Privacy Enhancing Technologies (BSc)

Lecture IV: Cryptographic Preliminaries

Univ.-Prof. Dr. Dominique Schröder

November 04, 2024

Overview

KW	Type	Lectures
42	**Lecture**	Internet and Privacy in the Web
43	**Lecture**	Fingerprinting & Censorship
44	**Exercise**	Introduction to Rust & Challenge Start
45	**Lecture**	**Cryptographic Preliminaries**
46	**Lecture**	Public Key Cryptography and Security Reductions
47	**Exercise**	Proofs and Theoretical Deepening
48	**Lecture**	TLS
49	**Exam**	**Midterm Exam**
50	**Exercise**	Challenge Hands-On
51	**Lecture**	Messaging I
3	**Lecture**	Messaging II (Forward Security, Post Compromise Security)
4	**Lecture**	Onion
5	**Lecture**	Q & A
6	**Exam**	**Final Exam**
9	**Exam**	**Reexam** (Both for Midterm and Final)

This Lecture's Agenda

What are the following cryptography primitives? When do we know they are secure?

- Symmetric Encryption and CPA security
- Pseudorandom Functions
- Digital Signatures
- Hash Functions

This Lecture's Agenda

What are the following cryptography primitives? When do we know they are secure?

- Symmetric Encryption and CPA security
- Pseudorandom Functions
- Digital Signatures
- Hash Functions

This Lecture's Agenda

What are the following cryptography primitives? When do we know they are secure?

- Symmetric Encryption and CPA security
- Pseudorandom Functions
- Digital Signatures
- Hash Functions

This Lecture's Agenda

What are the following cryptography primitives? When do we know they are secure?

- Symmetric Encryption and CPA security
- Pseudorandom Functions
- Digital Signatures
- Hash Functions

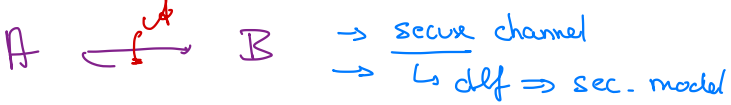
Literature

- J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, Chapman and Hall/CRC, 2014

Online?)

- O. Goldreich, *Foundations of Cryptography: Volume 1, Basic Tools*, Cambridge University Press, 2001

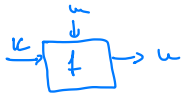
Computationally Secure Encryption



How would you encrypt a message m given an encryption key k?

Encryption

Intuition

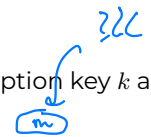


AES ← Instance

- We need a function f that takes an encryption key k and a message m as inputs.
  - The function f outputs a ciphertext c.
  - Without the correct key k, it should be infeasible for any party with access to c to decrypt the message.
  - With both the key k and the ciphertext c, computing f^{-1} to retrieve the original message should be straightforward.

Encryption

Intuition



- We need a function f that takes an encryption key k and a message m as inputs.
- The function f outputs a ciphertext c.
- Without the correct key k, it should be infeasible for any party with access to c to decrypt the message.
- With both the key k and the ciphertext c, computing f^{-1} to retrieve the original message should be straightforward.

Encryption

Intuition

- We need a function f that takes an encryption key k and a message m as inputs.
- The function f outputs a ciphertext c.
- Without the correct key k it should be infeasible for any party with access to c to decrypt the message.
- With both the key k and the ciphertext c, computing f^{-1} to retrieve the original message should be straightforward.

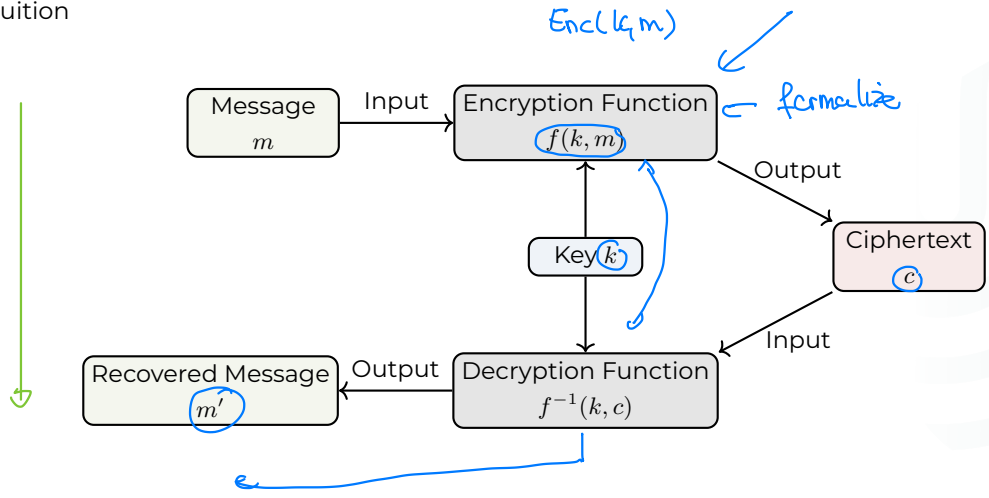
Encryption

Intuition

- We need a function f that takes an encryption key k and a message m as inputs.
- The function f outputs a ciphertext c.
- Without the correct key k, it should be infeasible for any party with access to c to decrypt the message.
- With both the key k and the ciphertext c, computing f^{-1} to retrieve the original message should be straightforward.

Encryption

Intuition



Private-Key Encryption

A Formal Definition

Definition (Private-Key Encryption Scheme)

A private-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is defined by a message space $\mathcal{M} = \{0,1\}^*$ along with the following three probabilistic polynomial time algorithms:

$k \leftarrow \mathsf{KGen}(1^\lambda)$: The key generation algorithm takes as input the security parameter 1^λ and outputs a key k chosen according to some distribution.

$c \leftarrow \mathsf{Enc}(k, m)$: The encryption algorithm takes as input the key k and a plaintext message $m \in \{0,1\}^*$ and outputs a ciphertext $c \in \mathcal{C}$. By $\mathsf{Enc}(k, m)$ we denote the encryption of the plaintext m using the key k.

