute it as $r_w = F_k(w)$
- PRF key k is stored as part of global secret key

Final Result – Stateless Tree-Based Signatures

We have constructed a signature scheme that is:

- Stateless
- $|\sigma| = \ell$ signatures
- Can sign arbitrary length m – using Hash-and-MAC
- Only requires private-key primitives (OWFs, CRHFs)

Making It Stateless

Idea: Use a PRF to generate randomness for Gen at each node

- For each node w, store r_w and set $(sk_w, pk_w) = \text{Gen}(r_w)$
- Instead of storing r_w, compute it as $r_w = F_k(w)$
- PRF key k is stored as part of global secret key

Final Result – Stateless Tree-Based Signatures

We have constructed a signature scheme that is:

- Stateless
- $|\sigma| = \ell$ signatures
- Can sign arbitrary length m – using Hash-and-MAC
- Only requires private-key primitives (OWFs, CRHFs)

Comparison to Public-Key Signatures

- Surprisingly, we can build signatures from private-key techniques
- But, public-key based signatures are more efficient (shorter sigs)

Outline

1. Lecture 23 Review
2. Digital Signatures from Private-Key Techniques
3. Digital Signatures from Discrete Log

Identification Schemes

Identification Scheme:

- Interactive protocol to allow a party to prove identity

Identification Schemes

Identification Scheme:

- Interactive protocol to allow a party to prove identity
- Relaxation of signature, similar to KE as relaxation for PKE

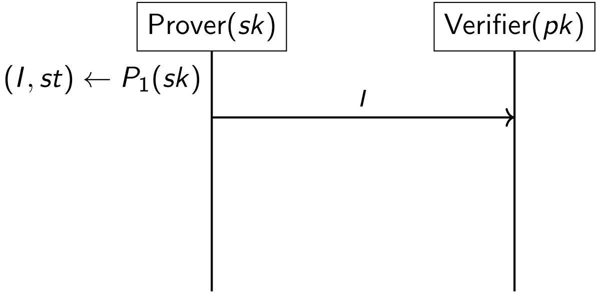
Identification Schemes

Identification Scheme:

- Interactive protocol to allow a party to prove identity
- Relaxation of signature, similar to KE as relaxation for PKE
- We consider 3-round variant with following form

Identification Scheme:

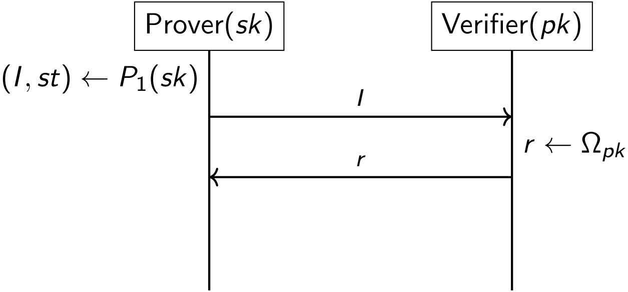
- Interactive protocol to allow a party to prove identity
- Relaxation of signature, similar to KE as relaxation for PKE
- We consider 3-round variant with following form



Identification Schemes

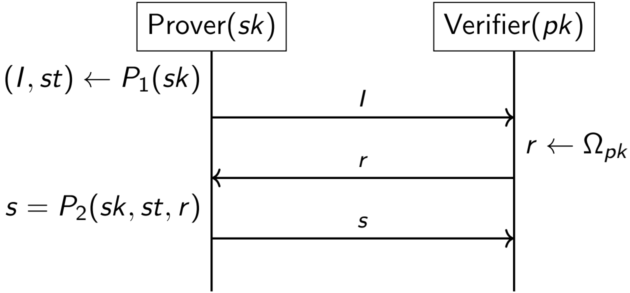
Identification Scheme:

- Interactive protocol to allow a party to prove identity
- Relaxation of signature, similar to KE as relaxation for PKE
- We consider 3-round variant with following form



Identification Scheme:

- Interactive protocol to allow a party to prove identity
- Relaxation of signature, similar to KE as relaxation for PKE
- We consider 3-round variant with following form



