Concurrent Signatures

Samir Benzammour

Algorithms and Computational Complexity
RWTH Aachen University

27th April 2020

Outline

Outline

Introduction

What is a Fair Signature Exchange and why important?

Fair Signature Exchange (Example)

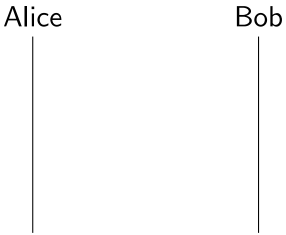
- Two parties A(lice) and B(ob)

Fair Signature Exchange (Example)

- Two parties A(lice) and B(ob)
- Want to sign an item (e.g. a contract)

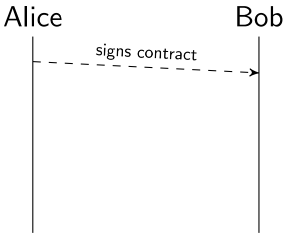
Fair Signature Exchange (Example)

- Two parties A(lice) and B(ob)
- Want to sign an item (e.g. a contract)



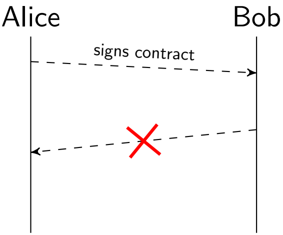
Fair Signature Exchange (Example)

- Two parties A(lice) and B(ob)
- Want to sign an item (e.g. a contract)



Fair Signature Exchange (Example)

- Two parties A(lice) and B(ob)
- Want to sign an item (e.g. a contract)



Introduction

- Which approaches already exist?

Introduction

- Which approaches already exist?
- Is *truly fair* exchange necessary?

Concurrent Approach

- A creates ambiguous signature with random bits and B's public key

Concurrent Approach

- A creates ambiguous signature with random bits and B's public key
- random bits through a hash function

Concurrent Approach

- A creates ambiguous signature with random bits and B's public key
- random bits through a hash function
- B creates its own signature with those *same* bits and A's public key

Concurrent Approach

- A creates ambiguous signature with random bits and B's public key
- random bits through a hash function
- B creates its own signature with those *same* bits and A's public key
- Until input of hash function is released both signatures stay ambiguous

Concurrent Approach

- A creates ambiguous signature with random bits and B's public key
- random bits through a hash function
- B creates its own signature with those *same* bits and A's public key
- Until input of hash function is released both signatures stay ambiguous
  - denoted as: *keystone*

Outline

Concurrent Scheme

Definition (Concurrent Signature Scheme)

A **concurrent signature scheme** is a digital signature scheme, that holds the following algorithms

- **SETUP**
- **ASIGN**
- **AVERIFY**
- **VERIFY**

Concurrent Scheme

Definition (Concurrent Signature Scheme)

A **concurrent signature scheme** is a digital signature scheme, that holds the following algorithms

- **SETUP**
- **ASIGN**
- **AVERIFY**
- **VERIFY**

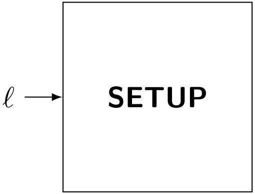
Notation

Let k denote the *keystone*

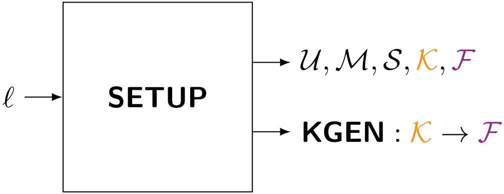
Algorithms

$\boxed{\textbf{SETUP}}$

Algorithms



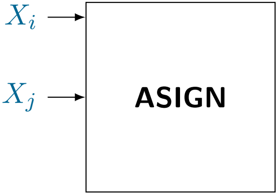
Algorithms


ℓ
SETUP
$\mathcal{U}, \mathcal{M}, \mathcal{S}, \mathcal{K}, \mathcal{F}$
KGEN : $\mathcal{K} \to \mathcal{F}$

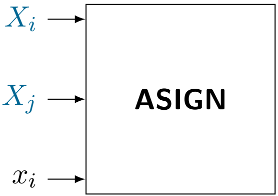
Algorithms

ASIGN

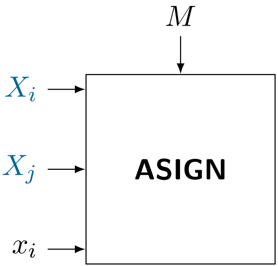
Algorithms



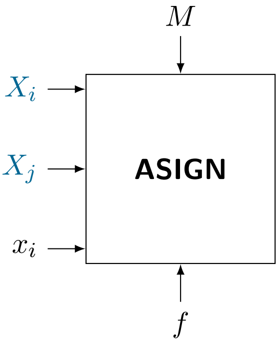
Algorithms



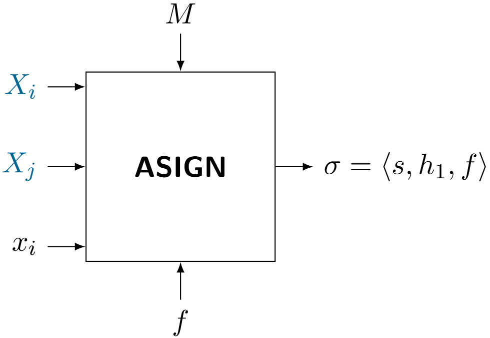
Algorithms



Algorithms



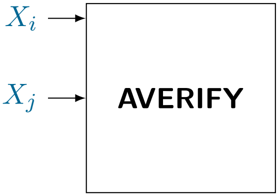
Algorithms



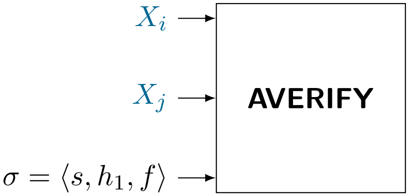
Algorithms

AVERIFY

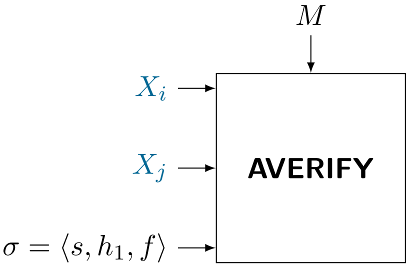
Algorithms



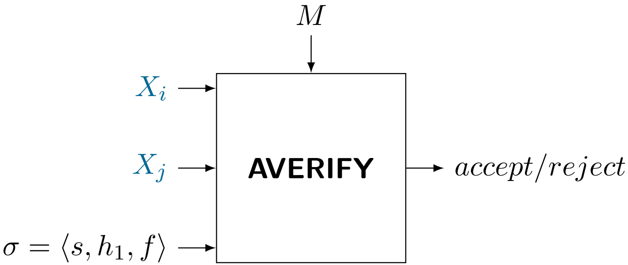
Algorithms



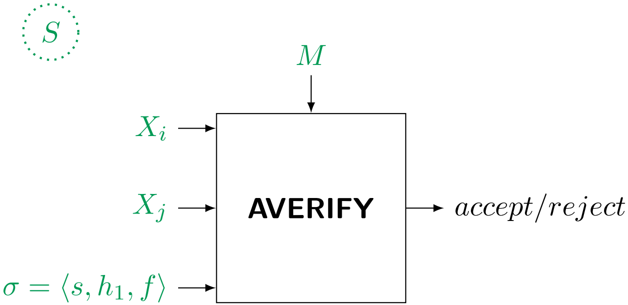
Algorithms



Algorithms



Algorithms

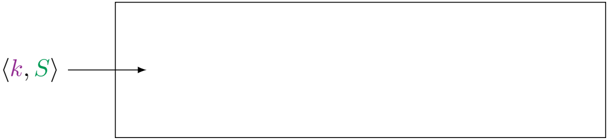


Algorithms

VERIFY

Algorithms

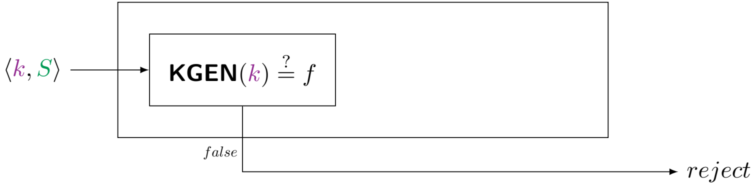
VERIFY



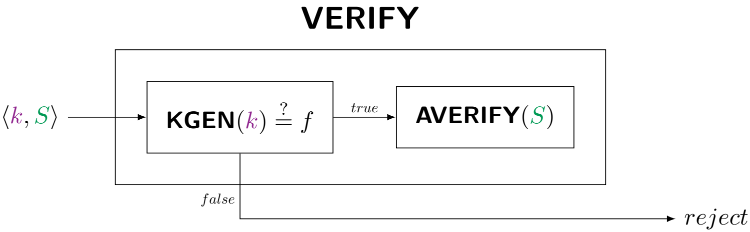
VERIFY

$\langle k, S \rangle \longrightarrow \boxed{\textbf{KGEN}(k) \stackrel{?}{=} f}$