$m = \mathsf{Dec}(k, c)$: The decryption algorithm takes as input a key k and a ciphertext c and outputs some plaintext m. We denote the decryption of the ciphertext c using the key k by $\mathsf{Dec}(k, c)$.

Private-Key Encryption

A Formal Definition

Definition (Private-Key Encryption Scheme)

A private-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is defined by a message space $\mathcal{M} = \{0,1\}^*$ along with the following three probabilistic polynomial time algorithms:

$k \leftarrow \mathsf{KGen}(1^\lambda)$: The key generation algorithm takes as input the security parameter 1^λ and outputs a key k chosen according to some distribution.

$c \leftarrow \mathsf{Enc}(k, m)$: The encryption algorithm takes as input the key k and a plaintext message $m \in \{0,1\}^*$ and outputs a ciphertext $c \in C$. By $\mathsf{Enc}(k, m)$ we denote the encryption of the plaintext m using the key k.

$m = \mathsf{Dec}(k, c)$: The decryption algorithm takes as input a key k and a ciphertext c and outputs some plaintext m. We denote the decryption of the ciphertext c using the key k by $\mathsf{Dec}(k, c)$.

Private-Key Encryption

A Formal Definition

Definition (Private-Key Encryption Scheme)

A private-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is defined by a message space $\mathcal{M} = \{0,1\}^*$ along with the following three probabilistic polynomial time algorithms:

$k \leftarrow \mathsf{KGen}(1^\lambda)$: The key generation algorithm takes as input the security parameter 1^λ and outputs a key k chosen according to some distribution.

$c \leftarrow \mathsf{Enc}(k, m)$: The encryption algorithm takes as input the key k and a plaintext message $m \in \{0,1\}^*$ and outputs a ciphertext $c \in \mathcal{C}$. By $\mathsf{Enc}(k, m)$ we denote the encryption of the plaintext m using the key k.

$m = \mathsf{Dec}(k, c)$: The decryption algorithm takes as input a key k and a ciphertext c and outputs some plaintext m. We denote the decryption of the ciphertext c using the key k by $\mathsf{Dec}(k, c)$.

Private-Key Encryption

Correctness = expected functionality

An encryption scheme must satisfy the following **(perfect) correctness** requirement:

Definition (Correctness of Private-Key Encryption Schemes)

For every security parameter $\lambda \in \mathbb{N}$, every key k output by $\mathsf{KGen}(1^\lambda)$ and every message $m \in \{0,1\}^*$, it holds that

$$\mathsf{Dec}(k, \mathsf{Enc}(k, m)) = m.$$

→ no collision

When do we consider our encryption scheme to be secure?

- C reveals no (new) inf. about m
- w/o k no information about m (trivial)
- k is hard to compute (correct key)
- large key space
- tamper proof (no modifications of m in c) → CCA security
- ⋮

many dif. sec. properties

Security Against Chosen-Plaintext Attacks

Cryptographic Security Definitions

Our Problem:

OTP $c = m \oplus k$ inf. theor. sec.

- Perfect, unbreakable security is generally infeasible due to computational limits. Practical security definitions balance robust guarantees with efficiency.
- Considering adversaries in security definitions is essential to create resilient schemes. Realistic adversaries may have varied access to data and can adapt their strategies to test for vulnerabilities.

Our Solution:

- Strike a middle ground that offers strong protection while remaining computationally practical.
- Clearly define specific security goals to outline exactly what we aim to achieve.
- Specify the adversary's capabilities, detailing their access, resources, and potential influence on the scheme.

Cryptographic Security Definitions

Our Problem:

- Perfect, unbreakable security is generally infeasible due to computational limits. Practical security definitions balance robust guarantees with efficiency.
- Considering adversaries in security definitions is essential to create resilient schemes. Realistic adversaries may have varied access to data and can adapt their strategies to test for vulnerabilities.

→ poly-time

Our Solution:

- Strike a middle ground that offers strong protection while remaining computationally practical.
- Clearly define specific security goals to outline exactly what we aim to achieve.
- Specify the adversary's capabilities, detailing their access, resources, and potential influence on the scheme.

Cryptographic Security Definitions

Our Problem:

- Perfect, unbreakable security is generally infeasible due to computational limits. Practical security definitions balance robust guarantees with efficiency.
- Considering adversaries in security definitions is essential to create resilient schemes. Realistic adversaries may have varied access to data and can adapt their strategies to test for vulnerabilities.

Our Solution:

- Strike a middle ground that offers strong protection while remaining computationally practical.
- Clearly define specific security goals to outline exactly what we aim to achieve.
- Specify the adversary's capabilities, detailing their access, resources, and potential influence on the scheme.

Cryptographic Security Definitions

HDF, SHA, ...

Our Problem:

- Perfect, unbreakable security is generally infeasible due to computational limits. Practical security definitions balance robust guarantees with efficiency.
- Considering adversaries in security definitions is essential to create resilient schemes. Realistic adversaries may have varied access to data and can adapt their strategies to test for vulnerabilities.

Our Solution:

- Strike a middle ground that offers strong protection while remaining computationally practical.
- Clearly define specific security goals to outline exactly what we aim to achieve.
- Specify the adversary's capabilities, detailing their access, resources, and potential influence on the scheme.

prov. sec. crypto does (not) get broken

Cryptographic Security Definitions

Our Problem:

- Perfect, unbreakable security is generally infeasible due to computational limits. Practical security definitions balance robust guarantees with efficiency.
- Considering adversaries in security definitions is essential to create resilient schemes. Realistic adversaries may have varied access to data and can adapt their strategies to test for vulnerabilities.

Our Solution:

- Strike a middle ground that offers strong protection while remaining computationally practical.
- Clearly define specific security goals to outline exactly what we aim to achieve.
- Specify the adversary's capabilities, detailing their access, resources, and potential influence on the scheme.

Cryptographic Security Definitions

Towards a Formalism

We break down security definitions into components that clarify the specific goals and adversarial assumptions. These core components are security goals and adversarial models.

- **Security Goal:** The desired outcome, such as confidentiality, integrity, or authenticity, that the scheme aims to uphold.
- **Adversarial Model:** The assumptions made about the adversary's capabilities, including their access to encrypted data, ability to choose plaintexts and computational resources.