Identification Schemes

Identification Scheme:

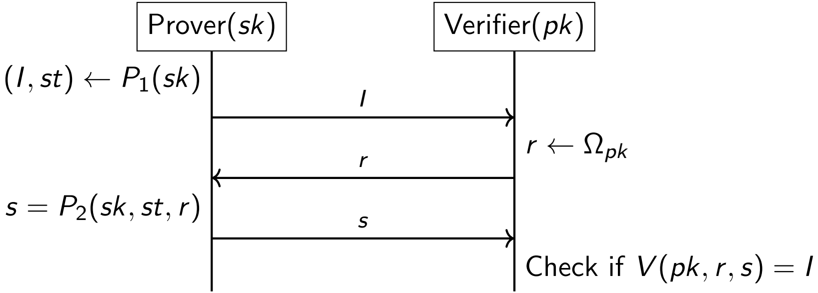
- Interactive protocol to allow a party to prove identity
- Relaxation of signature, similar to KE as relaxation for PKE
- We consider 3-round variant with following form

Prover(sk)		Verifier(pk)
$(I, st) \leftarrow P_1(sk)$	$\xrightarrow{\quad I \quad}$	
	$\xleftarrow{\quad r \quad}$	$r \leftarrow \Omega_{pk}$
$s = P_2(sk, st, r)$	$\xrightarrow{\quad s \quad}$	
		Check if $V(pk, r, s) = I$

Identification Schemes

Identification Scheme:

- Interactive protocol to allow a party to prove identity
- Relaxation of signature, similar to KE as relaxation for PKE
- We consider 3-round variant with following form



Non-degenerate:

- ID scheme is non-degenerate if for all sk, each I occurs with only negligible probability.

Security of Identification Schemes

Informal

Prover should not be able to get Verifier to accept without knowing *sk*

Security of Identification Schemes

Informal

Prover should not be able to get Verifier to accept without knowing *sk*

Let $\Pi = (\mathsf{Gen}, P_1, P_2, V)$ be an ID scheme. Consider the following game between an adversary $\mathcal{A}$ and a challenger:

$\mathsf{Ident}_{\mathcal{A},\Pi}(n)$

Security of Identification Schemes

Informal

Prover should not be able to get Verifier to accept without knowing sk

Let $\Pi = (\text{Gen}, P_1, P_2, V)$ be an ID scheme. Consider the following game between an adversary $\mathcal{A}$ and a challenger:

$\text{Ident}_{\mathcal{A},\Pi}(n)$

- Challenger runs $(pk, sk) \leftarrow \text{Gen}(1^n)$ and gives pk to $\mathcal{A}$

Security of Identification Schemes

Informal

Prover should not be able to get Verifier to accept without knowing sk

Let $\Pi = (\text{Gen}, P_1, P_2, V)$ be an ID scheme. Consider the following game between an adversary $\mathcal{A}$ and a challenger:

$\text{Ident}_{\mathcal{A},\Pi}(n)$

- Challenger runs $(pk, sk) \leftarrow \text{Gen}(1^n)$ and gives pk to $\mathcal{A}$
- $\mathcal{A}$ gets pk and oracle access to Trans_{sk} and outputs I

Security of Identification Schemes

Informal

Prover should not be able to get Verifier to accept without knowing sk

Let $\Pi = (\mathsf{Gen}, P_1, P_2, V)$ be an ID scheme. Consider the following game between an adversary $\mathcal{A}$ and a challenger:

$\mathsf{Ident}_{\mathcal{A},\Pi}(n)$

- Challenger runs $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ and gives pk to $\mathcal{A}$
- $\mathcal{A}$ gets pk and oracle access to Trans_{sk} and outputs I
  - Trans_{sk} runs honest execution and outputs transcript (I, r, s)

Security of Identification Schemes

Informal

Prover should not be able to get Verifier to accept without knowing *sk*

Let $\Pi = (\mathsf{Gen}, P_1, P_2, V)$ be an ID scheme. Consider the following game between an adversary $\mathcal{A}$ and a challenger:

$\mathsf{Ident}_{\mathcal{A},\Pi}(n)$

- Challenger runs $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ and gives pk to $\mathcal{A}$
- $\mathcal{A}$ gets pk and oracle access to Trans_{sk} and outputs I
  - Trans_{sk} runs honest execution and outputs transcript (I, r, s)
- Challenger chooses $r \leftarrow \Omega_{pk}$ and gives r to $\mathcal{A}$

Security of Identification Schemes

Informal

Prover should not be able to get Verifier to accept without knowing sk

Let $\Pi = (\text{Gen}, P_1, P_2, V)$ be an ID scheme. Consider the following game between an adversary $\mathcal{A}$ and a challenger:

$\text{Ident}_{\mathcal{A},\Pi}(n)$

- Challenger runs $(pk, sk) \leftarrow \text{Gen}(1^n)$ and gives pk to $\mathcal{A}$
- $\mathcal{A}$ gets pk and oracle access to Trans_{sk} and outputs I
  - Trans_{sk} runs honest execution and outputs transcript (I, r, s)
- Challenger chooses $r \leftarrow \Omega_{pk}$ and gives r to $\mathcal{A}$
- $\mathcal{A}$ receives r and responds with s

Security of Identification Schemes

Informal

Prover should not be able to get Verifier to accept without knowing sk

Let $\Pi = (\mathsf{Gen}, P_1, P_2, V)$ be an ID scheme. Consider the following game between an adversary $\mathcal{A}$ and a challenger:

$\mathsf{Ident}_{\mathcal{A},\Pi}(n)$

- Challenger runs $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ and gives pk to $\mathcal{A}$
- $\mathcal{A}$ gets pk and oracle access to Trans_{sk} and outputs I
  - Trans_{sk} runs honest execution and outputs transcript (I, r, s)
- Challenger chooses $r \leftarrow \Omega_{pk}$ and gives r to $\mathcal{A}$
- $\mathcal{A}$ receives r and responds with s
- We say that $\mathsf{Ident}_{\mathcal{A},\Pi}(n) = 1$ (i.e., $\mathcal{A}$ wins) if $V(pk, r, s) = I$

Security of Identification Schemes

Informal

Prover should not be able to get Verifier to accept without knowing sk

Let $\Pi = (\mathsf{Gen}, P_1, P_2, V)$ be an ID scheme. Consider the following game between an adversary $\mathcal{A}$ and a challenger:

$\mathsf{Ident}_{\mathcal{A},\Pi}(n)$

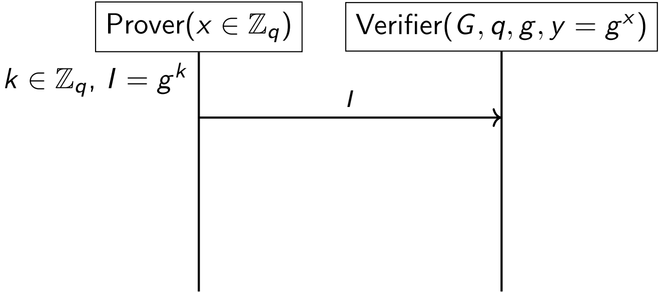
- Challenger runs $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ and gives pk to $\mathcal{A}$
- $\mathcal{A}$ gets pk and oracle access to Trans_{sk} and outputs I
  - Trans_{sk} runs honest execution and outputs transcript (I, r, s)
- Challenger chooses $r \leftarrow \Omega_{pk}$ and gives r to $\mathcal{A}$
- $\mathcal{A}$ receives r and responds with s
- We say that $\mathsf{Ident}_{\mathcal{A},\Pi}(n) = 1$ (i.e., $\mathcal{A}$ wins) if $V(pk, r, s) = I$.