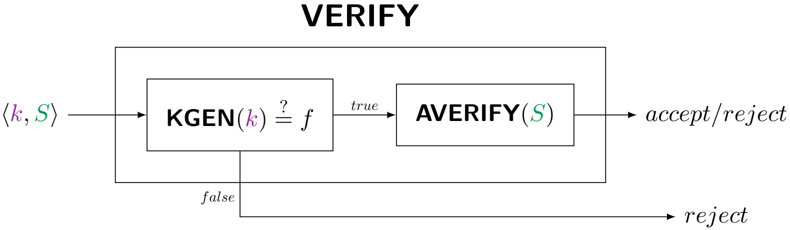
VERIFY



Algorithms


VERIFY
$\langle k, S \rangle$
$\textbf{KGEN}(k) \stackrel{?}{=} f$
$true$
$\textbf{AVERIFY}(S)$
$false$
$reject$

Algorithms


VERIFY
$\langle k, S\rangle$
$\textbf{KGEN}(k) \stackrel{?}{=} f$
$true$
$\textbf{AVERIFY}(S)$
$accept/reject$
$false$
$reject$

Concurrent Signature Protocol

- Describing signature exchange between A(lice) and B(ob)

Concurrent Signature Protocol

- Describing signature exchange between A(lice) and B(ob)
- Initial signer has to generate keystone

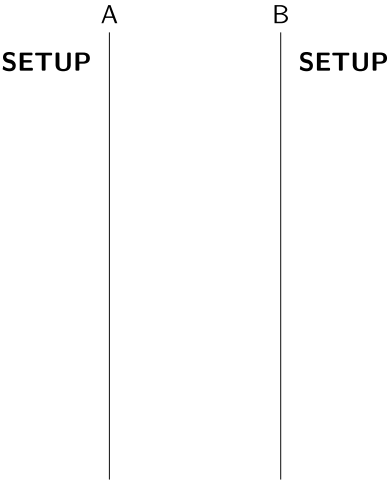
Concurrent Signature Protocol

- Describing signature exchange between A(lice) and B(ob)
- Initial signer has to generate keystone
- Matching signer responds by using the same keystone-fix

Concurrent Signature Protocol

- Describing signature exchange between A(lice) and B(ob)
- Initial signer has to generate keystone
- Matching signer responds by using the same keystone-fix
- both parties *ambiguous* until k released

Concurrent Signature Protocol



Concurrent Signature Protocol

A	B

SETUP | **SETUP**

$k \in \mathcal{K},\ f = \textbf{KGEN}(k)$

Concurrent Signature Protocol

A	B
SETUP	**SETUP**
$k \in \mathcal{K}$, $f = \textbf{KGEN}(k)$	
$\textbf{ASIGN}(X_A, X_B, x_A, f, M_A)$	

Concurrent Signature Protocol

A B

SETUP **SETUP**

$k \in \mathcal{K}$, $f = \textbf{KGEN}(k)$

$\textbf{ASIGN}(X_A, X_B, x_A, f, M_A)$
$= \langle s_A, h_A, f \rangle =: \sigma_A$

Concurrent Signature Protocol

A | B

SETUP | **SETUP**

$k \in \mathcal{K},\ f = \textbf{KGEN}(k)$

$\textbf{ASIGN}(X_A, X_B, x_A, f, M_A)$
$= \langle s_A, h_A, f \rangle =: \sigma_A$

$A \xrightarrow{\sigma_A} B$

Concurrent Signature Protocol

A		B
SETUP		**SETUP**
$k \in \mathcal{K}$, $f = \textbf{KGEN}(k)$		
$\textbf{ASIGN}(X_A, X_B, x_A, f, M_A)$ $= \langle s_A, h_A, f \rangle =: \sigma_A$	$\xrightarrow{\sigma_A}$	$\textbf{AVERIFY}(\sigma_A, X_A, X_B, M_A)$

Concurrent Signature Protocol

A		B
SETUP		**SETUP**
$k \in \mathcal{K},\ f = \textbf{KGEN}(k)$		
$\textbf{ASIGN}(X_A, X_B, x_A, f, M_A) = \langle s_A, h_A, f \rangle =: \sigma_A$	$\xrightarrow{\sigma_A}$	$\textbf{AVERIFY}(\sigma_A, X_A, X_B, M_A)$

Concurrent Signature Protocol

A		B
SETUP		**SETUP**
$k \in \mathcal{K},\ f = \textbf{KGEN}(k)$		
$\textbf{ASIGN}(X_A, X_B, x_A, f, M_A) = \langle s_A, h_A, f \rangle =: \sigma_A$	$\xrightarrow{\sigma_A}$	$\textbf{AVERIFY}(\sigma_A, X_A, X_B, M_A)$
		$\textbf{ASIGN}(X_B, X_A, x_B, f, M_B)$

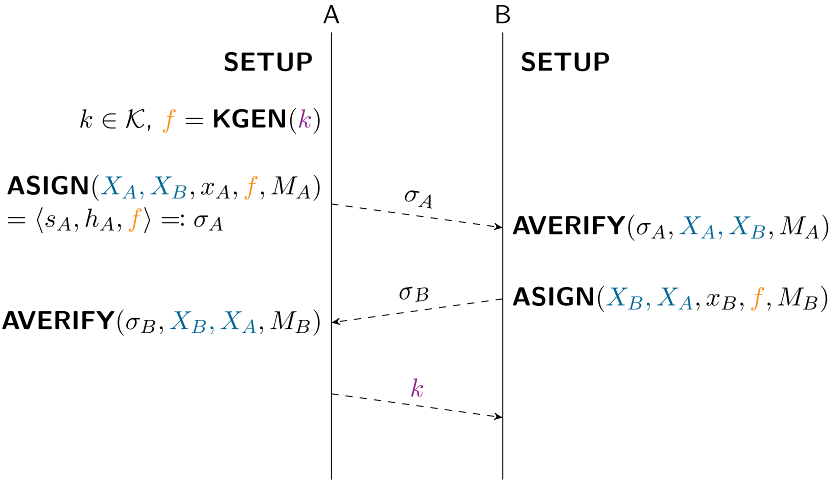
Concurrent Signature Protocol

A	B
SETUP	**SETUP**
$k \in \mathcal{K},\ f = \textbf{KGEN}(k)$	
$\textbf{ASIGN}(X_A, X_B, x_A, f, M_A) = \langle s_A, h_A, f \rangle =: \sigma_A$ $\xrightarrow{\sigma_A}$	$\textbf{AVERIFY}(\sigma_A, X_A, X_B, M_A)$
$\xleftarrow{\sigma_B}$	$\textbf{ASIGN}(X_B, X_A, x_B, f, M_B)$

Concurrent Signature Protocol

A		B
SETUP		**SETUP**
$k \in \mathcal{K},\ f = \textbf{KGEN}(k)$		
$\textbf{ASIGN}(X_A, X_B, x_A, f, M_A) = \langle s_A, h_A, f \rangle =: \sigma_A$	$\xrightarrow{\sigma_A}$	$\textbf{AVERIFY}(\sigma_A, X_A, X_B, M_A)$
$\textbf{AVERIFY}(\sigma_B, X_B, X_A, M_B)$	$\xleftarrow{\sigma_B}$	$\textbf{ASIGN}(X_B, X_A, x_B, f, M_B)$

Concurrent Signature Protocol



Outline

Security Model

Definition

- formalizes security properties for a given system

Security Model

Definition

- formalizes security properties for a given system
- defines the adversaries power in said system

Security Model

Definition

- formalizes security properties for a given system
- defines the adversaries power in said system
- if all properties are given, system is called *secure*