Cryptographic Security Definitions

Towards a Formalism

We break down security definitions into components that clarify the specific goals and adversarial assumptions. These core components are security goals and adversarial models.

1. **Security Goal:** The desired outcome, such as confidentiality, integrity, or authenticity, that the scheme aims to uphold.
2. **Adversarial Model:** The assumptions made about the adversary's capabilities, including their access to encrypted data, ability to choose plaintexts and computational resources.
   - close as possible to the reality

Security Against Chosen-Plaintext Attacks

Our Security Goal: We want to prevent any efficient adversary from extracting meaningful information about the plaintext from the ciphertext.

- Specifically, the adversary should be unable to deduce even a single bit of information about the plaintext from the ciphertext.

Our Adversarial Model:

- The encryption scheme must remain secure even if the adversary can access chosen plaintext encryptions before and after targeting the actual ciphertext.

Security Against Chosen-Plaintext Attacks

Our Security Goal: We want to prevent any efficient adversary from extracting meaningful information about the plaintext from the ciphertext.

- Specifically, the adversary should be unable to deduce even a single bit of information about the plaintext from the ciphertext.

Our Adversarial Model:

- The encryption scheme must remain secure even if the adversary can access chosen plaintext encryptions before and after targeting the actual ciphertext.

Security Against Chosen-Plaintext Attacks

Practical Relevance

- An attacker interacts with a terminal that sends encrypted messages to a server, potentially testing various inputs to analyze encryption responses.
- Historical example: During World War II, the British strategically placed mines at specific locations, anticipating that the Germans would report these locations back to their headquarters using encrypted messages. This allowed the British to intercept and analyze these encrypted communications.

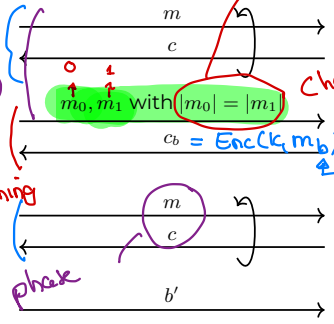
Security Against Chosen-Plaintext Attacks

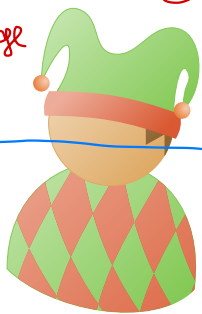
Practical Relevance

- An attacker interacts with a terminal that sends encrypted messages to a server, potentially testing various inputs to analyze encryption responses.
- Historical example: During World War II, the British strategically placed mines at specific locations, anticipating that the Germans would report these locations back to their headquarters using encrypted messages. This allowed the British to intercept and analyze these encrypted communications.

Security Against Chosen-Plaintext Attacks

We consider an adversary that is allowed to ask for encryptions of multiple messages chosen adaptively.





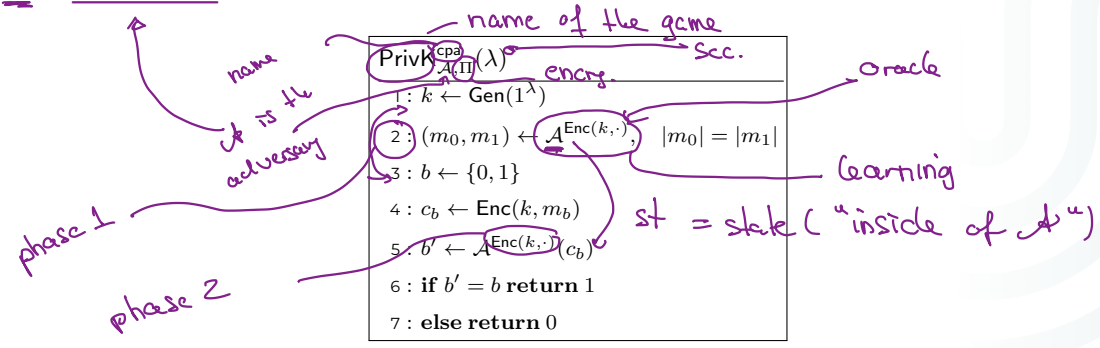
Adversary

Challenger

Adversary wins if $b = b'$ with probability $\geq \frac{1}{2} + non - negl.$

Security Against Chosen-Plaintext Attacks

We define an *experiment*, a game between an adversary and an imaginary challenger. Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme.



We write $\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(\lambda) = 1$ if the output is 1 and we say that $\mathcal{A}$ succeeded.

Security Against Chosen-Plaintext Attacks

A Formal Definition

The game $\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(\lambda)$ induces the following definition:

Definition (CPA security)

An encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ over a message space $\mathcal{M}$ has indistinguishable encryptions under a chosen-plaintext attack (or is CPA-secure) if for every PPT adversary $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

non-negl fraction

The notation $\mathcal{A}^{\mathsf{Enc}(k,\cdot)}$ means that the adversary $\mathcal{A}$ has access to an encryption oracle. The oracle encrypts any message of $\mathcal{A}$'s choice without revealing the secret key k to $\mathcal{A}$. The adversary can use the oracle as often as he wants to within his running time.

Pseudorandom Functions (building block)

Pseudorandom Functions (PRFs)

Idea:

- We aim for a keyed, deterministic function that behaves like a truly random function (consistent random sampling) as long as the key remains unknown.

Benefits:

- Allows for a random input of fixed size to efficiently generate a larger, quasi-random output.
- Provides a means to efficiently “blind” a message, which is foundational for encryption.

Applications:

- Beyond encryption, PRFs are essential in cryptographic protocols for tasks such as digital signatures, Message Authentication Codes (MACs), and more.

Pseudorandom Functions (PRFs)

Idea:

- We aim for a keyed, deterministic function that behaves like a truly random function (consistent random sampling) as long as the key remains unknown.

Benefits:

- Allows for a random input of fixed size to efficiently generate a larger, quasi-random output.
- Provides a means to efficiently “blind” a message, which is foundational for encryption.

Applications:

- Beyond encryption, PRFs are essential in cryptographic protocols for tasks such as digital signatures, Message Authentication Codes (MACs), and more.

