

tBLS: Threshold BLS Signature Scheme



Renas Bacho



Alexandra Boldyreva



Sourav Das



Julian Loss



souravdas1547@gmail.com

Outline

- Background on Bilinear pairing
- Boneh-Lynn Shacham Signature Scheme
- Threshold Secret Sharing
- Design of Threshold BLS Signature
- Security of Threshold BLS Signature
- Evaluation Results

Background on Bilinear Pairing

Background on Bilinear Pairing

- Finite field \mathbb{F} of prime order q

Background on Bilinear Pairing

- Finite field \mathbb{F} of prime order q
- Three groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ of order q each

Background on Bilinear Pairing

- Finite field \mathbb{F} of prime order q
- Three groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ of order q each
- Efficiently computable bilinear map.

$$e: \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$$

Background on Bilinear Pairing

- Finite field \mathbb{F} of prime order q
- Three groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ of order q each
- Efficiently computable bilinear map.

$$e: \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$$

1. Bilinearity:

Background on Bilinear Pairing

- Finite field \mathbb{F} of prime order q
- Three groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ of order q each
- Efficiently computable bilinear map.

$$e: \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$$

1. Bilinearity:

$$\forall a, b \in \mathbb{F}^*, (g_1, g_2) \in (\mathbb{G}_1, \mathbb{G}_2), \quad e(g_1^a, g_2^b) = e(g_1, g_2)^{a \cdot b}$$

Background on Bilinear Pairing

- Finite field \mathbb{F} of prime order q
- Three groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ of order q each
- Efficiently computable bilinear map.

$$e: \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$$

1. Bilinearity:

$$\forall a, b \in \mathbb{F}^*, (g_1, g_2) \in (\mathbb{G}_1, \mathbb{G}_2), \quad e(g_1^a, g_2^b) = e(g_1, g_2)^{a \cdot b}$$

2. Non-degenerate:

Background on Bilinear Pairing

- Finite field \mathbb{F} of prime order q
- Three groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ of order q each
- Efficiently computable bilinear map.

$$e: \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$$

1. Bilinearity:

$$\forall a, b \in \mathbb{F}^*, (g_1, g_2) \in (\mathbb{G}_1, \mathbb{G}_2), \quad e(g_1^a, g_2^b) = e(g_1, g_2)^{a \cdot b}$$

2. Non-degenerate:

Given generators $(g_1, g_2) \in (\mathbb{G}_1, \mathbb{G}_2)$, $e(g_1, g_2)$ generates \mathbb{G}_T

Boneh-Lynn-Sacham (BLS) Signatures

Boneh-Lynn-Sacham (BLS) Signatures

Bilinear pairing-based signature scheme

Boneh-Lynn-Sacham (BLS) Signatures

Bilinear pairing-based signature scheme

Key generation

$$\begin{aligned} \text{sk} &:= s \leftarrow \mathbb{F} \\ \text{pk} &:= g^s \in \mathbb{G}_1 \end{aligned}$$

Boneh-Lynn-Sacham (BLS) Signatures

Bilinear pairing-based signature scheme

Key generation

$$\begin{aligned} \text{sk} &:= s \leftarrow \mathbb{F} \\ \text{pk} &:= g^s \in \mathbb{G}_1 \end{aligned}$$

Signing

$$\sigma := H(m)^s \in \mathbb{G}_2$$

Boneh-Lynn-Sacham (BLS) Signatures

Bilinear pairing-based signature scheme

Key generation

$$\begin{aligned} \text{sk} &:= s \leftarrow \mathbb{F} \\ \text{pk} &:= g^s \in \mathbb{G}_1 \end{aligned}$$

Signing

$$\sigma := H(m)^s \in \mathbb{G}_2$$

Verification

$$e(\text{pk}, H(m)) = e(g, \sigma)$$

Boneh-Lynn-Sacham (BLS) Signatures

Bilinear pairing-based signature scheme

Key generation

$$\begin{aligned} \text{sk} &:= s \leftarrow \mathbb{F} \\ \text{pk} &:= g^s \in \mathbb{G}_1 \end{aligned}$$

Signing

$$\sigma := H(m)^s \in \mathbb{G}_2$$

Verification

$$e(\text{pk}, H(m)) = e(g, \sigma)$$

Correctness:

- LHS: $e(\text{pk}, H(m)) = e(g^s, H(m)) = e(g, H(m))^s$
- RHS: $e(g, \sigma) = e(g, H(m)^s) = e(g, H(m))^s$

Boneh-Lynn-Sacham (BLS) Signatures

Bilinear pairing-based signature scheme

Key generation

$$\begin{aligned} \text{sk} &:= s \leftarrow \mathbb{F} \\ \text{pk} &:= g^s \in \mathbb{G}_1 \end{aligned}$$

Signing

$$\sigma := H(m)^s \in \mathbb{G}_2$$

Verification

$$e(\text{pk}, H(m)) = e(g, \sigma)$$

Correctness:

- LHS: $e(\text{pk}, H(m)) = e(g^s, H(m)) = e(g, H(m))^s$
- RHS: $e(g, \sigma) = e(g, H(m)^s) = e(g, H(m))^s$