Security Model

Definition

- formalizes security properties for a given system
- defines the adversaries power in said system
- if all properties are given, system is called *secure*

we formalize security through

Security Model

Definition

- formalizes security properties for a given system
- defines the adversaries power in said system
- if all properties are given, system is called *secure*

we formalize security through

- Fairness
- Unforgeability
- Ambiguity

Security Model

- consider game with
  - adversary E
  - challenger C

Security Model

- consider game with
  - adversary E
  - challenger C
- C runs **SETUP** and publishes all public keys

Security Model

- consider game with
  - adversary E
  - challenger C
- C runs **SETUP** and publishes all public keys
- E can request

$$\left.\begin{array}{l} \\ \\ \\ \end{array}\right\} \ldots \text{Queries from } C$$

Security Model

- consider game with
  - adversary E
  - challenger C
- C runs **SETUP** and publishes all public keys
- E can request
  - **KGen...**

$\left.\begin{array}{l} \\ \\ \\ \end{array}\right\}$ ... Queries from C

Security Model

- consider game with
  - adversary E
  - challenger C
- C runs **SETUP** and publishes all public keys
- E can request
  - **KGen...**
  - **KReveal...**

$\left.\begin{array}{l} \\ \\ \\ \end{array}\right\}$ ... Queries from C

Security Model

- consider game with
  - adversary E
  - challenger C
- C runs **SETUP** and publishes all public keys
- E can request

$$\left.\begin{array}{l}\bullet\ \textbf{KGen...}\\ \bullet\ \textbf{KReveal...}\\ \bullet\ \textbf{ASign...}\end{array}\right\}\text{... Queries from } C$$

Security Model

- consider game with
  - adversary E
  - challenger C
- C runs **SETUP** and publishes all public keys
- E can request
  - **KGen...**
  - **KReveal...**
  - **ASign...**
  - **AVerify / Verify...**

$\left.\begin{array}{l} \\ \\ \\ \\ \end{array}\right\}$ ...Queries from C

Security Model

- consider game with
  - adversary E
  - challenger C
- C runs **SETUP** and publishes all public keys
- E can request
  - **KGen. . .**
  - **KReveal. . .**
  - **ASign. . .**
  - **AVerify / Verify. . .**
  - **Private Key Extraction. . .**

. . . Queries from C

Fairness

- adversary E can query everything except **AVERIFY** and **VERIFY** queries

Fairness

- adversary E can query everything except **AVERIFY** and **VERIFY** queries
- E returns keystone k and a valid signature $S = \langle \sigma, X_c, X_d, M \rangle$

Fairness

- adversary E can query everything except **AVERIFY** and **VERIFY** queries
- E returns keystone k and a valid signature $S = \langle \sigma, X_c, X_d, M \rangle$
- adversary wins if one of the following holds

Fairness

- adversary E can query everything except **AVERIFY** and **VERIFY** queries
- E returns keystone k and a valid signature $S = \langle \sigma, X_c, X_d, M \rangle$
- adversary wins if one of the following holds
  1. E created k s.t. signature is accepted by **VERIFY**

Fairness

- adversary E can query everything except **AVERIFY** and **VERIFY** queries
- E returns keystone k and a valid signature $S = \langle \sigma, X_c, X_d, M \rangle$
- adversary wins if one of the following holds
  1. E created k s.t. signature is accepted by **VERIFY**
  2. E creates additional valid signature S' with same keystone-fix

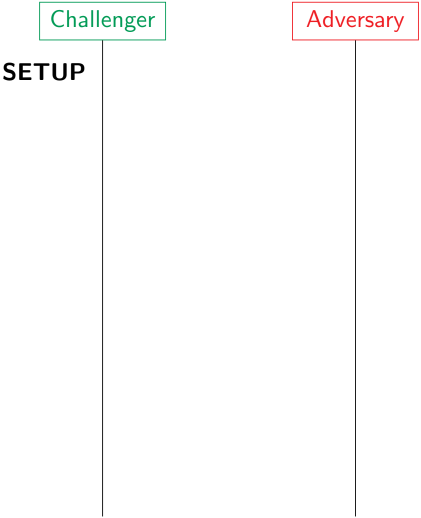
Fairness

- adversary E can query everything except **AVERIFY** and **VERIFY** queries
- E returns keystone k and a valid signature $S = \langle \sigma, X_c, X_d, M \rangle$
- adversary wins if one of the following holds
  1. E created k s.t. signature is accepted by **VERIFY**
  2. E creates additional valid signature S' with same keystone-fix
     - but only one of both is accepted by **VERIFY**

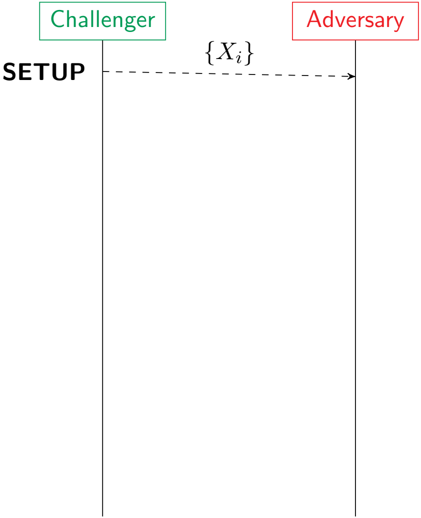
Fairness

- adversary E can query everything except **AVERIFY** and **VERIFY** queries
- E returns keystone k and a valid signature $S = \langle \sigma, X_c, X_d, M \rangle$
- adversary wins if one of the following holds
  1. E created k s.t. signature is accepted by **VERIFY**
  2. E creates additional valid signature S' with same keystone-fix
     - but only one of both is accepted by **VERIFY**
     - i.e. E creates signature with same keystone but is not bound by it