Pseudorandom Functions (PRFs)

Idea:

- We aim for a keyed, deterministic function that behaves like a truly random function (consistent random sampling) as long as the key remains unknown.

Benefits:

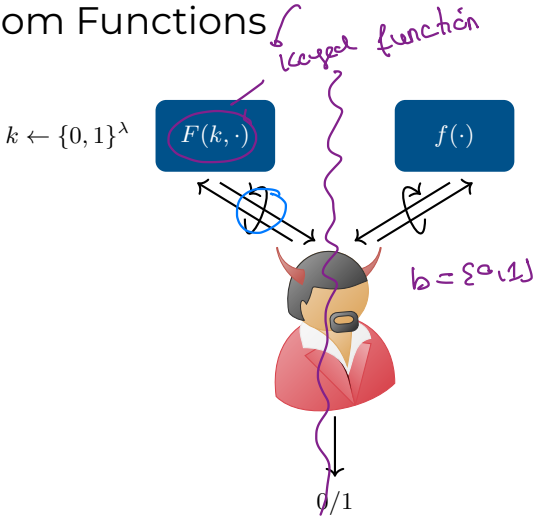
- Allows for a random input of fixed size to efficiently generate a larger, quasi-random output.
- Provides a means to efficiently “blind” a message, which is foundational for encryption.

Applications:

- Beyond encryption, PRFs are essential in cryptographic protocols for tasks such as digital signatures, Message Authentication Codes (MACs), and more.

Pseudorandom Functions

An Inutition



Pseudorandom Functions

A Formal Definition

Let $Func_n = \{f | f : \{0,1\}^n \to \{0,1\}^n\}$ be the set of all functions mapping n-bit strings to n-bit strings.

Definition (Pseudorandom Functions)

Let $F : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^n$ be an efficient, length-preserving, keyed function. F is a pseudorandom function (PRF) if for all PPT distinguishers D, there exists a negligible function negl such that

$$\left|\Pr\left[D^{F(k,\cdot)}(1^\lambda) = 1\right] - \Pr\left[D^{f(\cdot)}(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda),$$

where the first probability is taken over uniform choice of $k \in \{0,1\}^\lambda$ and the randomness of D, and the second probability is taken over uniform choice of $f \in Func_n$ and the randomness of D.

Pseudorandom Functions

Examples

Let F_k be a PRF. Are the following PRFs as well?

- $F^1(k, x) := F(k, x)||0$

PRF concat

→ NOT A PRF, x_0 yello, x_2, yello, ---- (PRF)

yllb ← not fixed (F)

- $F^2(k, x) := F(k, x||0)$

this is a PRF

Signature Schemes

Signature Schemes

A Definition

Definition (Digital signature scheme)

A digital signature scheme Π_{DS} consists of three PPT algorithms $(\mathsf{KGen}_{\mathsf{DS}}, \mathsf{Sign}, \mathsf{Vrfy})$ that are defined as follows:

$(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KGen}_{\mathsf{DS}}(1^\lambda)$: The key generation algorithm outputs a pair of keys $(\mathsf{vk}, \mathsf{sk})$, where vk is the verification key and sk the corresponding signing key.

$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$: The sign algorithm on input a signing key sk and a message m outputs a signature σ.

$0/1 \leftarrow \mathsf{Vrfy}(\mathsf{vk}, m, \sigma)$: The verification algorithm on input a verification key vk, a message m, and a signature σ deterministically outputs a bit b, with $b = 1$ meaning valid and $b = 0$ meaning invalid.

Signature Schemes

A Definition

Definition (Digital signature scheme)

A digital signature scheme Π_{DS} consists of three PPT algorithms $(\mathsf{KGen}_{\mathsf{DS}}, \mathsf{Sign}, \mathsf{Vrfy})$ that are defined as follows:

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}_{\mathsf{DS}}(1^\lambda)$: The key generation algorithm outputs a pair of keys $(\mathsf{vk}, \mathsf{sk})$, where vk is the verification key and sk the corresponding signing key.

$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$: The sign algorithm on input a signing key sk and a message m outputs a signature σ.

$0/1 \leftarrow \mathsf{Vrfy}(\mathsf{vk}, m, \sigma)$: The verification algorithm on input a verification key vk, a message m, and a signature σ deterministically outputs a bit b, with $b = 1$ meaning valid and $b = 0$ meaning invalid.

Signature Schemes

A Definition

Definition (Digital signature scheme)

A digital signature scheme Π_{DS} consists of three PPT algorithms $(\mathsf{KGen}_{\mathsf{DS}}, \mathsf{Sign}, \mathsf{Vrfy})$ that are defined as follows:

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}_{\mathsf{DS}}(1^\lambda)$: The key generation algorithm outputs a pair of keys $(\mathsf{vk}, \mathsf{sk})$, where vk is the verification key and sk the corresponding signing key.

$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$: The sign algorithm on input a signing key sk and a message m outputs a signature σ.

$0/1 \leftarrow \mathsf{Vrfy}(\mathsf{vk}, m, \sigma)$: The verification algorithm on input a verification key vk, a message m, and a signature σ deterministically outputs a bit b, with $b = 1$ meaning valid and $b = 0$ meaning invalid.

Signature Schemes

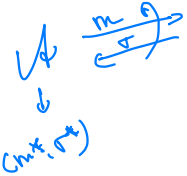
Correctness

Definition (Correctness of signature schemes)

A signature scheme $\Pi_{\mathsf{DS}} = (\mathsf{KGen}_{\mathsf{DS}}, \mathsf{Sign}, \mathsf{Vrfy})$ is correct, if for every security paramter $\lambda \in \mathbb{N}$, any $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KGen}_{\mathsf{DS}}(1^\lambda)$, any message $m \in \mathcal{M}$, and any $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$, it holds that $\mathsf{Vrfy}(\mathsf{vk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1$.

Signature Schemes

Existential Unforgeability



Security experiment for existential unforgeability under a chosen message attack of the signature scheme DS. The set Q contains the messages that were signed by the signing oracle. We omit a formal definition and refer to the reference.