Definition: An ID scheme $\Pi = (\mathsf{Gen}, P_1, P_2, V)$ is *secure* if for all PPT $\mathcal{A}$,

$$\Pr[\mathsf{Ident}_{\mathcal{A},\Pi}(n) = 1] \leq \mathsf{negl}(n)$$

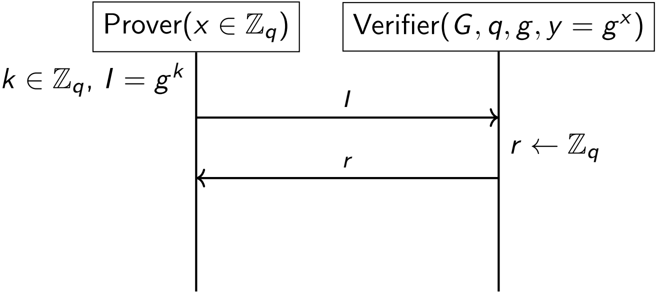
Schnorr ID Scheme

Prover$(x \in \mathbb{Z}_q)$	Verifier$(G, q, g, y = g^x)$

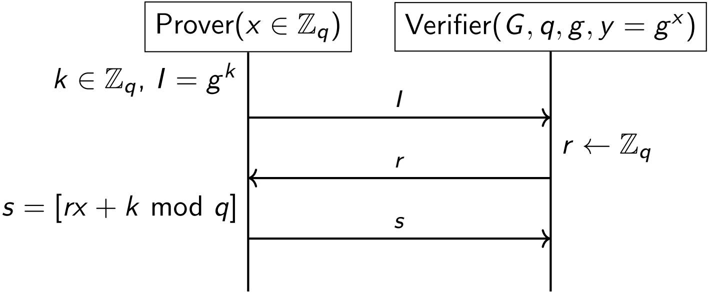


Prover($x \in \mathbb{Z}_q$)
Verifier($G, q, g, y = g^x$)
$k \in \mathbb{Z}_q$, $I = g^k$
I

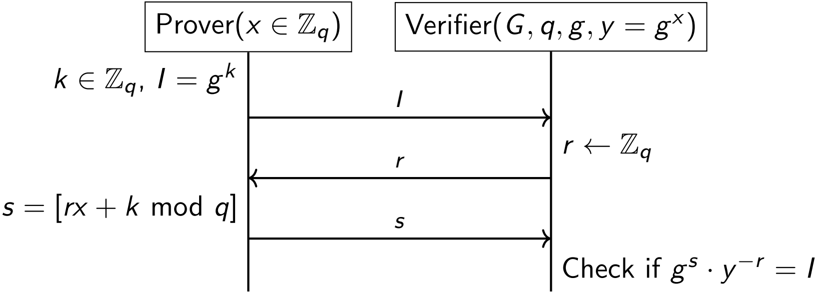
Schnorr ID Scheme



Schnorr ID Scheme



Schnorr ID Scheme



Schnorr ID Scheme

Prover($x \in \mathbb{Z}_q$)		Verifier($G, q, g, y = g^x$)
$k \in \mathbb{Z}_q,\ I = g^k$	$\xrightarrow{\quad I \quad}$	
	$\xleftarrow{\quad r \quad}$	$r \leftarrow \mathbb{Z}_q$
$s = [rx + k \bmod q]$	$\xrightarrow{\quad s \quad}$	
		Check if $g^s \cdot y^{-r} = I$

Correctness: $g^s \cdot y^{-r} = g^{(rx+k)} \cdot g^{-rx} = g^k = I$

Security of Schnorr ID Scheme

Proof Sketch:

Security of Schnorr ID Scheme

Proof Sketch:

- Assume that $\mathcal{A}_c$ breaks ID scheme security:

Security of Schnorr ID Scheme

Proof Sketch:

- Assume that $\mathcal{A}_c$ breaks ID scheme security:
  - Trans oracle is not useful, $\mathcal{A}_c$ can simulate transcripts on its own
    - Sample $r, s \leftarrow \mathbb{Z}_q$, set $I = g^s \cdot y^{-r}$

Security of Schnorr ID Scheme

Proof Sketch:

- Assume that $\mathcal{A}_c$ breaks ID scheme security:
  - Trans oracle is not useful, $\mathcal{A}_c$ can simulate transcripts on its own
    - Sample $r, s \leftarrow \mathbb{Z}_q$, set $I = g^s \cdot y^{-r}$
  - $\mathcal{A}_c$ gets $y = g^x$, sends I, gets r, and computes s s.t. $g^s \cdot y^{-r} = I$