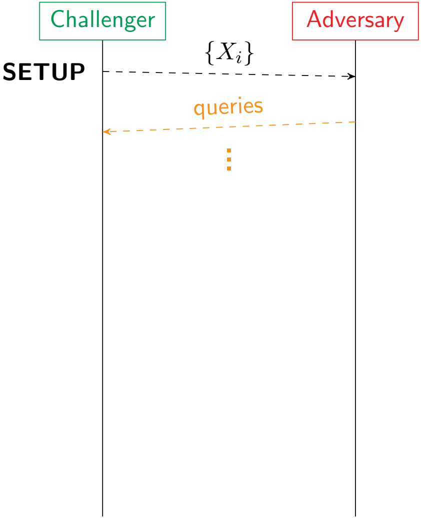
Ambiguity



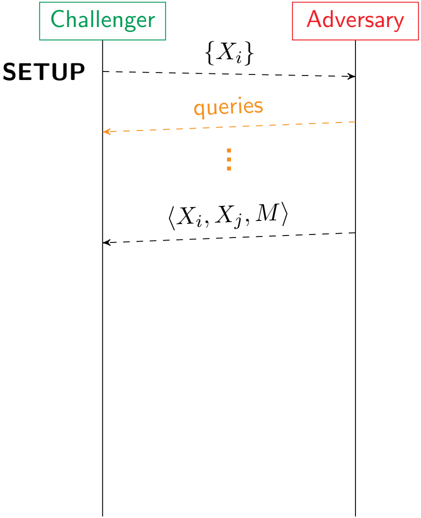
Ambiguity



Ambiguity

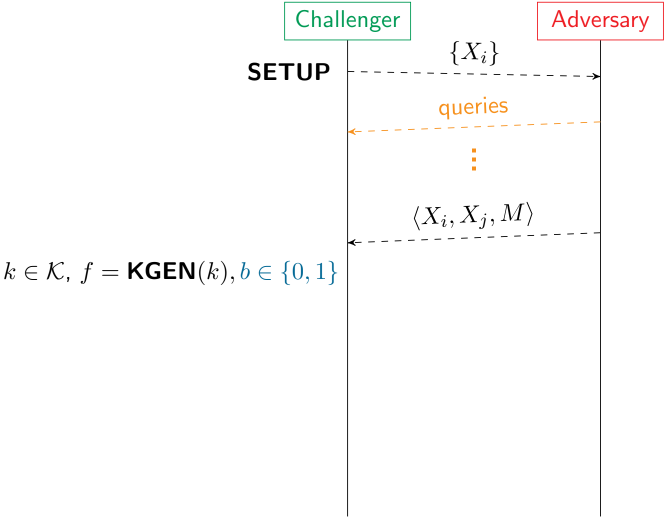


Ambiguity

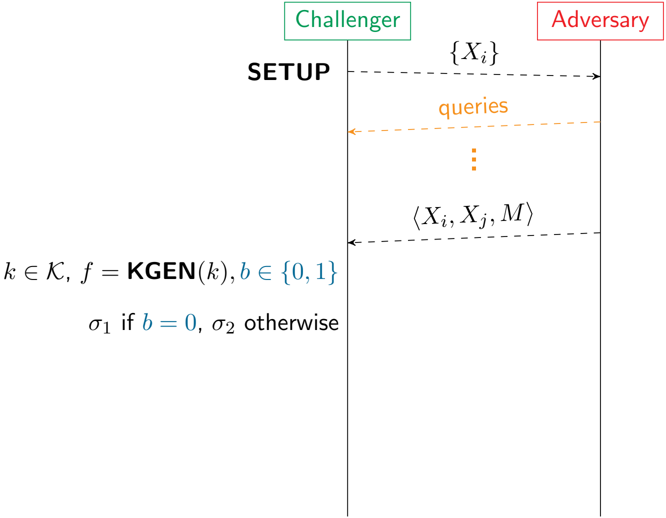


Challenger
Adversary
SETUP
$\{X_i\}$
queries
$\langle X_i, X_j, M \rangle$

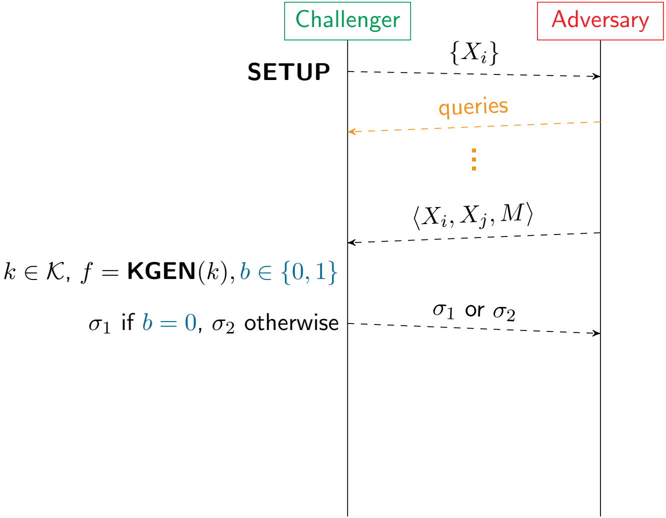
Ambiguity


Challenger
Adversary
SETUP
$\{X_i\}$
queries
$\langle X_i, X_j, M \rangle$
$k \in \mathcal{K},\ f = \textbf{KGEN}(k), b \in \{0,1\}$

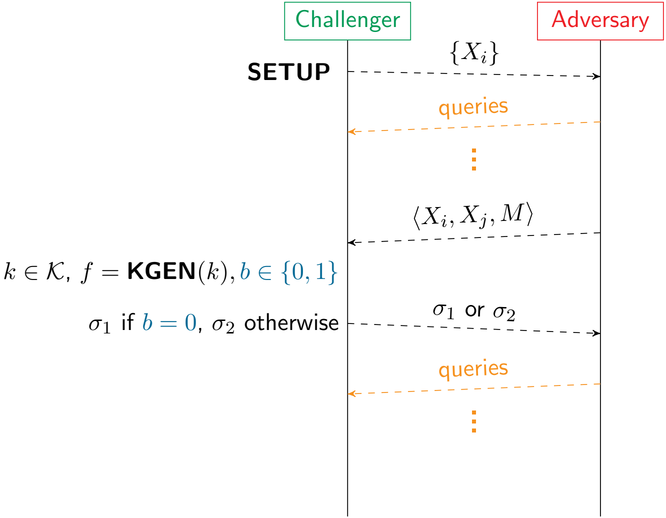
Ambiguity


Challenger
Adversary
SETUP
$\{X_i\}$
queries
$\langle X_i, X_j, M \rangle$
$k \in \mathcal{K},\ f = \textbf{KGEN}(k), b \in \{0, 1\}$
σ_1 if $b = 0$, σ_2 otherwise

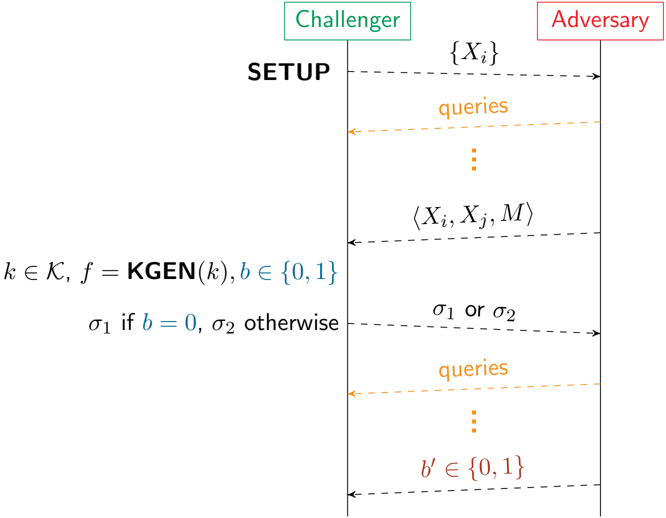
Ambiguity


Challenger
Adversary
SETUP
$\{X_i\}$
queries
$\langle X_i, X_j, M \rangle$
$k \in \mathcal{K},\ f = \textbf{KGEN}(k), b \in \{0,1\}$
σ_1 if $b = 0$, σ_2 otherwise
σ_1 or σ_2

Ambiguity

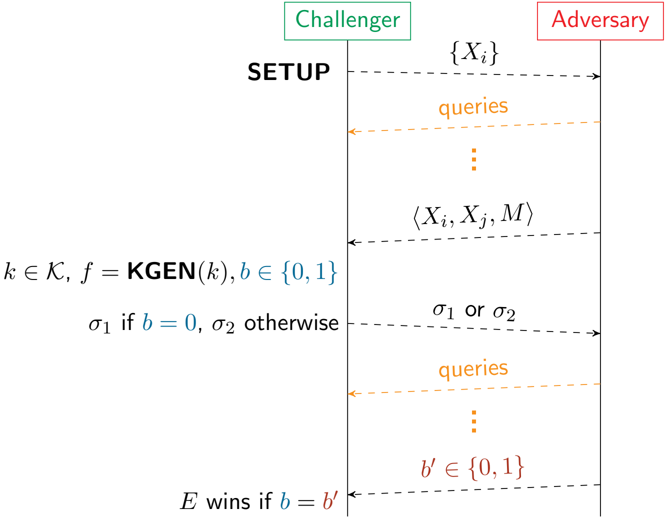

Challenger
Adversary
$\{X_i\}$
SETUP
queries
⋮
$\langle X_i, X_j, M \rangle$
$k \in \mathcal{K},\ f = \textbf{KGEN}(k), b \in \{0, 1\}$
σ_1 if $b = 0$, σ_2 otherwise
σ_1 or σ_2
queries
⋮

Ambiguity



Challenger
Adversary
SETUP
$\{X_i\}$
queries
$\langle X_i, X_j, M \rangle$
$k \in \mathcal{K},\ f = \textbf{KGEN}(k), b \in \{0,1\}$
σ_1 if $b = 0$, σ_2 otherwise
σ_1 or σ_2
queries
$b' \in \{0,1\}$

Ambiguity



Challenger
Adversary
SETUP
$\{X_i\}$
queries
$\langle X_i, X_j, M \rangle$
$k \in \mathcal{K},\ f = \textbf{KGEN}(k), b \in \{0,1\}$
σ_1 or σ_2
σ_1 if $b = 0$, σ_2 otherwise
queries
$b' \in \{0,1\}$
E wins if $b = b'$

Unforgeability

- consider previous adversary-challenger game

Unforgeability

- consider previous adversary-challenger game
- E can query everything

Unforgeability

- consider previous adversary-challenger game
- E can query everything
- eventually E outputs tuple $S = (\sigma, X_c, X_d, M)$

Unforgeability

- consider previous adversary-challenger game
- E can query everything
- eventually E outputs tuple $S = (\sigma, X_c, X_d, M)$
- E wins if **AVERIFY**$(S) = accept$ and no **Private Key Extraction** query was made and

Unforgeability

- consider previous adversary-challenger game
- E can query everything
- eventually E outputs tuple $S = (\sigma, X_c, X_d, M)$
- E wins if **AVERIFY**$(S) = accept$ and no **Private Key Extraction** query was made and
    1. No **ASIGN** query with our parameters, and no **Private Key Extraction** query for X_c or X_d