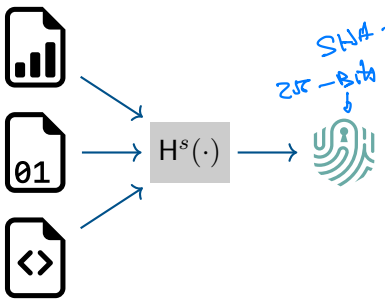
We have a signature scheme Π_{DS} that can sign messages of length λ, how can we efficiently sign messages of length λ^2?

Hash Functions

Hash Functions

Intuition

Hash functions take data of any size and transform it into a fixed-size, unique “fingerprint” or “digest.”

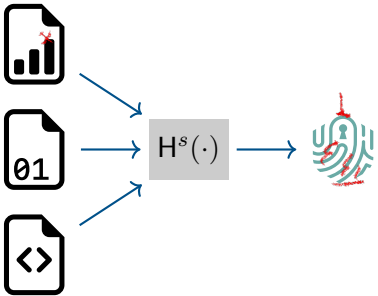


- **Efficiency:** Quickly condenses large data into a manageable format.
- **Data Integrity:** Any small change in input data results in a completely different hash, allowing for easy detection of tampering.
- **Security:** Hard to reverse, making them useful for securing passwords and sensitive information.

Hash Functions

Intuition

Hash functions take data of any size and transform it into a fixed-size, unique “fingerprint” or “digest.”

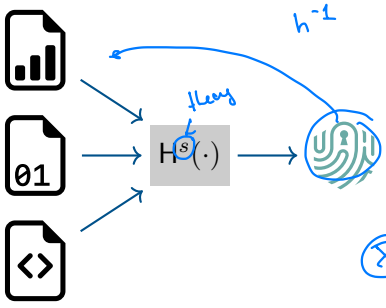


- **Efficiency:** Quickly condenses large data into a manageable format.
- **Data Integrity:** Any small change in input data results in a completely different hash, allowing for easy detection of tampering.
- **Security:** Hard to reverse, making them useful for securing passwords and sensitive information.

Hash Functions

Intuition

Hash functions take data of any size and transform it into a fixed-size, unique “fingerprint” or “digest.”



- **Efficiency:** Quickly condenses large data into a manageable format.
- **Data Integrity:** Any small change in input data results in a completely different hash, allowing for easy detection of tampering.
- **Security:** Hard to reverse, making them useful for securing passwords and sensitive information.

Hash Functions

Applications

- **Data Storage and Retrieval:** Used in hash tables for efficient data indexing and fast lookups.
- **Digital Signatures:** Ensures the integrity and authenticity of documents and messages.
- **Password Security:** Passwords are hashed and stored securely to protect user information.
- **Blockchain and Cryptocurrencies:** Used to verify transactions and maintain the integrity of blockchain data.
- **File Integrity Checks:** Commonly used to verify downloaded files are intact and unaltered.

Hash Functions

Applications

- **Data Storage and Retrieval:** Used in hash tables for efficient data indexing and fast lookups.
- **Digital Signatures:** Ensures the integrity and authenticity of documents and messages.
- **Password Security:** Passwords are hashed and stored securely to protect user information.
- **Blockchain and Cryptocurrencies:** Used to verify transactions and maintain the integrity of blockchain data.
- **File Integrity Checks:** Commonly used to verify downloaded files are intact and unaltered.

Hash Functions

Applications

- **Data Storage and Retrieval:** Used in hash tables for efficient data indexing and fast lookups.
- **Digital Signatures:** Ensures the integrity and authenticity of documents and messages.
- **Password Security:** Passwords are hashed and stored securely to protect user information.
- **Blockchain and Cryptocurrencies:** Used to verify transactions and maintain the integrity of blockchain data.
- **File Integrity Checks:** Commonly used to verify downloaded files are intact and unaltered.

Hash Functions

Applications

- **Data Storage and Retrieval:** Used in hash tables for efficient data indexing and fast lookups.
- **Digital Signatures:** Ensures the integrity and authenticity of documents and messages.
- **Password Security:** Passwords are hashed and stored securely to protect user information.
- **Blockchain and Cryptocurrencies:** Used to verify transactions and maintain the integrity of blockchain data.
- **File Integrity Checks:** Commonly used to verify downloaded files are intact and unaltered.

Hash Functions

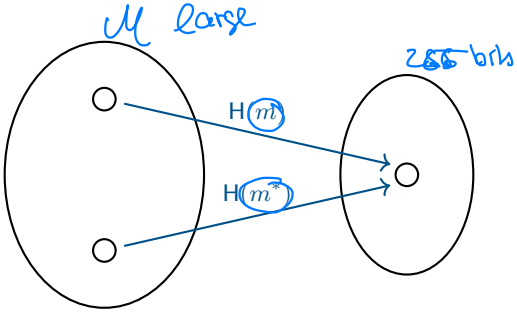
Applications

- **Data Storage and Retrieval:** Used in hash tables for efficient data indexing and fast lookups.
- **Digital Signatures:** Ensures the integrity and authenticity of documents and messages.
- **Password Security:** Passwords are hashed and stored securely to protect user information.
- **Blockchain and Cryptocurrencies:** Used to verify transactions and maintain the integrity of blockchain data.
- **File Integrity Checks:** Commonly used to verify downloaded files are intact and unaltered.

Collision Resistance

Intuition

Assume, we have a hash function $\mathsf{H} : \mathcal{M} \to \mathcal{R}$ with $|\mathcal{M}| >> |\mathcal{R}|$. By the pigeonhole principle, we will have collisions in this function. Where a collision means, that for $m \neq m^*$, we have $\mathsf{H}(m) = \mathsf{H}(m^*)$



Hash Functions

A Definition

Definition (Hash Function)

A hash function is a pair of PPT algorithms $(\mathsf{Gen}, \mathsf{H})$ satisfying the following:

$\mathsf{KGen}(1^\lambda)$: Output a key s (1^λ is implicit in s). (practice not true), security proofs

$\mathsf{H}(x)$: There exists a polynomial ℓ such that H takes as input a string $x \in \{0,1\}^*$ and outputs a string $\mathsf{H}^s(x) \in \{0,1\}^{\ell(n)}$.