Security of Schnorr ID Scheme

Proof Sketch:

- Assume that $\mathcal{A}_c$ breaks ID scheme security:
  - Trans oracle is not useful, $\mathcal{A}_c$ can simulate transcripts on its own
    - Sample $r, s \leftarrow \mathbb{Z}_q$, set $I = g^s \cdot y^{-r}$
  - $\mathcal{A}_c$ gets $y = g^x$, sends I, gets r, and computes s s.t. $g^s \cdot y^{-r} = I$
    - If $\mathcal{A}_c$ can do this twice (for same I), it can solve DLOG

Security of Schnorr ID Scheme

Proof Sketch:

- Assume that $\mathcal{A}_c$ breaks ID scheme security:
  - Trans oracle is not useful, $\mathcal{A}_c$ can simulate transcripts on its own
    - Sample $r, s \leftarrow \mathbb{Z}_q$, set $I = g^s \cdot y^{-r}$
  - $\mathcal{A}_c$ gets $y = g^x$, sends I, gets r, and computes s s.t. $g^s \cdot y^{-r} = I$
    - If $\mathcal{A}_c$ can do this twice (for same I), it can solve DLOG

$$\begin{aligned} g^{s_1} \cdot y^{-r_1} = I = g^{s_2} \cdot y^{-r_2} \quad &\Longrightarrow \quad g^{s_1 - s_2} = y^{r_1 - r_2} \\ &\Longrightarrow \quad \log_g y = [(s_1 - s_2)(r_1 - r_2)^{-1} \bmod q] \end{aligned}$$

- Build $\mathcal{A}_r$ that solves DLOG:

Security of Schnorr ID Scheme

Proof Sketch:

- Assume that $\mathcal{A}_c$ breaks ID scheme security:
  - Trans oracle is not useful, $\mathcal{A}_c$ can simulate transcripts on its own
    - Sample $r, s \leftarrow \mathbb{Z}_q$, set $I = g^s \cdot y^{-r}$
  - $\mathcal{A}_c$ gets $y = g^x$, sends I, gets r, and computes s s.t. $g^s \cdot y^{-r} = I$
    - If $\mathcal{A}_c$ can do this twice (for same I), it can solve DLOG

$$\begin{aligned} g^{s_1} \cdot y^{-r_1} = I = g^{s_2} \cdot y^{-r_2} \quad &\Longrightarrow \quad g^{s_1 - s_2} = y^{r_1 - r_2} \\ &\Longrightarrow \quad \log_g y = [(s_1 - s_2)(r_1 - r_2)^{-1} \bmod q] \end{aligned}$$

- Build $\mathcal{A}_r$ that solves DLOG:
  - $\mathcal{A}_r$ runs $\mathcal{A}_c$ twice with the same randomness (producing same I), but gives it two different r

Security of Schnorr ID Scheme

Proof Sketch:

- Assume that $\mathcal{A}_c$ breaks ID scheme security:
  - Trans oracle is not useful, $\mathcal{A}_c$ can simulate transcripts on its own
    - Sample $r, s \leftarrow \mathbb{Z}_q$, set $I = g^s \cdot y^{-r}$
  - $\mathcal{A}_c$ gets $y = g^x$, sends I, gets r, and computes s s.t. $g^s \cdot y^{-r} = I$
    - If $\mathcal{A}_c$ can do this twice (for same I), it can solve DLOG

$$g^{s_1} \cdot y^{-r_1} = I = g^{s_2} \cdot y^{-r_2} \implies g^{s_1 - s_2} = y^{r_1 - r_2}$$
$$\implies \log_g y = [(s_1 - s_2)(r_1 - r_2)^{-1} \bmod q]$$

- Build $\mathcal{A}_r$ that solves DLOG:
  - $\mathcal{A}_r$ runs $\mathcal{A}_c$ twice with the same randomness (producing same I), but gives it two different r
  - If $\mathcal{A}_c$ succeeds twice, then break DLOG