Unforgeability

- consider previous adversary-challenger game
- E can query everything
- eventually E outputs tuple $S = (\sigma, X_c, X_d, M)$
- E wins if **AVERIFY**$(S) = accept$ and no **Private Key Extraction** query was made and
  1. No **ASIGN** query with our parameters, and no **Private Key Extraction** query for X_c or X_d
  2. No **ASIGN** query with $\langle X_c, X_i, f, M \rangle$, no **Private Key Extraction** query for X_c

Outline

Discrete Logarithmic Problem

Definition (discrete logarithm)

- Group $\mathbb{G}$ given
- b^k always defined in $\mathbb{G}$ through $b^x = \underbrace{b \cdot b \cdots b}_{x \text{ times}}$
- the **discrete logarithm** is an integer x, such that $b^x = a$

Discrete Logarithmic Problem

Definition (discrete logarithm)

- Group $\mathbb{G}$ given
- b^k always defined in $\mathbb{G}$ through $b^x = \underbrace{b \cdot b \cdots b}_{x \text{ times}}$
- the **discrete logarithm** is an integer x, such that $b^x = a$

Definition (group generator)

Let $\mathbb{G}$ be a cyclic group of order p. Then $g \in \mathbb{G}$ is generator of $\mathbb{G}$ if

$$\mathbb{G} = \{g^i \bmod p \mid i \in \mathbb{N}\}$$

Concrete Scheme

- based on Ring Signatures

Concrete Scheme

- based on Ring Signatures
- **SETUP**

Concrete Scheme

- based on Ring Signatures
- **SETUP**
  - outputs two large primes p and q with $q \mid p-1$, and $g \in (\mathbb{Z}/p\mathbb{Z})^*$

Concrete Scheme

- based on Ring Signatures
- **SETUP**
  - outputs two large primes p and q with $q \mid p-1$, and $g \in (\mathbb{Z}/p\mathbb{Z})^*$
  - $\mathcal{S} \equiv \mathcal{F} = \mathbb{Z}_q$ and $\mathcal{M} \equiv \mathcal{K} = \{0,1\}^*$

Concrete Scheme

- based on Ring Signatures
- **SETUP**
  - outputs two large primes p and q with $q \mid p-1$, and $g \in (\mathbb{Z}/p\mathbb{Z})^*$
  - $\mathcal{S} \equiv \mathcal{F} = \mathbb{Z}_q$ and $\mathcal{M} \equiv \mathcal{K} = \{0,1\}^*$
  - H_1, $H_2 : \{0,1\}^* \to \mathbb{Z}_q$ cryptographic hash functions

Concrete Scheme

- based on Ring Signatures
- **SETUP**
  - outputs two large primes p and q with $q \mid p - 1$, and $g \in (\mathbb{Z}/p\mathbb{Z})^*$
  - $\mathcal{S} \equiv \mathcal{F} = \mathbb{Z}_q$ and $\mathcal{M} \equiv \mathcal{K} = \{0, 1\}^*$
  - H_1, $H_2 : \{0, 1\}^* \to \mathbb{Z}_q$ cryptographic hash functions
  - **KGEN** $:= H_1$

Concrete Scheme

- based on Ring Signatures
- **SETUP**
  - outputs two large primes p and q with $q \mid p-1$, and $g \in (\mathbb{Z}/p\mathbb{Z})^*$
  - $\mathcal{S} \equiv \mathcal{F} = \mathbb{Z}_q$ and $\mathcal{M} \equiv \mathcal{K} = \{0,1\}^*$
  - H_1, $H_2 : \{0,1\}^* \to \mathbb{Z}_q$ cryptographic hash functions
  - **KGEN** $:= H_1$
  - public keys $X_i = g^{x_i} \bmod p$ published

Concrete Scheme

- **ASIGN** : $\langle X_i, X_j, x_i, f, M \rangle$

Concrete Scheme

- **ASIGN** : $\langle X_i, X_j, x_i, f, M \rangle$
  - $h = H_2(g^t {X_j}^f \bmod p \parallel M)$ with $t \in \mathbb{Z}_q$ random

Concrete Scheme

- **ASIGN** : $\langle X_i, X_j, x_i, f, M \rangle$
  - $h = H_2(g^t {X_j}^f \bmod p \parallel M)$ with $t \in \mathbb{Z}_q$ random
  - $h_1 = h - f \bmod q$

Concrete Scheme

- **ASIGN** : $\langle X_i, X_j, x_i, f, M \rangle$
  - $h = H_2(g^t {X_j}^f \bmod p \parallel M)$ with $t \in \mathbb{Z}_q$ random
  - $h_1 = h - f \bmod q$
  - $s = t - h_1 x_i \bmod q$

Concrete Scheme

- **ASIGN** : $\langle X_i, X_j, x_i, f, M\rangle$
  - $h = H_2(g^t {X_j}^f \bmod p \parallel M)$ with $t \in \mathbb{Z}_q$ random
  - $h_1 = h - f \bmod q$
  - $s = t - h_1 x_i \bmod q$
- **AVERIFY** : $\langle\langle s, h_1, f\rangle, X_i, X_j, M\rangle$

Concrete Scheme

- **ASIGN** : $\langle X_i, X_j, x_i, f, M\rangle$
  - $h = H_2(g^t {X_j}^f \bmod p \parallel M)$ with $t \in \mathbb{Z}_q$ random
  - $h_1 = h - f \bmod q$
  - $s = t - h_1 x_i \bmod q$
- **AVERIFY** : $\langle\langle s, h_1, f\rangle, X_i, X_j, M\rangle$
  - returns: $h_1 + f \stackrel{?}{=} H_2(g^s {X_i}^{h_1} {X_j}^f \bmod p \parallel M) \bmod q$

Lemmas

Lemma (Fairness)

The concurrent signature scheme in our example is fair in the random oracle model

Lemma (Ambiguity)

The concurrent signature scheme in our example is ambiguous in the random oracle model

Lemma (Unforgeability)

The concurrent signature scheme in our example is existentially *unforgeable* under a *chosen message attack* in the random oracle model, assuming the hardness of the discrete logarithm problem

Random Oracle

Definition (Oracle)

An **oracle** (machine) is an abstract machine that takes an input and generates a solution for it without knowing its inner workings.

Random Oracle

Definition (Oracle)

An **oracle** (machine) is an abstract machine that takes an input and generates a solution for it without knowing its inner workings.

Definition (Random Oracle)

A **random oracle** is a oracle, which fullfills the following properties

- returns each *unique* request with a truly random value (chosen from output domain)
- repeated requests (always) return the same response

Definitions

Definition (Chosen Message Attack)

A chosen message attack allows the adversary to query the signature query with messages of her choice.

Definition (Negligible Function)

A negligible function, is a function negl such that

$$\mathrm{negl}(n) < \frac{1}{n^c}$$

for all sufficiently large n

Forking Lemma

applies to signature schemes with signature form $\langle r_1, h, r_2 \rangle$ on message M

Forking Lemma

applies to signature schemes with signature form $\langle r_1, h, r_2 \rangle$ on message M

Definition

If an algorithm E creates a signature $\langle r_1, h, r_2 \rangle$ from public data with non-negligible probability, there exists algorithm A which controls E and forces it to rerun to create a second valid signature $\langle r_1, h', r_2' \rangle$.