Hash Functions

Remarks

- The notation $H^s(x)$ emphasizes that s is (generally) not kept secret.
- In practice, hash functions are usually unkeyed. In theory, this is problematic as one could hardcode a collision.
- However, you cannot hardcode a collision for every possible key when using a keyed hash function.
- In the real world, unkeyed hash functions are collision resistant in the sense that colliding pairs are unknown and computationally difficult to find.

Hash Functions

Remarks

- The notation $H^s(x)$ emphasizes that s is (generally) not kept secret.
- In practice, hash functions are usually unkeyed. In theory, this is problematic as one could hardcode a collision.
- However, you cannot hardcode a collision for every possible key when using a keyed hash function.
- In the real world, unkeyed hash functions are collision resistant in the sense that colliding pairs are unknown and computationally difficult to find.

Hash Functions

Remarks

- The notation $H^s(x)$ emphasizes that s is (generally) not kept secret.
- In practice, hash functions are usually unkeyed. In theory, this is problematic as one could hardcode a collision.
- However, you cannot hardcode a collision for every possible key when using a keyed hash function.
- In the real world, unkeyed hash functions are collision resistant in the sense that colliding pairs are unknown and computationally difficult to find.

Hash Functions

Remarks

- The notation $H^s(x)$ emphasizes that s is (generally) not kept secret.
- In practice, hash functions are usually unkeyed. In theory, this is problematic as one could hardcode a collision.
- However, you cannot hardcode a collision for every possible key when using a keyed hash function.
- In the real world, unkeyed hash functions are collision resistant in the sense that colliding pairs are unknown and computationally difficult to find.

Collision Resistance

A Security Experiment

We define an experiment, where $\Pi = (\mathsf{Gen}, \mathsf{H})$ is a hash function.

$\mathsf{HashColl}_{\mathcal{A},\Pi}(\lambda)$

1 : $s \leftarrow \mathsf{Gen}(1^\lambda)$
2 : $(x_0, x_1) \leftarrow \mathcal{A}(s)$
3 : **if** $x_0 \neq x_1$ **and** $\mathsf{H}^s(x_0) = \mathsf{H}^s(x_1)$
4 : **return** 1
5 : **else return** 0

Winning condition

The adversary wins if $x_0 \neq x_1$ and $H^s(x_0) = H^s(x_1)$.

We write $\mathsf{HashColl}_{\mathcal{A},\Pi}(n) = 1$ if the output is 1 and we say that $\mathcal{A}$ succeeded.

Collision Resistance

A Definition

Definition (Collision Resistance)

A hash function $\Pi = (\mathsf{Gen}, \mathsf{H})$ is collision resistant, if for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that

$$\Pr[\mathsf{HashColl}_{\mathcal{A},\Pi}(\lambda) = 1] \leq \mathsf{negl}(\lambda).$$

super small

$x_0, x_1 \in \{0,1\}^*$
return $x_0 \neq x_1$

Instances

The Schnorr Signature Scheme (Claus Peter Schnorr)

The Schnorr Signature Scheme (1)

What is the Schnorr Signature Scheme?

- Developed by Claus Schnorr in the 1980s, the Schnorr Signature Scheme is a digital signature method.
- Based on the **Discrete Logarithm Problem (DLP)**, it's designed to securely authenticate digital messages.
- Known for its **efficiency and simplicity**, the scheme is particularly lightweight in terms of computational requirements.

Why is it Important?

- **Efficiency**: Fewer operations are needed compared to similar schemes (e.g., DSA), making it faster and resource-friendly.
- **Provable Security**: Its security ties directly to the difficulty of the DLP, a widely accepted cryptographic assumption.
- **Compact Signatures**: Produces smaller signatures, which is beneficial for storage and transmission, especially in blockchain applications.

The Schnorr Signature Scheme (1)

What is the Schnorr Signature Scheme?

- Developed by Claus Schnorr in the 1980s, the Schnorr Signature Scheme is a digital signature method.
- Based on the **Discrete Logarithm Problem (DLP)**, it's designed to securely authenticate digital messages.
- Known for its **efficiency and simplicity**, the scheme is particularly lightweight in terms of computational requirements.

Why is it Important?

- **Efficiency**: Fewer operations are needed compared to similar schemes (e.g., DSA), making it faster and resource-friendly.
- **Provable Security**: Its security ties directly to the difficulty of the DLP, a widely accepted cryptographic assumption.
- **Compact Signatures**: Produces smaller signatures, which is beneficial for storage and transmission, especially in blockchain applications.

The Schnorr Signature Scheme (1)

What is the Schnorr Signature Scheme?

- Developed by Claus Schnorr in the 1980s, the Schnorr Signature Scheme is a digital signature method.
- Based on the **Discrete Logarithm Problem (DLP)**, it's designed to securely authenticate digital messages.
- Known for its **efficiency and simplicity**, the scheme is particularly lightweight in terms of computational requirements.

Why is it Important?

- **Efficiency**: Fewer operations are needed compared to similar schemes (e.g., DSA), making it faster and resource-friendly.
- **Provable Security**: Its security ties directly to the difficulty of the DLP, a widely accepted cryptographic assumption.
- **Compact Signatures**: Produces smaller signatures, which is beneficial for storage and transmission, especially in blockchain applications.

The Schnorr Signature Scheme (1)

What is the Schnorr Signature Scheme?

- Developed by Claus Schnorr in the 1980s, the Schnorr Signature Scheme is a digital signature method.
- Based on the **Discrete Logarithm Problem (DLP)**, it's designed to securely authenticate digital messages.
- Known for its **efficiency and simplicity**, the scheme is particularly lightweight in terms of computational requirements.

Why is it Important?

- **Efficiency**: Fewer operations are needed compared to similar schemes (e.g., DSA), making it faster and resource-friendly.
- **Provable Security**: Its security ties directly to the difficulty of the DLP, a widely accepted cryptographic assumption.
- **Compact Signatures**: Produces smaller signatures, which is beneficial for storage and transmission, especially in blockchain applications.

The Schnorr Signature Scheme (1)

What is the Schnorr Signature Scheme?