Fiat-Shamir Transform

Goal

Convert 3-round ID scheme as earlier to a non-interactive signature

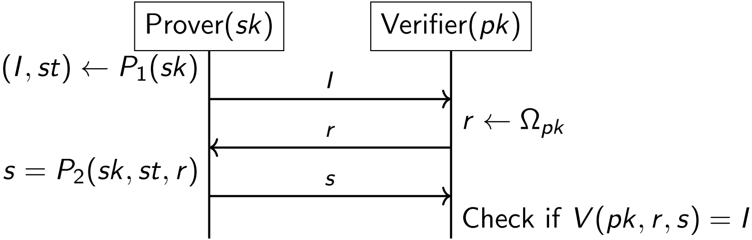
- Key Idea: Have Signer compute r himself using a hash

Fiat-Shamir Transform

Goal

Convert 3-round ID scheme as earlier to a non-interactive signature

- Key Idea: Have Signer compute r himself using a hash

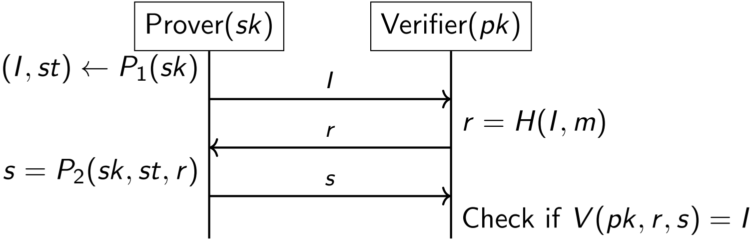


Fiat-Shamir Transform

Goal

Convert 3-round ID scheme as earlier to a non-interactive signature

- Key Idea: Have Signer compute r himself using a hash

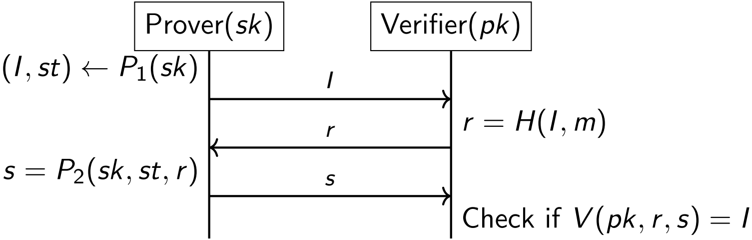


Fiat-Shamir Transform

Goal

Convert 3-round ID scheme as earlier to a non-interactive signature

- Key Idea: Have Signer compute r himself using a hash



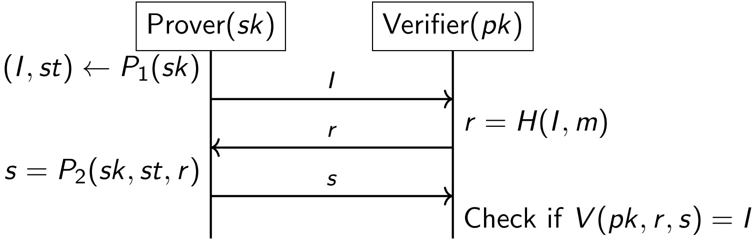
Fiat-Shamir Transform

Fiat-Shamir Transform

Goal

Convert 3-round ID scheme as earlier to a non-interactive signature

- Key Idea: Have Signer compute r himself using a hash



Fiat-Shamir Transform

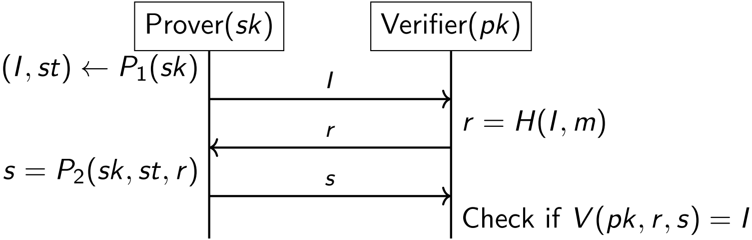
- $\text{Sign}_{sk}(m)$: $I \leftarrow P_1(sk)$, set $r = H(I, m)$, $s = P_2(sk, r)$, out $\sigma = (r, s)$