Unforgeability - Proof

Proof Idea:

1. Assumption: hardness of the discrete logarithm

Unforgeability - Proof

Proof Idea:

1. Assumption: hardness of the discrete logarithm
2. Proof by contraposition: Scheme is forgeable with an non-negligible probability

Unforgeability - Proof

Proof Idea:

1. Assumption: hardness of the discrete logarithm
2. Proof by contraposition: Scheme is forgeable with an non-negligible probability
3. Create system, that uses the adversary to solve the discrete logarithm problem

Proof - Setup

- need signature format $\langle r_1, h, r_2 \rangle$

Proof - Setup

- need signature format $\langle r_1, h, r_2 \rangle$
  - easily deriviable from $\sigma = \langle s, h_1, h_2 \rangle$

Proof - Setup

- need signature format $\langle r_1, h, r_2 \rangle$
  - easily deriviable from $\sigma = \langle s, h_1, h_2 \rangle$
- H_1, H_2 random oracles

Proof - Setup

- need signature format $\langle r_1, h, r_2 \rangle$
  - easily deriviable from $\sigma = \langle s, h_1, h_2 \rangle$
- H_1, H_2 random oracles
- E is forging-adversary

Proof - Setup

- need signature format $\langle r_1, h, r_2 \rangle$
  - easily deriviable from $\sigma = \langle s, h_1, h_2 \rangle$
- H_1, H_2 random oracles
- E is forging-adversary
- $\mathcal{A}$ is an algorithm that solves the discrete logarithm problem

Proof - Setup

- need signature format $\langle r_1, h, r_2 \rangle$
  - easily deriviable from $\sigma = \langle s, h_1, h_2 \rangle$
- H_1, H_2 random oracles
- E is forging-adversary
- $\mathcal{A}$ is an algorithm that solves the discrete logarithm problem
  - simulates random oracles and challenger C

Proof - Setup

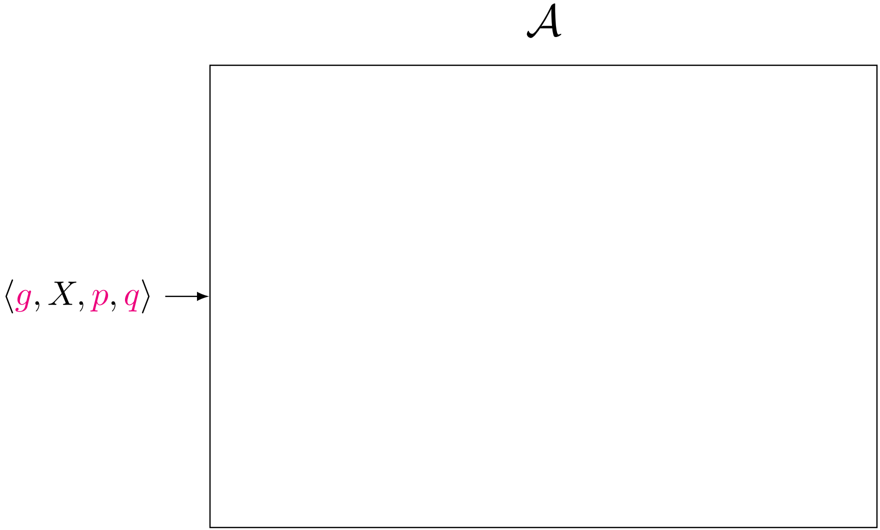
- need signature format $\langle r_1, h, r_2 \rangle$
  - easily deriviable from $\sigma = \langle s, h_1, h_2 \rangle$
- H_1, H_2 random oracles
- E is forging-adversary
- $\mathcal{A}$ is an algorithm that solves the discrete logarithm problem
  - simulates random oracles and challenger C
  - input: $\langle g, X, p, q \rangle$

Proof - Setup

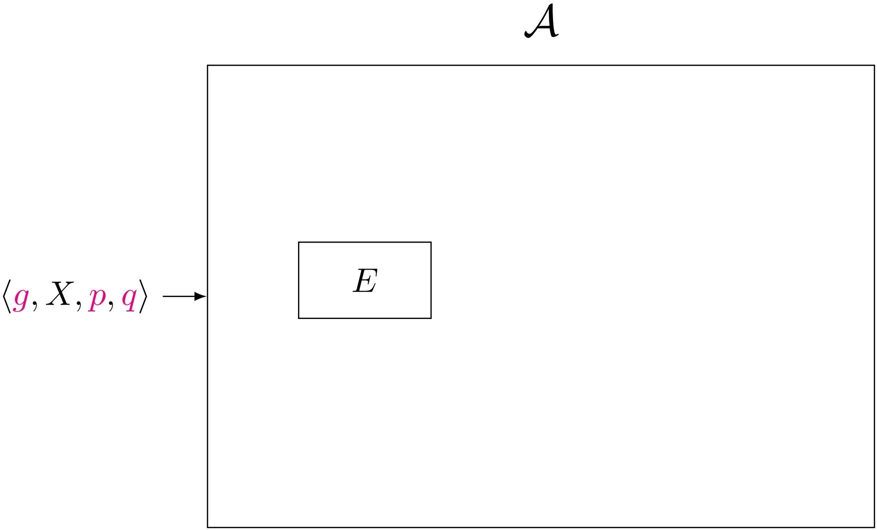
- need signature format $\langle r_1, h, r_2 \rangle$
  - easily deriviable from $\sigma = \langle s, h_1, h_2 \rangle$
- H_1, H_2 random oracles
- E is forging-adversary
- $\mathcal{A}$ is an algorithm that solves the discrete logarithm problem
  - simulates random oracles and challenger C
  - input: $\langle g, X, p, q \rangle$
  - goal: find $x \in \mathbb{Z}_q$ s.t. $g^x = X \bmod p$