- Developed by Claus Schnorr in the 1980s, the Schnorr Signature Scheme is a digital signature method.
- Based on the **Discrete Logarithm Problem (DLP)**, it's designed to securely authenticate digital messages.
- Known for its **efficiency and simplicity**, the scheme is particularly lightweight in terms of computational requirements.

Why is it Important?

- **Efficiency**: Fewer operations are needed compared to similar schemes (e.g., DSA), making it faster and resource-friendly.
- **Provable Security**: Its security ties directly to the difficulty of the DLP, a widely accepted cryptographic assumption.
- **Compact Signatures**: Produces smaller signatures, which is beneficial for storage and transmission, especially in blockchain applications.

The Schnorr Signature Scheme (1)

What is the Schnorr Signature Scheme?

- Developed by Claus Schnorr in the 1980s, the Schnorr Signature Scheme is a digital signature method.
- Based on the **Discrete Logarithm Problem (DLP)**, it's designed to securely authenticate digital messages.
- Known for its **efficiency and simplicity**, the scheme is particularly lightweight in terms of computational requirements.

Why is it Important?

- **Efficiency**: Fewer operations are needed compared to similar schemes (e.g., DSA), making it faster and resource-friendly.
- **Provable Security**: Its security ties directly to the difficulty of the DLP, a widely accepted cryptographic assumption.
- **Compact Signatures**: Produces smaller signatures, which is beneficial for storage and transmission, especially in blockchain applications.

The Schnorr Signature Scheme (2)

Where is it Used?

- **Cryptocurrencies**: Often used in blockchain and cryptocurrency platforms like Bitcoin to verify transactions efficiently.
- **IoT and Embedded Systems**: Due to its low computational overhead, Schnorr is suitable for resource-constrained environments.
- **Privacy-Enhancing Protocols**: Employed in Zero-Knowledge Proofs and privacy-focused applications, thanks to its flexibility in cryptographic constructions.

The Schnorr Signature Scheme (2)

Where is it Used?

- **Cryptocurrencies**: Often used in blockchain and cryptocurrency platforms like Bitcoin to verify transactions efficiently.
- **IoT and Embedded Systems**: Due to its low computational overhead, Schnorr is suitable for resource-constrained environments.
- **Privacy-Enhancing Protocols**: Employed in Zero-Knowledge Proofs and privacy-focused applications, thanks to its flexibility in cryptographic constructions.

The Schnorr Signature Scheme (2)

Where is it Used?

- **Cryptocurrencies**: Often used in blockchain and cryptocurrency platforms like Bitcoin to verify transactions efficiently.
- **IoT and Embedded Systems**: Due to its low computational overhead, Schnorr is suitable for resource-constrained environments.
- **Privacy-Enhancing Protocols**: Employed in Zero-Knowledge Proofs and privacy-focused applications, thanks to its flexibility in cryptographic constructions.

proof correctness w/o leaking anything

Schnorr Signature Scheme

Construction (For formal security proof, read the reference)

Defines parameter

commitment

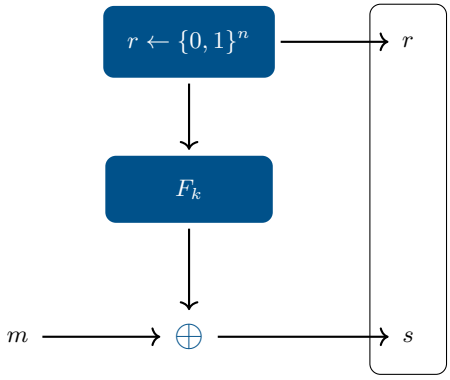
Setup(λ)	KGen(λ)	Sign(sk, m)	Vrfy(vk, m, σ)
1 : $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^\lambda)$	1 : $\mathsf{sk} \leftarrow\$ \mathbb{Z}_q$	1 : $r \leftarrow\$ \mathbb{Z}_p$	1 : $(R, s) := \sigma$
2 : $\mathsf{param} := (\mathbb{G}, q, g)$	2 : $\mathsf{vk} \leftarrow g^{\mathsf{sk}}$	2 : $R \leftarrow g^r$	2 : **return** $g^s \stackrel{?}{=} R \cdot \mathsf{vk}^h$
3 : **return** param	3 : **return** $(\mathsf{sk}, \mathsf{vk})$	3 : $h \leftarrow \mathsf{H}(\mathsf{vk}, R, m)$	
		4 : $s \leftarrow \mathsf{sk} \cdot h + r$	
		5 : **return** (R, s)	

randomizer that blinds

CPA-Secure Encryption from PRFs

CPA-Secure Encryption from PRFs

Construction Idea



CPA-Secure Encryption from PRFs

Formal Construction

Let F be a function. We define a private-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ for messages of length n as follows:

Construction

$\mathsf{Gen}(1^\lambda)$: Return a key $k \leftarrow \{0,1\}^\lambda$ of size λ chosen uniformly at random.

$\mathsf{Enc}(k, m)$: Choose $r \leftarrow \{0,1\}^\lambda$ uniformly at random and compute

$$c = (r, s) = (r, F(k, r) \oplus m).$$

$\mathsf{Dec}(k, c)$: Parse c as (r, s) and output the plaintext message computing

$$m = F(k, r) \oplus s.$$

CPA-Secure Encryption from PRFs

Formal Construction

Let F be a function. We define a private-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ for messages of length n as follows:

Construction

$\mathsf{Gen}(1^\lambda)$: Return a key $k \leftarrow \{0,1\}^\lambda$ of size λ chosen uniformly at random.

$\mathsf{Enc}(k, m)$: Choose $r \leftarrow \{0,1\}^\lambda$ uniformly at random and compute

$$c = (r, s) = (r, F(k, r) \oplus m).$$

$\mathsf{Dec}(k, c)$: Parse c as (r, s) and output the plaintext message computing

$$m = F(k, r) \oplus s.$$

CPA-Secure Encryption from PRFs

Formal Construction

Let F be a function. We define a private-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ for messages of length n as follows:

Construction

$\mathsf{Gen}(1^\lambda)$: Return a key $k \leftarrow \{0,1\}^\lambda$ of size λ chosen uniformly at random.