Fiat-Shamir Transform

Goal

Convert 3-round ID scheme as earlier to a non-interactive signature

- Key Idea: Have Signer compute r himself using a hash



Fiat-Shamir Transform

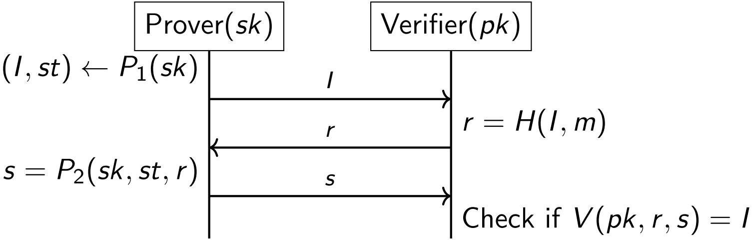
- $\text{Sign}_{sk}(m)$: $I \leftarrow P_1(sk)$, set $r = H(I, m)$, $s = P_2(sk, r)$, out $\sigma = (r, s)$
- $\text{Verify}_{pk}(m, \sigma)$: Compute $I = V(pk, r, s)$, check if $r = H(I, m)$

Fiat-Shamir Transform

Goal

Convert 3-round ID scheme as earlier to a non-interactive signature

- Key Idea: Have Signer compute r himself using a hash



Fiat-Shamir Transform

- $\mathrm{Sign}_{sk}(m)$: $I \leftarrow P_1(sk)$, set $r = H(I, m)$, $s = P_2(sk, r)$, out $\sigma = (r, s)$
- $\mathrm{Verify}_{pk}(m, \sigma)$: Compute $I = V(pk, r, s)$, check if $r = H(I, m)$
- Security: Secure if H is modeled as random oracle

Schnorr ID Scheme

Prover($x \in \mathbb{Z}_q$)		Verifier($G, q, g, y = g^x$)
$k \in \mathbb{Z}_q,\ I = g^k$	$\xrightarrow{\quad I \quad}$	
	$\xleftarrow{\quad r \quad}$	$r \leftarrow \mathbb{Z}_q$
$s = [rx + k \bmod q]$	$\xrightarrow{\quad s \quad}$	
		Check if $g^s \cdot y^{-r} = I$

Correctness: $g^s \cdot y^{-r} = g^{(rx+k)} \cdot g^{-rx} = g^k = I$

Apply Fiat Shamir

Replace $r \leftarrow \mathbb{Z}_q$ with $H(I, m)$

Schnorr Digital Signature

Schnorr Signature

- $\mathsf{Gen}(1^n)$:
  - $(G, q, g) \leftarrow \mathsf{Gen}(1^n)$, $x \leftarrow \mathbb{Z}_q$, $y = g^x$

Schnorr Digital Signature

Schnorr Signature

- Gen(1^n):
  - $(G, q, g) \leftarrow \text{Gen}(1^n)$, $x \leftarrow \mathbb{Z}_q$, $y = g^x$
  - $pk = (G, q, g, y)$, $sk = x$

Schnorr Digital Signature

Schnorr Signature

- $\mathsf{Gen}(1^n)$:
  - $(G, q, g) \leftarrow \mathsf{Gen}(1^n)$, $x \leftarrow \mathbb{Z}_q$, $y = g^x$
  - $pk = (G, q, g, y)$, $sk = x$
- $\mathsf{Sign}_{sk}(m \in \{0,1\}^*)$:
  - $k \leftarrow \mathbb{Z}_q$, $I = g^k$, $r = H(I, m)$ using $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$

Schnorr Digital Signature

Schnorr Signature

- $\mathsf{Gen}(1^n)$:
  - $(G, q, g) \leftarrow \mathsf{Gen}(1^n)$, $x \leftarrow \mathbb{Z}_q$, $y = g^x$
  - $pk = (G, q, g, y)$, $sk = x$
- $\mathsf{Sign}_{sk}(m \in \{0,1\}^*)$:
  - $k \leftarrow \mathbb{Z}_q$, $I = g^k$, $r = H(I, m)$ using $H : \{0,1\}^* \to \mathbb{Z}_q$
  - $s = [rx + k \bmod q]$