Proof - Simulation

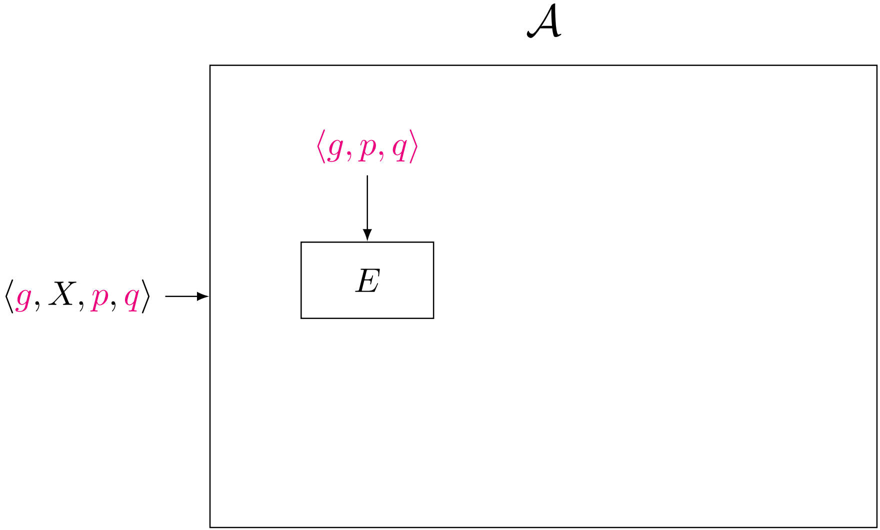
$\mathcal{A}$


$\mathcal{A}$
$\langle g, X, p, q \rangle$

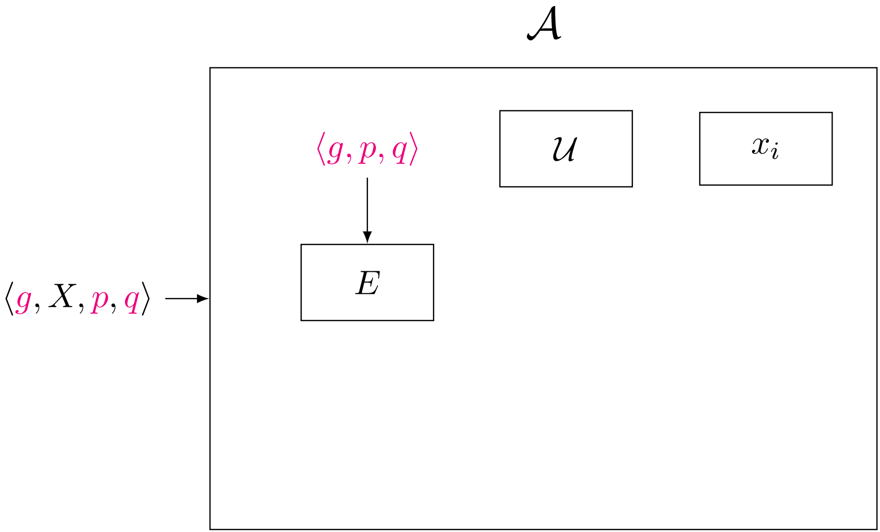
Proof - Simulation



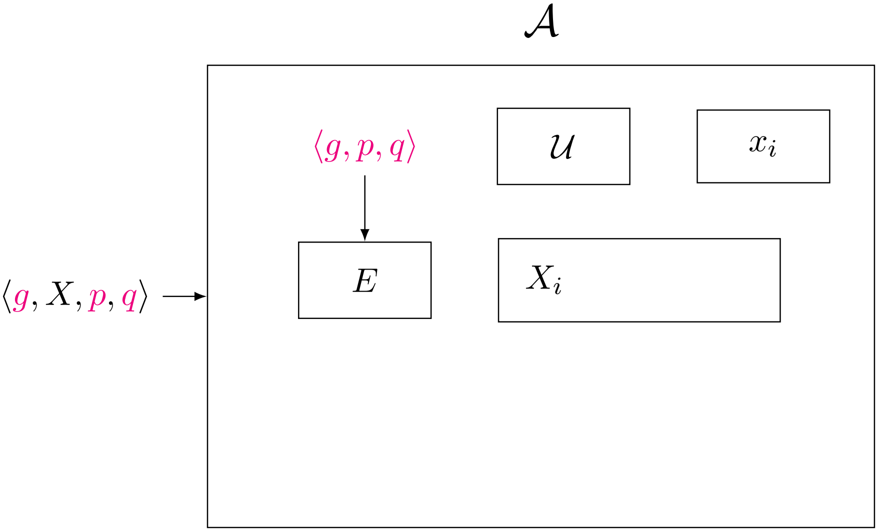
Proof - Simulation



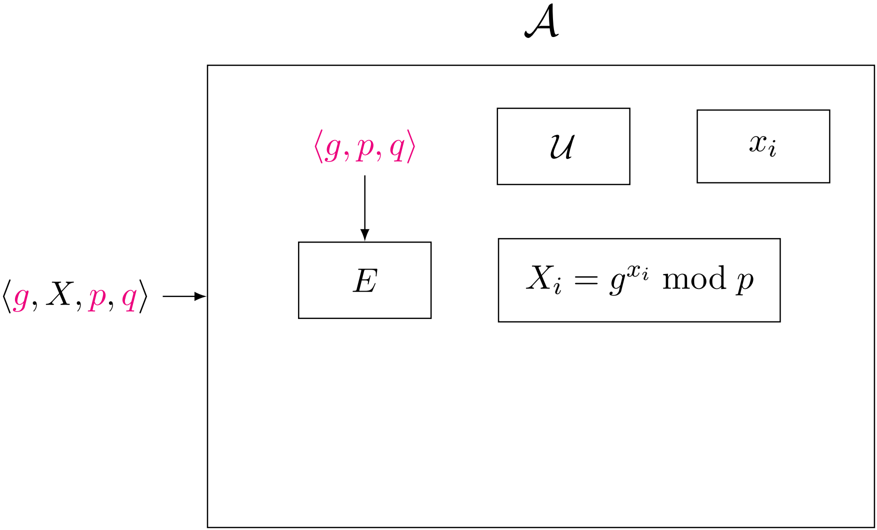
Proof - Simulation



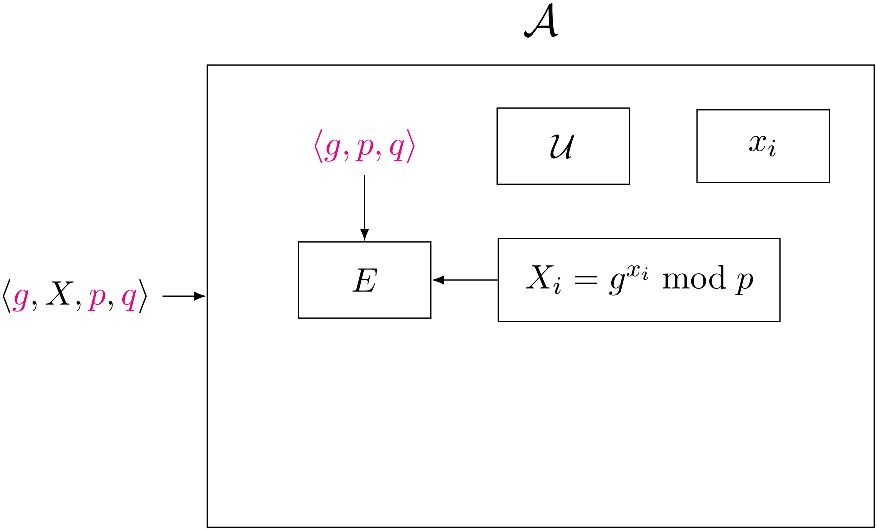
Proof - Simulation



Proof - Simulation



Proof - Simulation



Proof - Simulation

- H_1, H_2-**Queries**

Proof - Simulation

- H_1, H_2-**Queries**
- **KGen-Queries**

Proof - Simulation

- H_1, H_2-**Queries**
- **KGen-Queries**
- **KReveal-Queries**

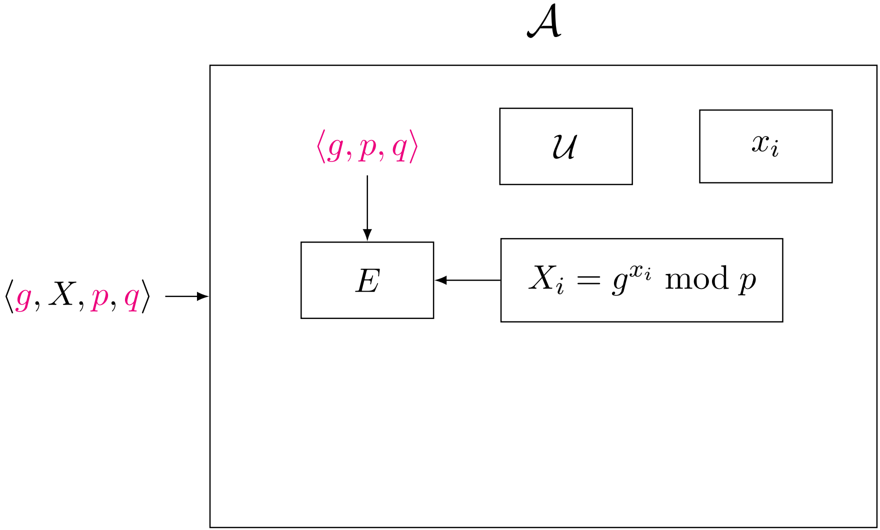
Proof - Simulation

- H_1, H_2-**Queries**
- **KGen-Queries**
- **KReveal-Queries**
- **ASign-Queries**

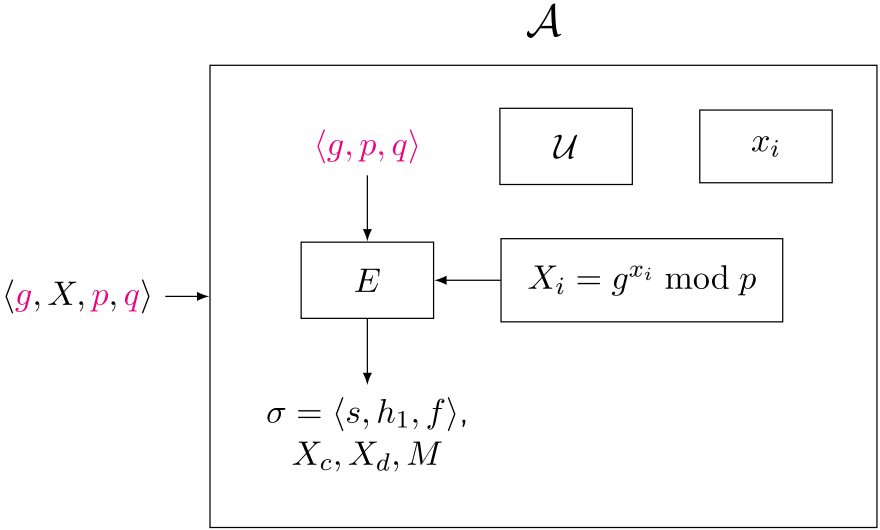
Proof - Simulation

- H_1, H_2-**Queries**
- **KGen-Queries**
- **KReveal-Queries**
- **ASign-Queries**
- **Private Key Extraction-Queries**

Proof - Simulation



Proof - Simulation



Proof - Simulation

With valid signature, one of the following holds

Proof - Simulation

With valid signature, one of the following holds

- No **ASIGN** query with $\langle X_c, X_d, f, M \rangle$, and no **Private Key Extraction** query for X_c or X_d

Proof - Simulation

With valid signature, one of the following holds

- No **ASIGN** query with $\langle X_c, X_d, f, M \rangle$, and no **Private Key Extraction** query for X_c or X_d
- No **ASIGN** query with $\langle X_c, X_i, f, M \rangle$, no **Private Key Extraction** query for X_c