$\mathsf{Enc}(k, m)$: Choose $r \leftarrow \{0,1\}^\lambda$ uniformly at random and compute

$$c = (r, s) = (r, F(k, r) \oplus m).$$

$\mathsf{Dec}(k, c)$: Parse c as (r, s) and output the plaintext message computing

$$m = F(k, r) \oplus s.$$

Proof – CPA-Secure Encryption from PRFs

Theorem

If F is a pseudorandom function, then the previous construction is a CPA-secure private-key encryption scheme for messages of length λ.

How do we show security?

Reduction Proofs

Reduction Proofs

Say we want to prove the following theorem with a proof by reduction.

Theorem

Let H be an known primitive. From that we construct a new primitive G, which uses H in some way and has some specific properties.

What do we want to achieve?

We assume that there is an efficient adversary $\mathcal{A}$ against the new primitive G. From that we construct an efficient adversary $\mathcal{B}$ against the known primitive H, which uses $\mathcal{A}$ to break the primitive H. Because we know, that the primitive H has to hold, this is a contradiction. We can conclude, that an adversary $\mathcal{A}$ against the new primitive G can't exist, so this new primitive G has to fulfill the required properties.

Reduction Proofs

Say we want to prove the following theorem with a proof by reduction.

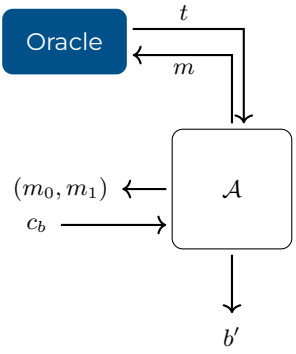
Theorem

Let H be an known primitive. From that we construct a new primitive G, which uses H in some way and has some specific properties.

What do we want to achieve?

We assume that there is an efficient adversary $\mathcal{A}$ against the new primitive G. From that we construct an efficient adversary $\mathcal{B}$ against the known primitive H, which uses $\mathcal{A}$ to break the primitive H. Because we know, that the primitive H has to hold, this is a contradiction. We can conclude, that an adversary $\mathcal{A}$ against the new primitive G can't exist, so this new primitive G has to fulfill the required properties.

Reduction Proofs – $\mathcal{A}$'s World

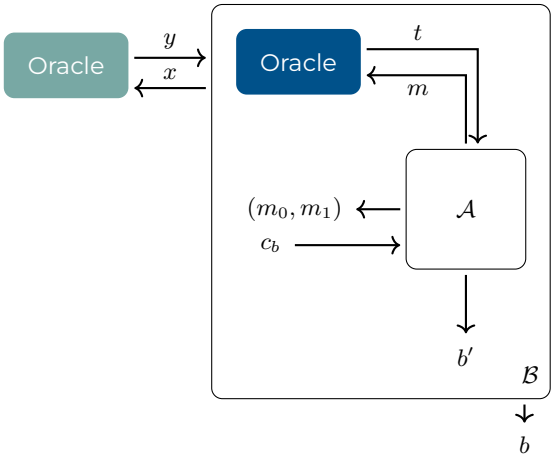


An overview of how the reduction works.

How to construct the reduction?

- We start with the *security game* of the known primitive H recalling the expected interfaces for adversary $\mathcal{A}$.
- These interfaces *may* include decryption/encryption oracles, or those outputting a PRF/PRG or their random equivalents.
- The adversary *may* interact directly with the challenger, with in/outgoing messages.
- The figure beside illustrates this setup for the CPA game discussed before.

Reduction Proofs – $\mathcal{B}$ Simulates the Game for $\mathcal{A}$

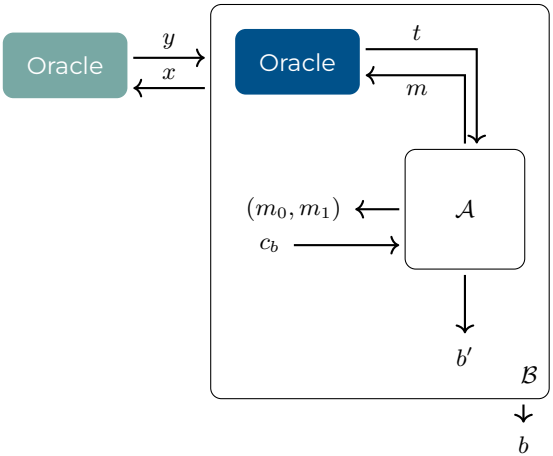


An overview of how the reduction works.

How to construct the reduction?

- $\mathcal{B}$'s goal is to solve the underlying hard problem.
- $\mathcal{B}$ *might* get an instance of this problem as input and *possibly* access to oracles.
- $\mathcal{B}$ simulates *all* interfaces for $\mathcal{A}$ such that $\mathcal{A}$ does not realize that is executed within another algorithm.
- The figure beside illustrates this setup, showing necessary oracles and I/O between $\mathcal{A}$ and $\mathcal{B}$ for the security reduction to a PRF.

Reduction Proofs



An overview of how the reduction works.

What now?

- We need to show, how $\mathcal{B}$ can break the secure primitive (or assumption) H, whenever there exists an adversary $\mathcal{A}$, that breaks G (that is the hard part).
- If we do so, no efficient adversary $\mathcal{A}$ against G can exist, since otherwise the secure primitive (or assumption) H can also be broken.
- We conclude, that G is secure.

Reduction Proofs

Warning

- Security reductions are highly exam-relevant.
- We will practice security reductions in the upcoming exercise sessions.
- In addition, you must practice on your own and ask us if you get lost.
- You will not pass this course if you do not understand security reductions.

We expect that you read about security reductions in the book of Katz and Lindell.

Reduction Proofs

An example (2)

Theorem

Let $F : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^n$ a pseudorandom function. Then the function $F'(k,x) := F(k,x||0)$ is a pseudorandom function as well.

We want to prove this theorem with a proof by reduction.

Reduction Proofs

An example (2)

Feedback

Your *anonymous feedback* is invaluable to us, and we are truly grateful for your time and insights.

Thank you for helping us improve!

https://tuwel.tuwien.ac.at/mod/feedback/view.php?id=2451827

PRIVACY
ENHANCING
TECHNOLOGIES