Security: Computational Diffie-Hellman (CDH) in the random oracle model

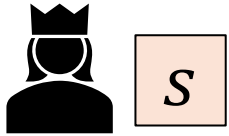
(n, t) Threshold Secret Sharing

(n, t) Threshold Secret Sharing

- Share a secret s into n shares

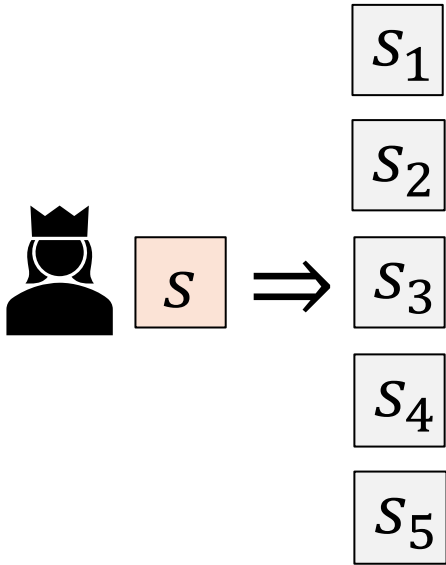
(n, t) Threshold Secret Sharing

- Share a secret s into n shares



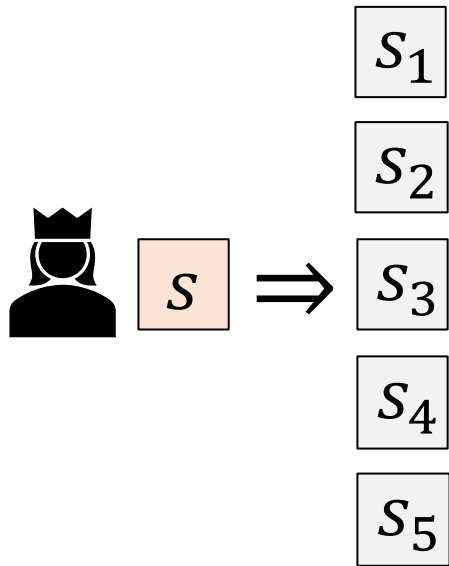
(n, t) Threshold Secret Sharing

- Share a secret s into n shares



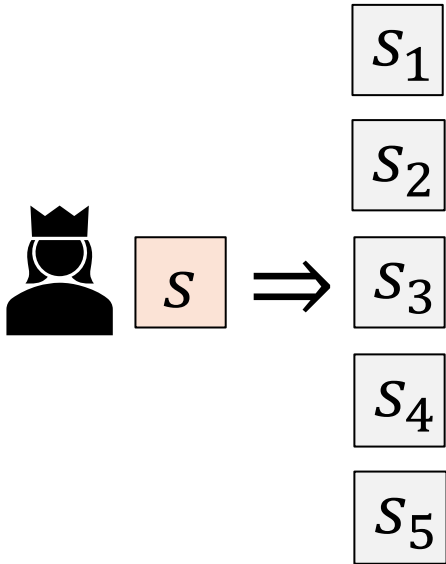
(n, t) Threshold Secret Sharing

- Share a secret s into n shares
- $\geq t + 1$ shares reveals s



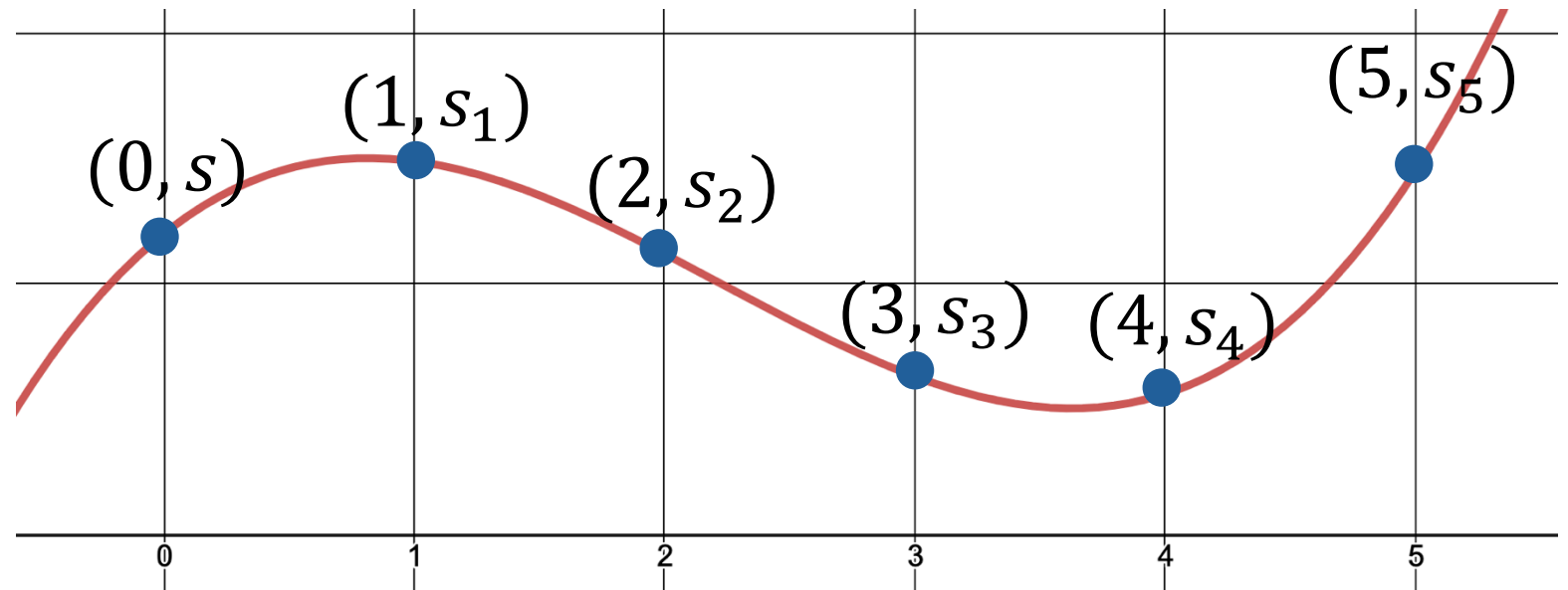
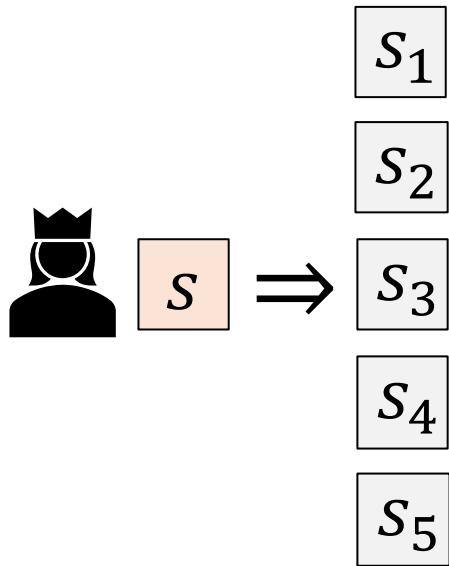
(n, t) Threshold Secret Sharing

- Share a secret s into n shares
- $\geq t + 1$ shares reveals s
- $\leq t$ shares hides s



(n, t) Threshold Secret Sharing

- Share a secret s into n shares
- $\geq t + 1$ shares **reveals** s
- $\leq t$ shares **hides** s
- Example: Shamir secret sharing



Lagrange Interpolation

Lagrange Interpolation

$$(a_1, P(a_1)), (a_2, P(a_2)), \dots, (a_t, P(a_t))$$

Lagrange Interpolation

$(a_1, P(a_1)), (a_2, P(a_2)), \dots, (a_t, P(a_t))$

\Rightarrow

$P(0)$

Lagrange Interpolation

$$(a_1, P(a_1)), (a_2, P(a_2)), \dots, (a_t, P(a_t)) \Rightarrow P(0)$$

Lagrange polynomials:

$$\lambda_{a_i}(X) = \prod_{i \neq j} \frac{(X - a_j)}{(a_i - a_j)}$$

Lagrange Interpolation

$$(a_1, P(a_1)), (a_2, P(a_2)), \dots, (a_t, P(a_t)) \Rightarrow P(0)$$

Lagrange polynomials:

$$\lambda_{a_i}(X) = \prod_{i \neq j} \frac{(X - a_j)}{(a_i - a_j)} \Rightarrow \lambda_{a_i}(a_i) = 1$$

Lagrange Interpolation

$$(a_1, P(a_1)), (a_2, P(a_2)), \dots, (a_t, P(a_t)) \Rightarrow P(0)$$

Lagrange polynomials:

$$\lambda_{a_i}(X) = \prod_{i \neq j} \frac{(X - a_j)}{(a_i - a_j)}$$

\Rightarrow

$$\lambda_{a_i}(a_i) = 1$$

$$\lambda_{a_i}(a_j) = 0$$

Lagrange Interpolation

$$(a_1, P(a_1)), (a_2, P(a_2)), \dots, (a_t, P(a_t)) \Rightarrow P(0)$$

Lagrange polynomials:

$$\lambda_{a_i}(X) = \prod_{i \neq j} \frac{(X - a_j)}{(a_i - a_j)} \Rightarrow \begin{aligned} \lambda_{a_i}(a_i) &= 1 \\ \lambda_{a_i}(a_j) &= 0 \end{aligned}$$

$$P(X) = \sum_{\forall a_i} P(a_i) \cdot \lambda_{a_i}(X)$$

Lagrange Interpolation

$$(a_1, P(a_1)), (a_2, P(a_2)), \dots, (a_t, P(a_t)) \Rightarrow P(0)$$

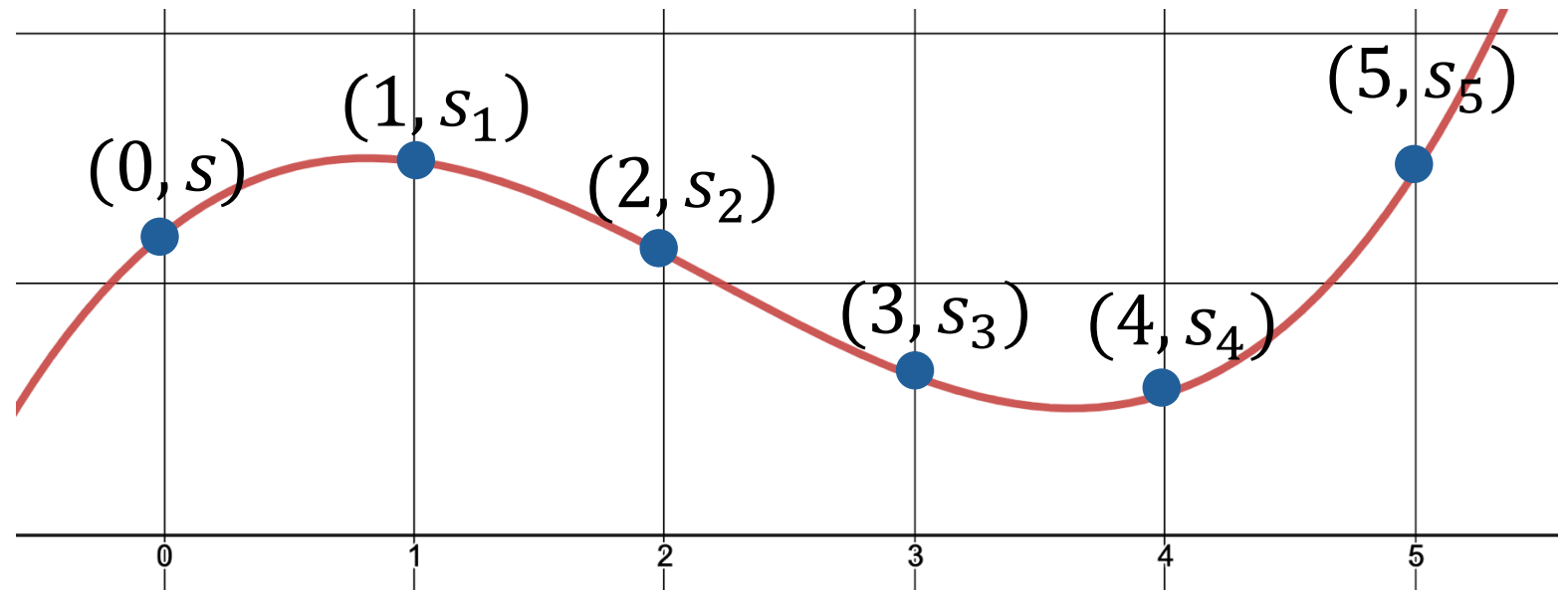
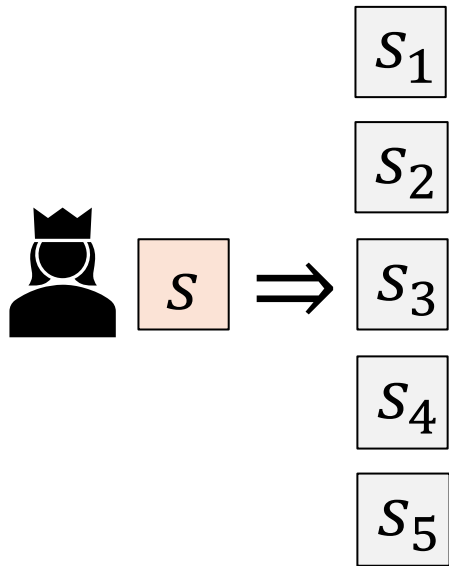
Lagrange polynomials:

$$\lambda_{a_i}(X) = \prod_{i \neq j} \frac{(X - a_j)}{(a_i - a_j)} \Rightarrow \begin{aligned} \lambda_{a_i}(a_i) &= 1 \\ \lambda_{a_i}(a_j) &= 0 \end{aligned}$$

$$P(X) = \sum_{\forall a_i} P(a_i) \cdot \lambda_{a_i}(X) \Rightarrow P(0) = \sum_{\forall a_i} P(a_i) \cdot \lambda_{a_i}(0)$$

(n, t) Threshold Secret Sharing

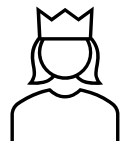
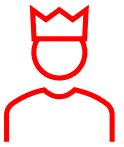
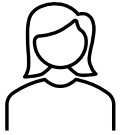
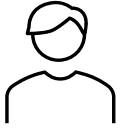
- Share a secret s into n shares
- $\geq t + 1$ shares **reveals** s
- $\leq t$ shares **hides** s
- Example: Shamir secret sharing



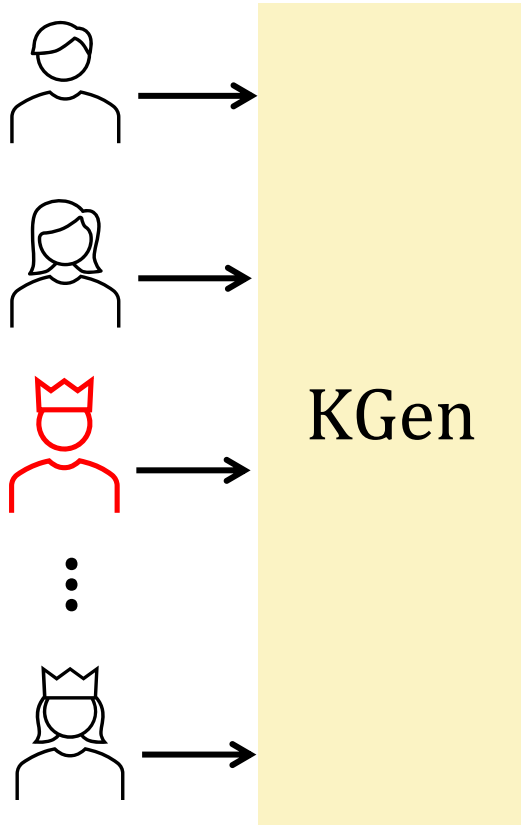
BLS Threshold signature [Boldyreva'03]