Proof - Simulation

with second case we have equation

$$h = h_1 + f = H_2(g^s X_c^{h_1} X_d^f \bmod p \parallel M)$$

Proof - Simulation

with second case we have equation

$$h = h_1 + f = H_2(g^s X_c^{h_1} X_d^f \bmod p \parallel M)$$

Case 1: $h = h_1 + f$ never appeared in previous signature query

Proof - Simulation

with second case we have equation

$$h = h_1 + f = H_2(g^s X_c^{h_1} X_d^f \bmod p \parallel M)$$

Case 1: $h = h_1 + f$ never appeared in previous signature query

- force E to rerun simulation to produce (r_1, h', r_2') with $h \neq h'$

Proof - Simulation

with second case we have equation

$$h = h_1 + f = H_2(g^s X_c^{h_1} X_d^f \bmod p \parallel M)$$

Case 1: $h = h_1 + f$ never appeared in previous signature query

- force E to rerun simulation to produce (r_1, h', r_2') with $h \neq h'$
- we have $h = h_1 + f \neq h_1' + f' = h'$

Proof - Simulation

with second case we have equation

$$h = h_1 + f = H_2(g^s X_c^{h_1} X_d^f \bmod p \parallel M)$$

Case 1: $h = h_1 + f$ never appeared in previous signature query

- force E to rerun simulation to produce (r_1, h', r_2') with $h \neq h'$
- we have $h = h_1 + f \neq h_1' + f' = h'$
    - $h_1 \stackrel{?}{=} h_1'$
    - $f \stackrel{?}{=} f'$
- due to different output from oracle queries:

Proof - Simulation

with second case we have equation

$$h = h_1 + f = H_2(g^s X_c^{h_1} X_d^f \bmod p \parallel M)$$

Case 1: $h = h_1 + f$ never appeared in previous signature query

- force E to rerun simulation to produce (r_1, h', r_2') with $h \neq h'$
- we have $h = h_1 + f \neq h_1' + f' = h'$
  - $h_1 \stackrel{?}{=} h_1'$
  - $f \stackrel{?}{=} f'$
- due to different output from oracle queries:

$$g^s X^{h_1} X_d^f = g^{s'} X^{h_1'} X_d^f$$

Proof - Simulation

with second case we have equation

$$h = h_1 + f = H_2(g^s X_c^{h_1} X_d^f \bmod p \parallel M)$$

Case 1: $h = h_1 + f$ never appeared in previous signature query

- force E to rerun simulation to produce (r_1, h', r_2') with $h \neq h'$
- we have $h = h_1 + f \neq h_1' + f' = h'$
  - $h_1 \stackrel{?}{=} h_1'$
  - $f \stackrel{?}{=} f'$
- due to different output from oracle queries:

$$g^s X^{h_1} X_d^f = g^{s'} X^{h_1'} X_d^f$$

Proof - Simulation

$$g^s X^{h_1} {X_d}^f = g^{s'} X^{h'_1} {X_d}^f$$

Proof - Simulation

$$g^{s}X^{h_1}{X_d}^{f} = g^{s'}X^{h_1'}{X_d}^{f}$$

Proof - Simulation

$$g^{s} X^{h_1} {X_d}^{f} = g^{s'} X^{h_1'} {X_d}^{f}$$
$$s + x h_1 + f = s' + x h_1' + f$$

Proof - Simulation

$$g^s X^{h_1} {X_d}^f = g^{s'} X^{h'_1} {X_d}^f$$
$$s + xh_1 + f = s' + xh'_1 + f$$
$$s + xh_1 = s' + xh'_1$$

Proof - Simulation

$$
\begin{aligned}
g^s X^{h_1} {X_d}^f &= g^{s'} X^{h'_1} {X_d}^f \\
s + xh_1 + f &= s' + xh'_1 + f \\
s + xh_1 &= s' + xh'_1 \\
x &= \frac{s - s'}{h'_1 - h_1}
\end{aligned}
$$

Proof - Simulation

$$
\begin{aligned}
g^s X^{h_1} {X_d}^f &= g^{s'} X^{h'_1} {X_d}^f \\
s + xh_1 + f &= s' + xh'_1 + f \\
s + xh_1 &= s' + xh'_1 \\
x &= \frac{s - s'}{h'_1 - h_1}
\end{aligned}
$$

- solved the discrete logarithm problem with non-negligible probability

Proof - Simulation

$$
\begin{aligned}
g^s X^{h_1} {X_d}^f &= g^{s'} X^{h'_1} {X_d}^f \\
s + xh_1 + f &= s' + xh'_1 + f \\
s + xh_1 &= s' + xh'_1 \\
x &= \frac{s - s'}{h'_1 - h_1}
\end{aligned}
$$

- solved the discrete logarithm problem with non-negligible probability
- Henceforth, $\mathcal{A}$ solves the discrete logarithm problem in case 1

Proof - Simulation

Case 2: $h' = h$, output of signature query $\langle X_{c'}, X_{d'}, f', M' \rangle$

Proof - Simulation

Case 2: $h' = h$, output of signature query $\langle X_{c'}, X_{d'}, f', M' \rangle$

- then we have: $\sigma = \langle s', h', f' \rangle$

Proof - Simulation

Case 2: $h' = h$, output of signature query $\langle X_{c'}, X_{d'}, f', M' \rangle$

- then we have: $\sigma = \langle s', h', f' \rangle$
- due to $h = h'$ we have

$$h = H_2(g^s X^{h_1} X_d^f \parallel M) = H_2(g^{s'} X_{c'}^{h'_1} X_{d'}^{f'} \parallel M') = h'$$

Proof - Simulation

Case 2: $h' = h$, output of signature query $\langle X_{c'}, X_{d'}, f', M' \rangle$

- then we have: $\sigma = \langle s', h', f' \rangle$
- due to $h = h'$ we have

$$h = H_2(g^s X^{h_1} X_d^f \parallel M) = H_2(g^{s'} X_{c'}^{h_1'} X_{d'}^{f'} \parallel M') = h'$$

Proof - Simulation

Case 2: $h' = h$, output of signature query $\langle X_{c'}, X_{d'}, f', M' \rangle$

- then we have: $\sigma = \langle s', h', f' \rangle$
- due to $h = h'$ we have

$$h = H_2(g^s X^{h_1} X_d^f \parallel M) = H_2(g^{s'} X_{c'}^{h_1'} X_{d'}^{f'} \parallel M') = h'$$

Proof - Simulation

Case 2: $h' = h$, output of signature query $\langle X_{c'}, X_{d'}, f', M' \rangle$

- then we have: $\sigma = \langle s', h', f' \rangle$
- due to $h = h'$ we have

$$h = H_2(g^s X^{h_1} X_d^f \parallel M) = H_2(g^{s'} X_{c'}^{h_1'} X_{d'}^{f'} \parallel M') = h'$$

- when $X_{c'}, X_{d'} \neq X$, or $X_{c'}, X_{d'} = X$ but exponents are different

$$g^s X^{h_1} X_d^f = X_{c'}^{h_1'} X_{d'}^{f'}$$

  is easily solveable for x

Proof - Simulation

- case where $X_{c'}, X_{d'} = X$ and exponents are the same is negligible

Proof - Simulation

- case where $X_{c'}, X_{d'} = X$ and exponents are the same is negligible
  - $X_{c'}$ because no **ASIGN** query can be done on $\langle X_c, X_i, f, M \rangle$

Proof - Simulation

- case where $X_{c'}, X_{d'} = X$ and exponents are the same is negligible
  - $X_{c'}$ because no **ASIGN** query can be done on $\langle X_c, X_i, f, M \rangle$
  - $X_{d'}$ because some H_1 has to match h_1'

Proof - Simulation

- case where $X_{c'}, X_{d'} = X$ and exponents are the same is negligible
  - $X_{c'}$ because no **ASIGN** query can be done on $\langle X_c, X_i, f, M \rangle$
  - $X_{d'}$ because some H_1 has to match h'_1
- $\mathcal{A}$ can solve the discrete logarithm in case 2

Proof - Simulation

- case where $X_{c'}, X_{d'} = X$ and exponents are the same is negligible
  - $X_{c'}$ because no **ASIGN** query can be done on $\langle X_c, X_i, f, M \rangle$
  - $X_{d'}$ because some H_1 has to match h_1'
- $\mathcal{A}$ can solve the discrete logarithm in case 2
- Hence, $\mathcal{A}$ can solve the discrete logarithm $\lightning$

Outline

Conclusion

- sign entities without a significant disadvantage
- ambiguous until all parties are committed
- still not *truly fair*