Schnorr Digital Signature

Schnorr Signature

- $\mathsf{Gen}(1^n)$:
  - $(G, q, g) \leftarrow \mathsf{Gen}(1^n)$, $x \leftarrow \mathbb{Z}_q$, $y = g^x$
  - $pk = (G, q, g, y)$, $sk = x$
- $\mathsf{Sign}_{sk}(m \in \{0,1\}^*)$:
  - $k \leftarrow \mathbb{Z}_q$, $I = g^k$, $r = H(I, m)$ using $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$
  - $s = [rx + k \mod q]$
  - Output $\sigma = (r, s)$

Schnorr Digital Signature

Schnorr Signature

- $\mathrm{Gen}(1^n)$:
  - $(G, q, g) \leftarrow \mathrm{Gen}(1^n)$, $x \leftarrow \mathbb{Z}_q$, $y = g^x$
  - $pk = (G, q, g, y)$, $sk = x$
- $\mathrm{Sign}_{sk}(m \in \{0,1\}^*)$:
  - $k \leftarrow \mathbb{Z}_q$, $I = g^k$, $r = H(I, m)$ using $H : \{0,1\}^* \to \mathbb{Z}_q$
  - $s = [rx + k \bmod q]$
  - Output $\sigma = (r, s)$
- $\mathrm{Verify}_{pk}(m, \sigma)$:
  - Set $I = g^s \cdot y^{-r}$

Schnorr Digital Signature

Schnorr Signature

- Gen(1^n):
  - $(G, q, g) \leftarrow \text{Gen}(1^n)$, $x \leftarrow \mathbb{Z}_q$, $y = g^x$
  - $pk = (G, q, g, y)$, $sk = x$
- $\text{Sign}_{sk}(m \in \{0,1\}^*)$:
  - $k \leftarrow \mathbb{Z}_q$, $I = g^k$, $r = H(I, m)$ using $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$
  - $s = [rx + k \bmod q]$
  - Output $\sigma = (r, s)$
- $\text{Verify}_{pk}(m, \sigma)$:
  - Set $I = g^s \cdot y^{-r}$
  - Output 1 if $H(I, m) = r$

Schnorr Digital Signature

Schnorr Signature

- $\text{Gen}(1^n)$:
  - $(G, q, g) \leftarrow \text{Gen}(1^n)$, $x \leftarrow \mathbb{Z}_q$, $y = g^x$
  - $pk = (G, q, g, y)$, $sk = x$
- $\text{Sign}_{sk}(m \in \{0,1\}^*)$:
  - $k \leftarrow \mathbb{Z}_q$, $I = g^k$, $r = H(I, m)$ using $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$
  - $s = [rx + k \bmod q]$
  - Output $\sigma = (r, s)$
- $\text{Verify}_{pk}(m, \sigma)$:
  - Set $I = g^s \cdot y^{-r}$
  - Output 1 if $H(I, m) = r$

Security: Secure based on DLOG in random oracle model

Schnorr Digital Signature

Schnorr Signature

- $\mathsf{Gen}(1^n)$:
  - $(G, q, g) \leftarrow \mathsf{Gen}(1^n)$, $x \leftarrow \mathbb{Z}_q$, $y = g^x$
  - $pk = (G, q, g, y)$, $sk = x$
- $\mathsf{Sign}_{sk}(m \in \{0,1\}^*)$:
  - $k \leftarrow \mathbb{Z}_q$, $I = g^k$, $r = H(I, m)$ using $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$
  - $s = [rx + k \bmod q]$
  - Output $\sigma = (r, s)$
- $\mathsf{Verify}_{pk}(m, \sigma)$:
  - Set $I = g^s \cdot y^{-r}$
  - Output 1 if $H(I, m) = r$

Security: Secure based on DLOG in random oracle model

Digital Signature Algorithm (DSA)

- Standard DSA algorithm uses similar paradigm, achieves same security