Key Generation

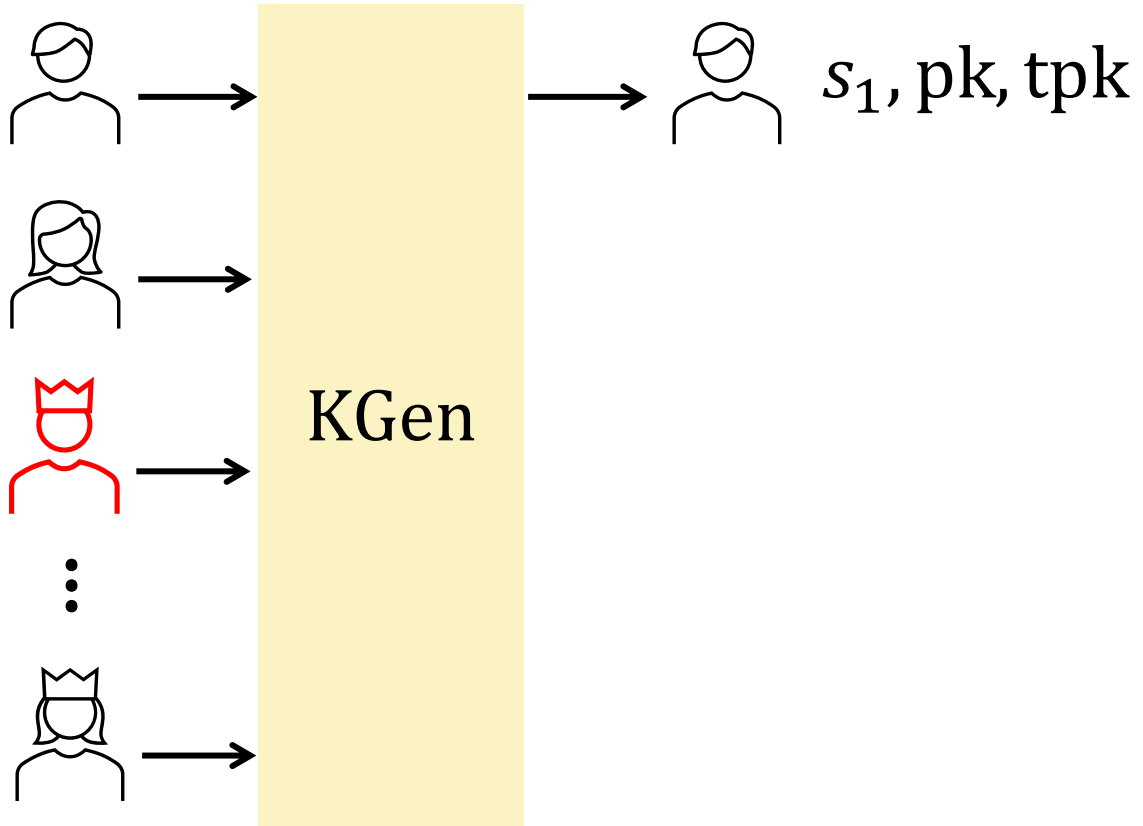
Key Generation



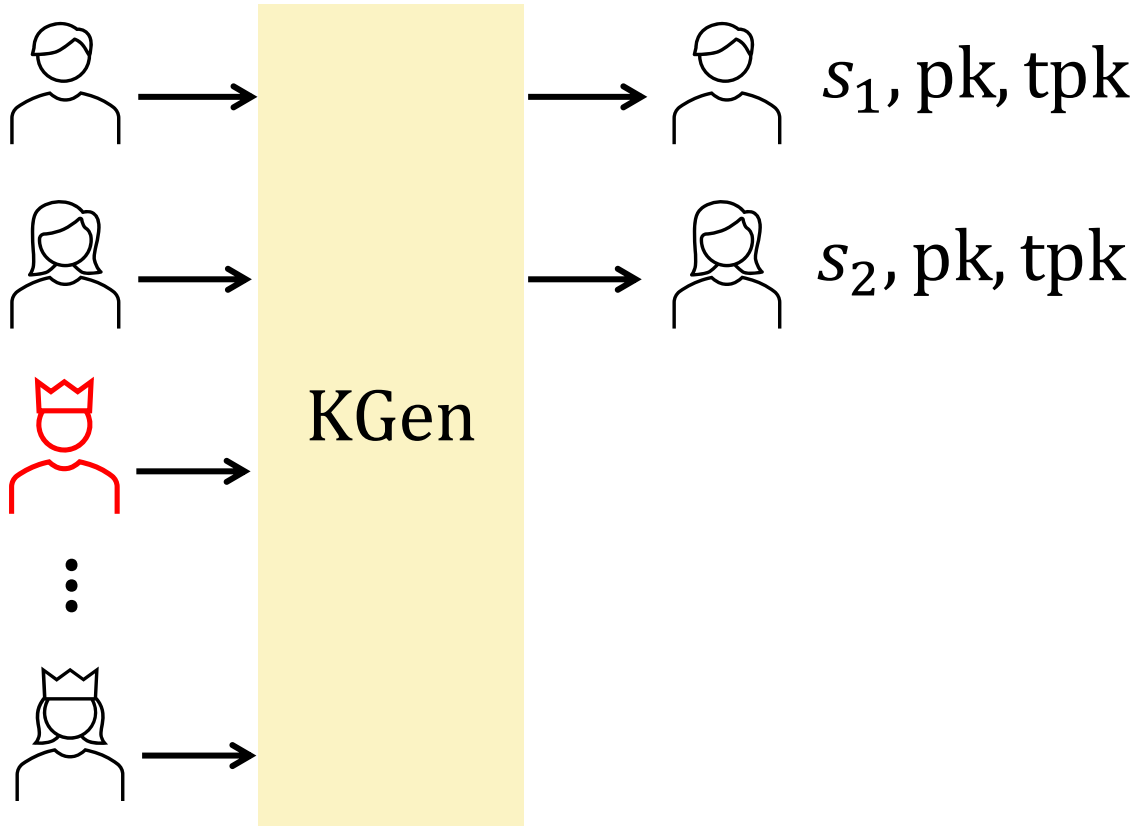
Key Generation



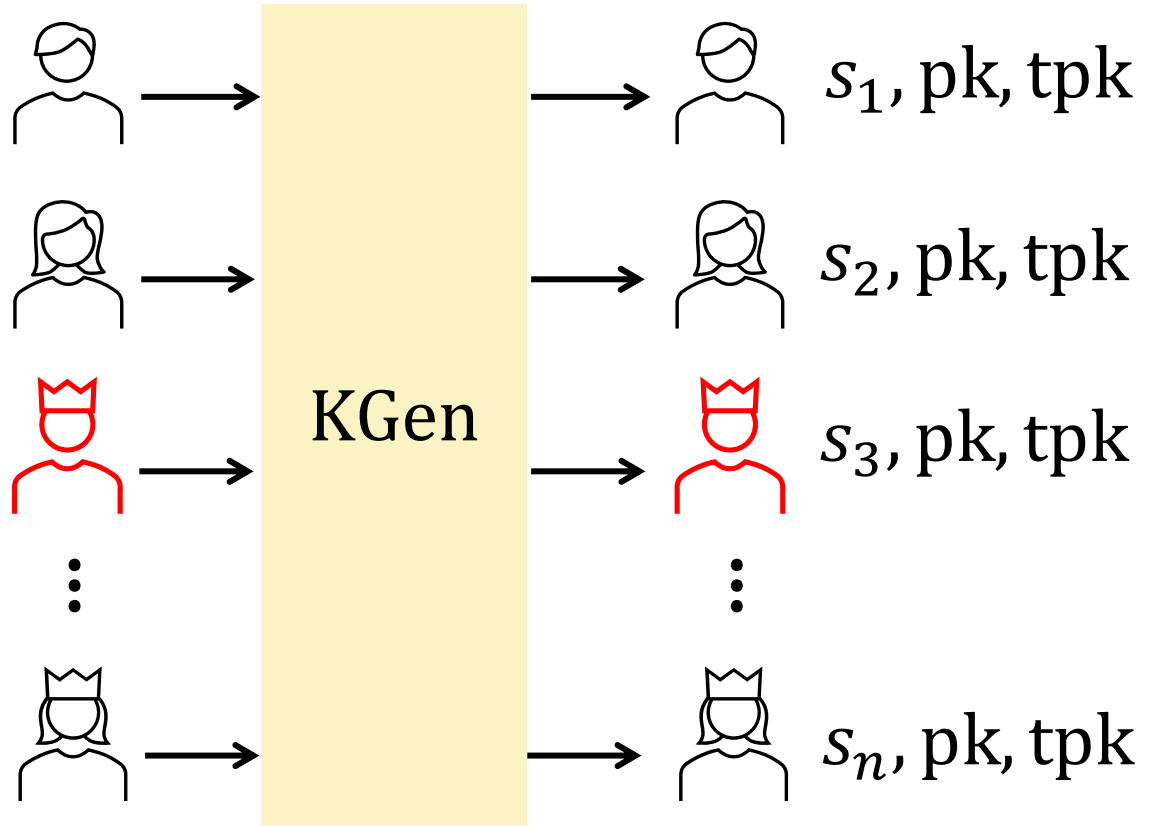
Key Generation



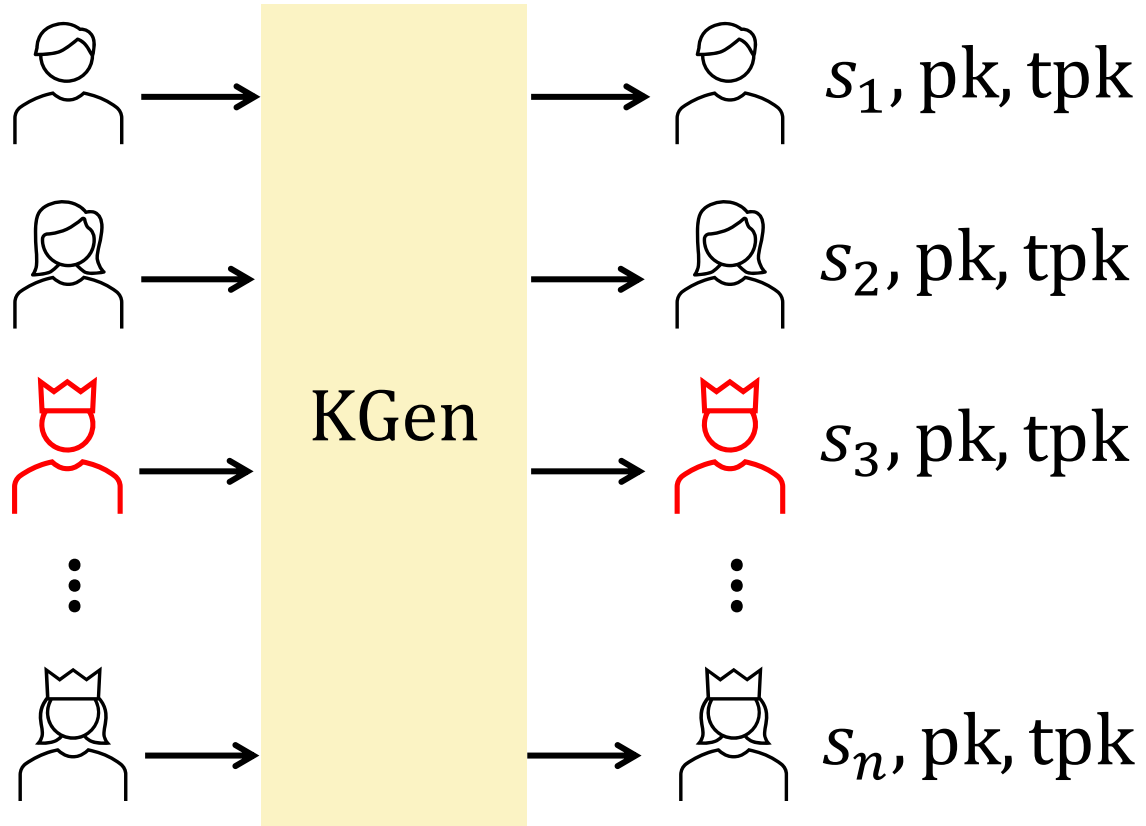
Key Generation



Key Generation



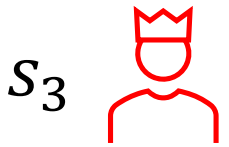
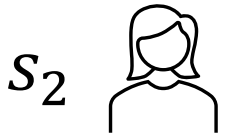
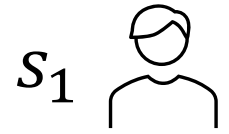
Key Generation



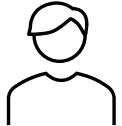
$$\begin{aligned} \{s_1, \dots, s_n\} &\leftarrow \text{Share}(s) \\ pk &= g^s \\ tpk &= \{g^{s_1}, \dots, g^{s_n}\} \end{aligned}$$


Signing

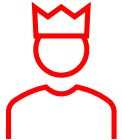
Signing



Signing

s_1  $\sigma_1 = H(m)^{s_1}$

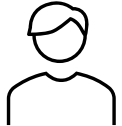
s_2 


s_3 

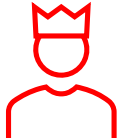
⋮

s_n 

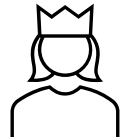
Signing

s_1  $\sigma_1 = H(m)^{s_1}$

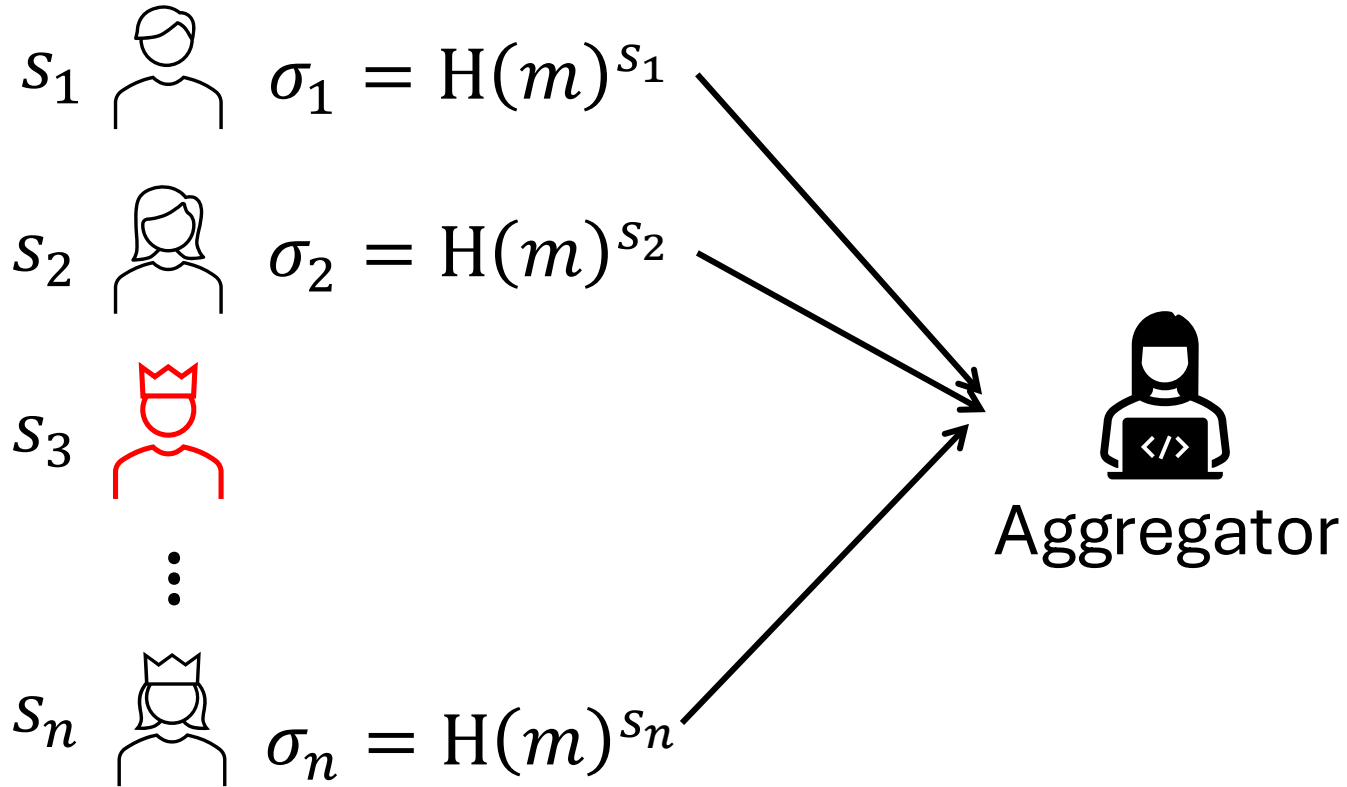
s_2  $\sigma_2 = H(m)^{s_2}$

s_3 

⋮

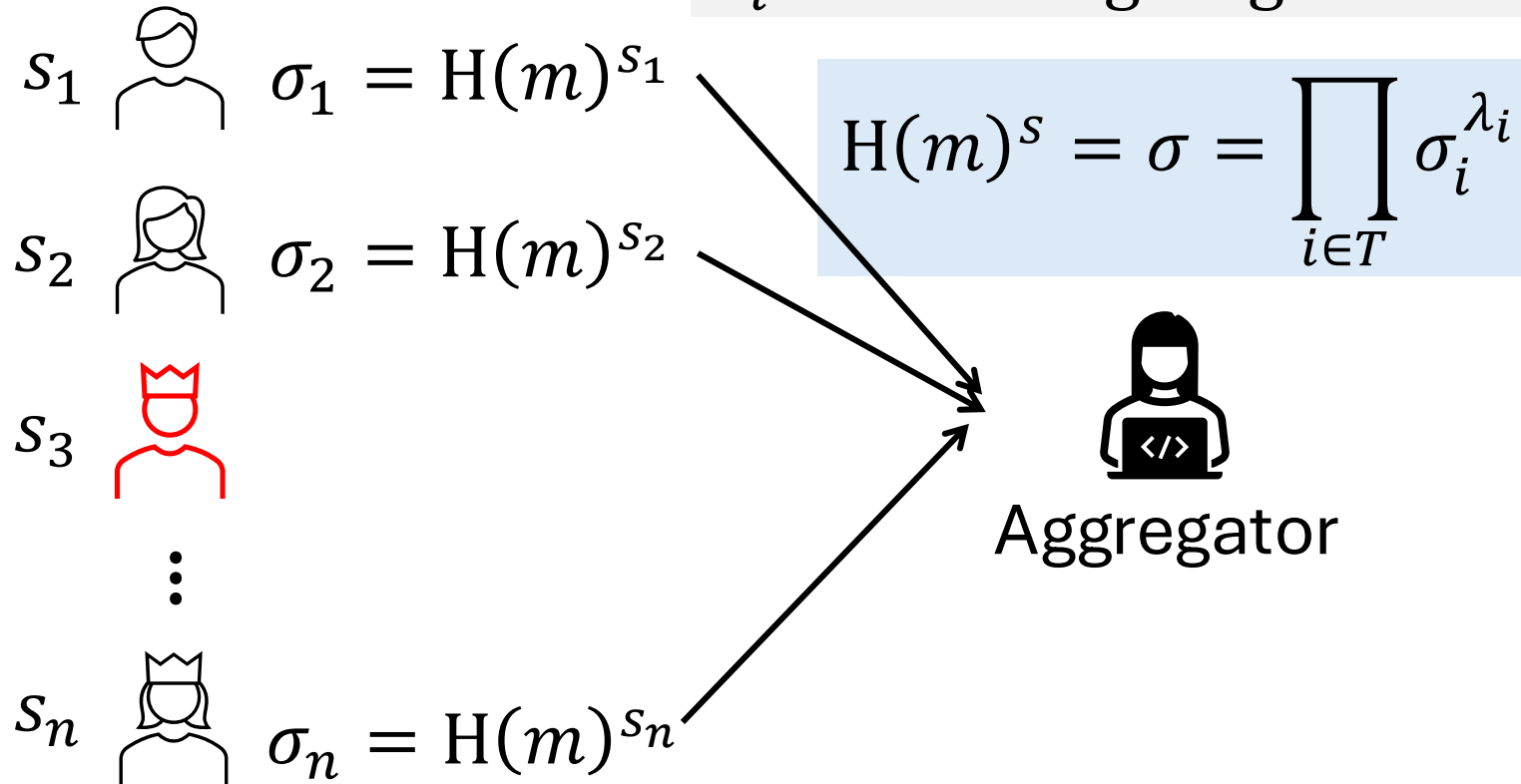
s_n  $\sigma_n = H(m)^{s_n}$

Signing



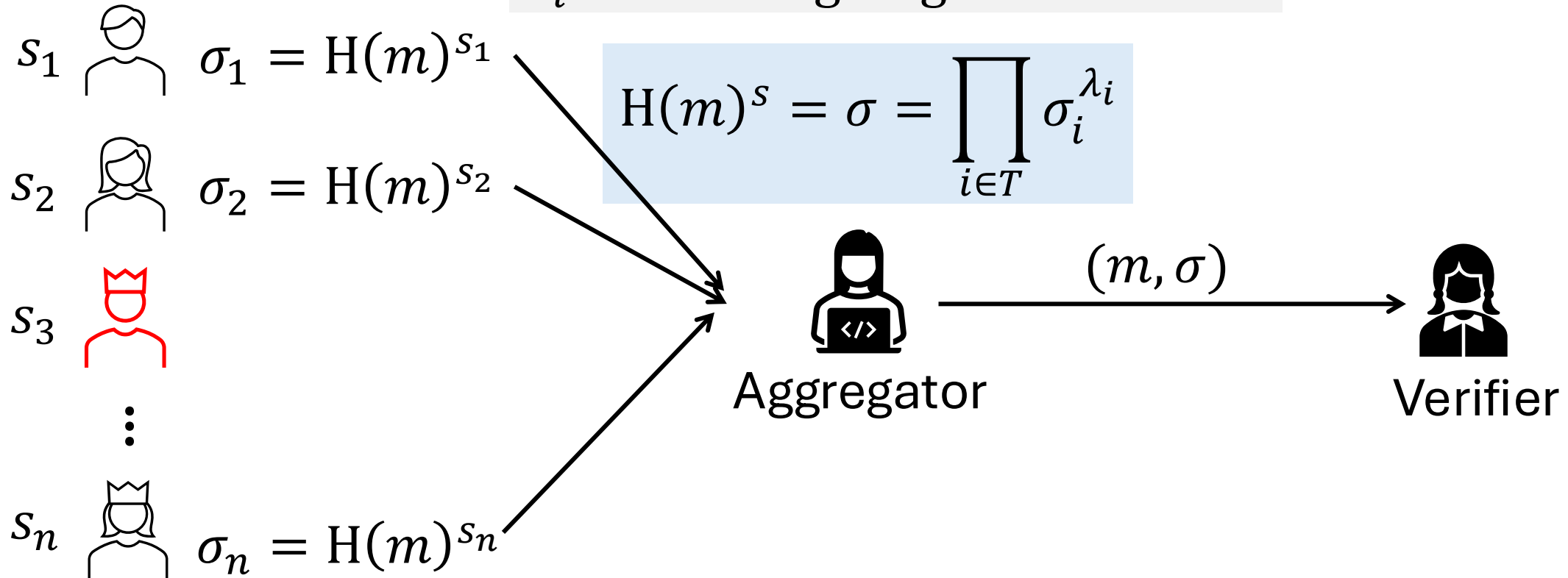
Signing

λ_i are the Lagrange coefficients.



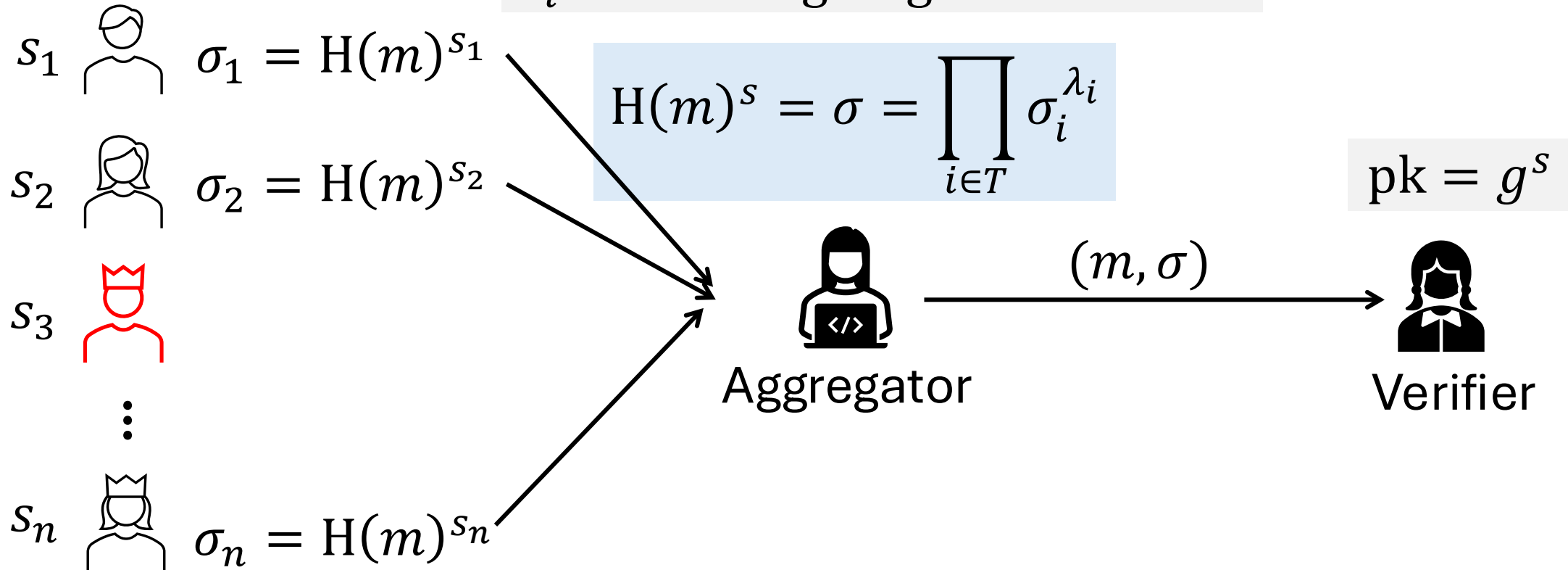
Signing

λ_i are the Lagrange coefficients.



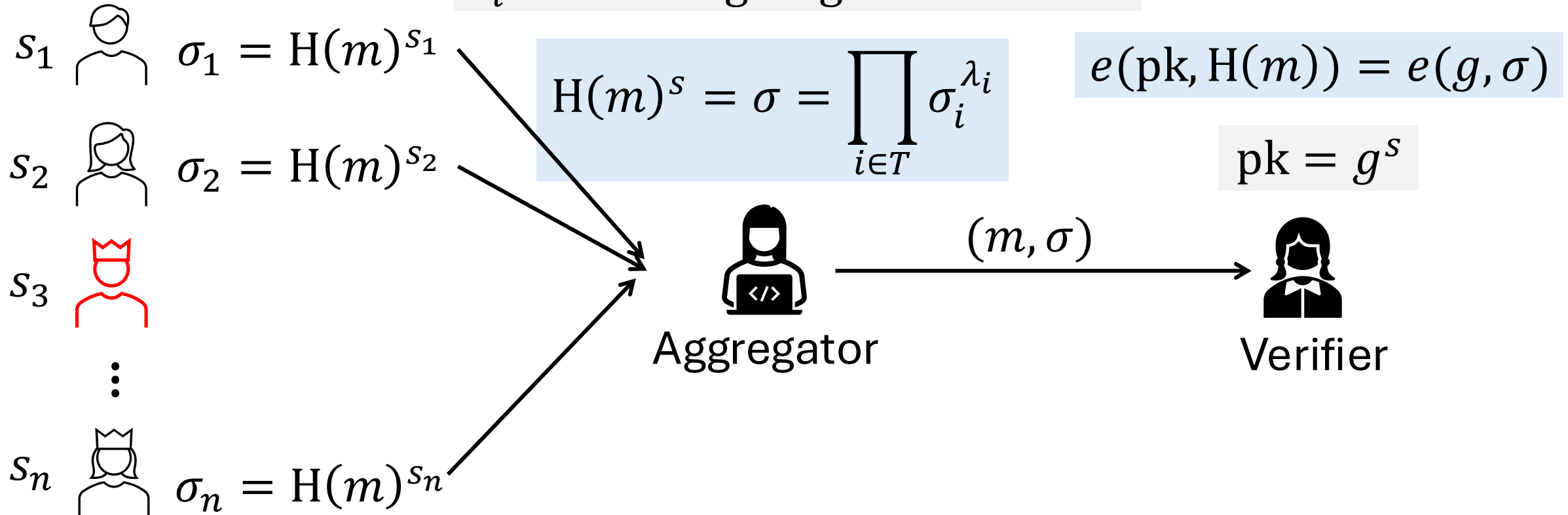
Signing

λ_i are the Lagrange coefficients.



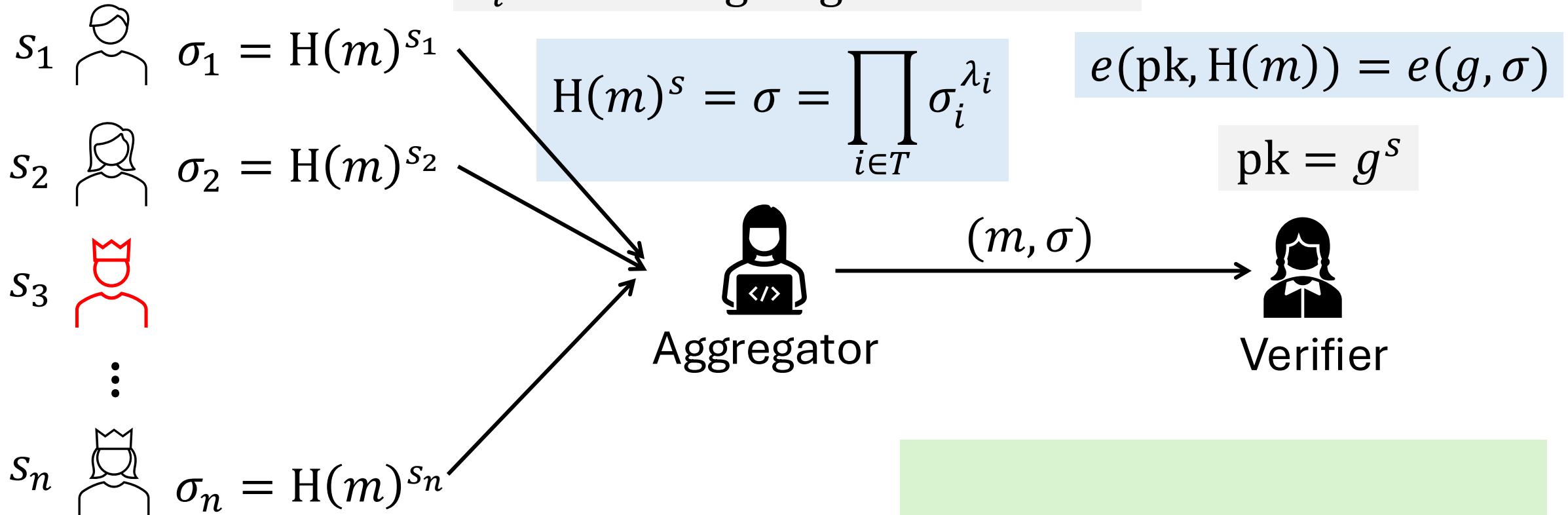
Signing

λ_i are the Lagrange coefficients.



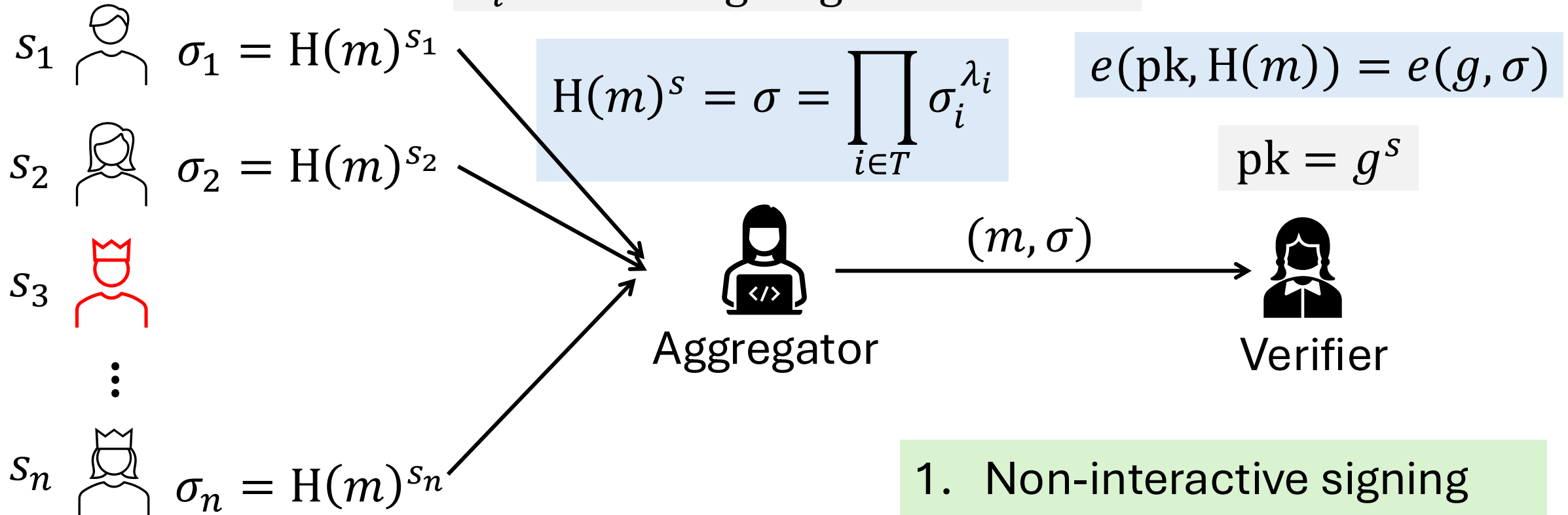
Signing

λ_i are the Lagrange coefficients.



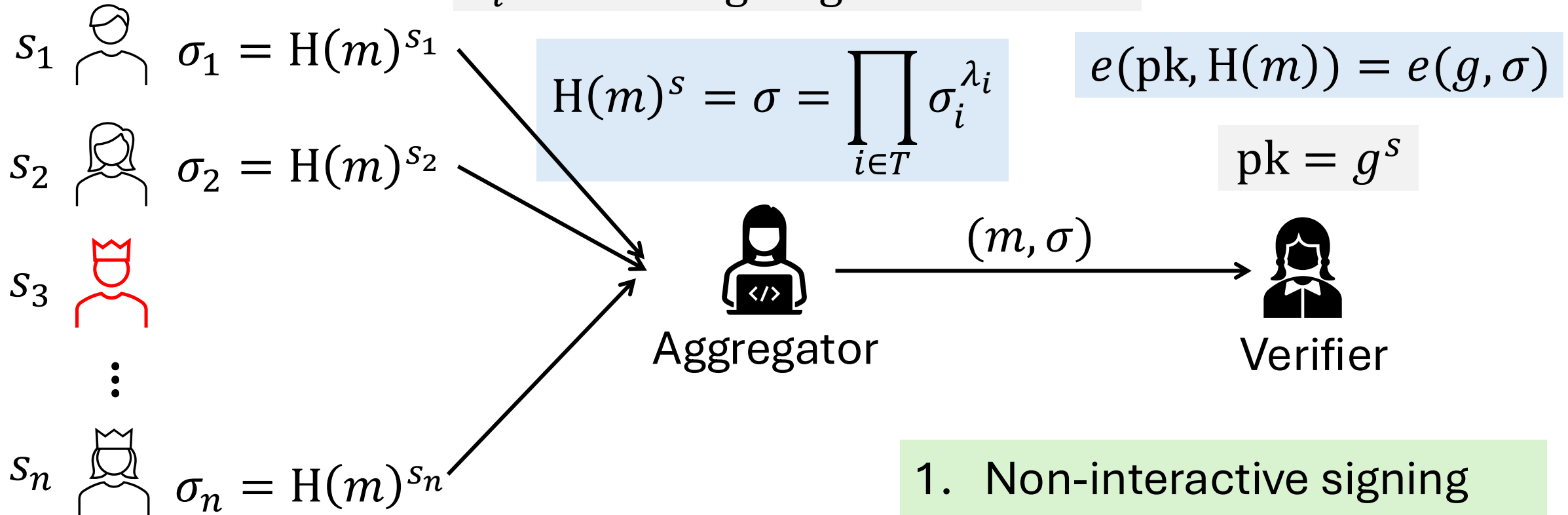
Signing

λ_i are the Lagrange coefficients.



Signing

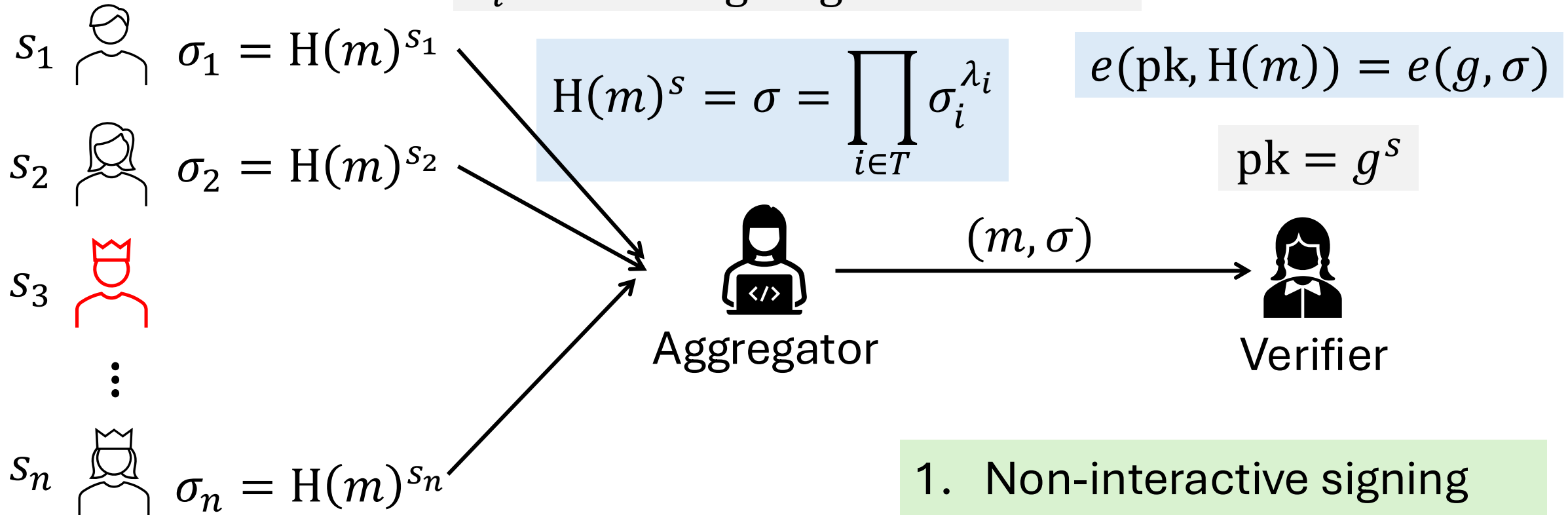
λ_i are the Lagrange coefficients.



- 1. Non-interactive signing
- 2. Constant signature size

Signing

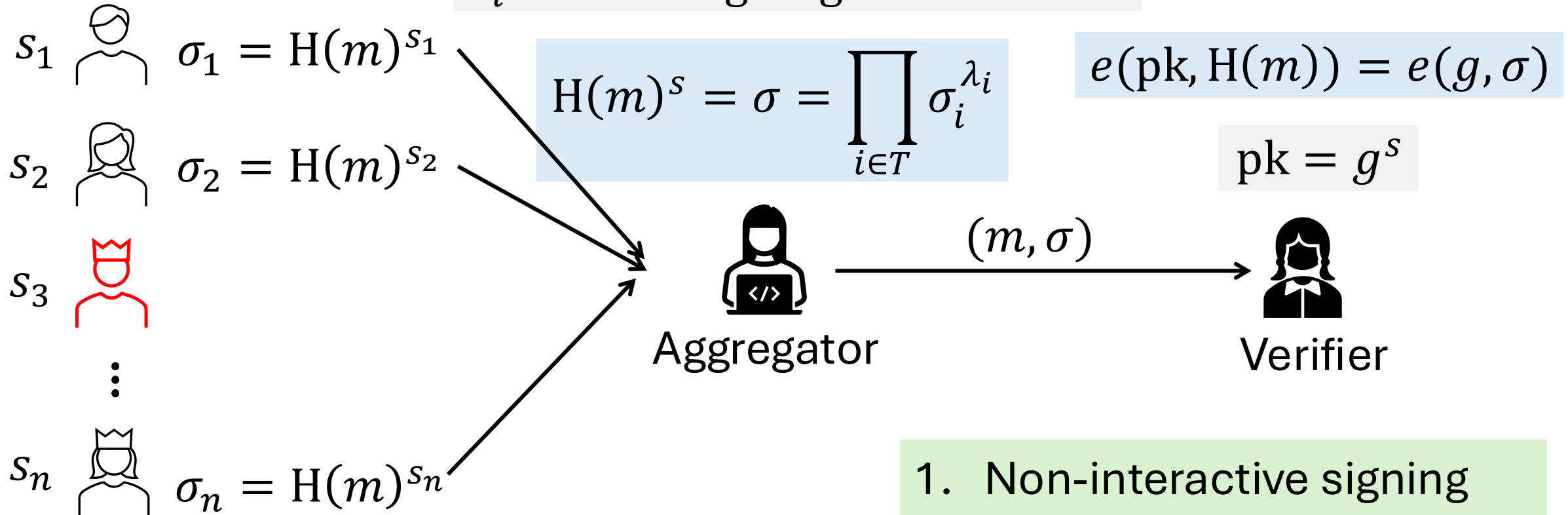
λ_i are the Lagrange coefficients.



- 1. Non-interactive signing
- 2. Constant signature size
- 3. Constant verification cost

Signing

λ_i are the Lagrange coefficients.

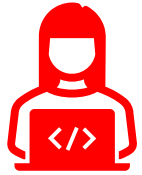


1. Non-interactive signing
2. Constant signature size
3. Constant verification cost
4. Unique signatures

Security of BLS Threshold Signature

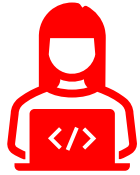
Threat model: Static vs Adaptive Corruption

Threat model: Static vs Adaptive Corruption



Static \mathcal{A}

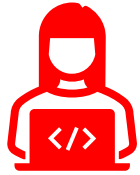
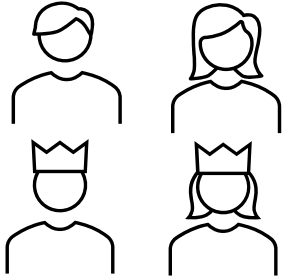
Threat model: Static vs Adaptive Corruption



Static \mathcal{A}

Decide corrupt parties
before the protocol

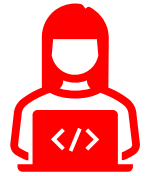
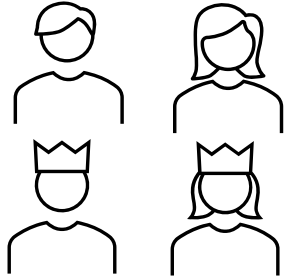
Threat model: Static vs Adaptive Corruption



Static \mathcal{A}

Decide corrupt parties
before the protocol

Threat model: Static vs Adaptive Corruption

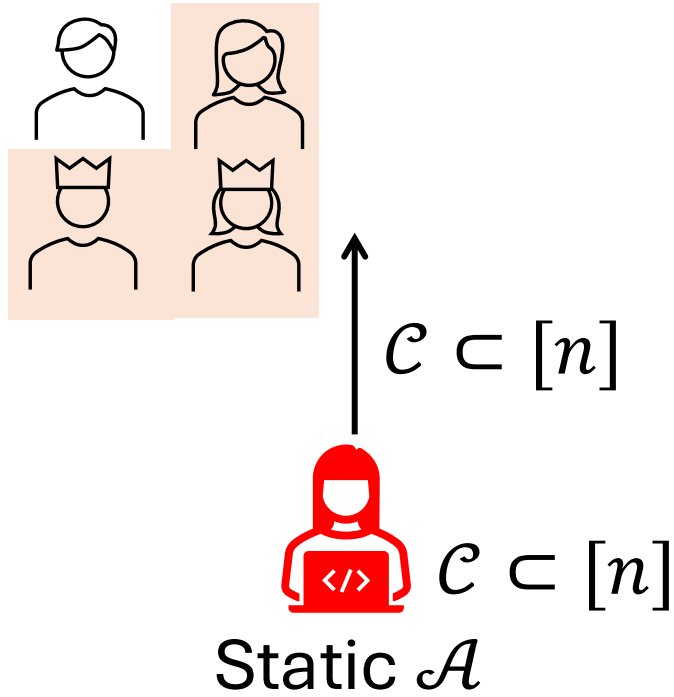


$\mathcal{C} \subset [n]$

Static \mathcal{A}

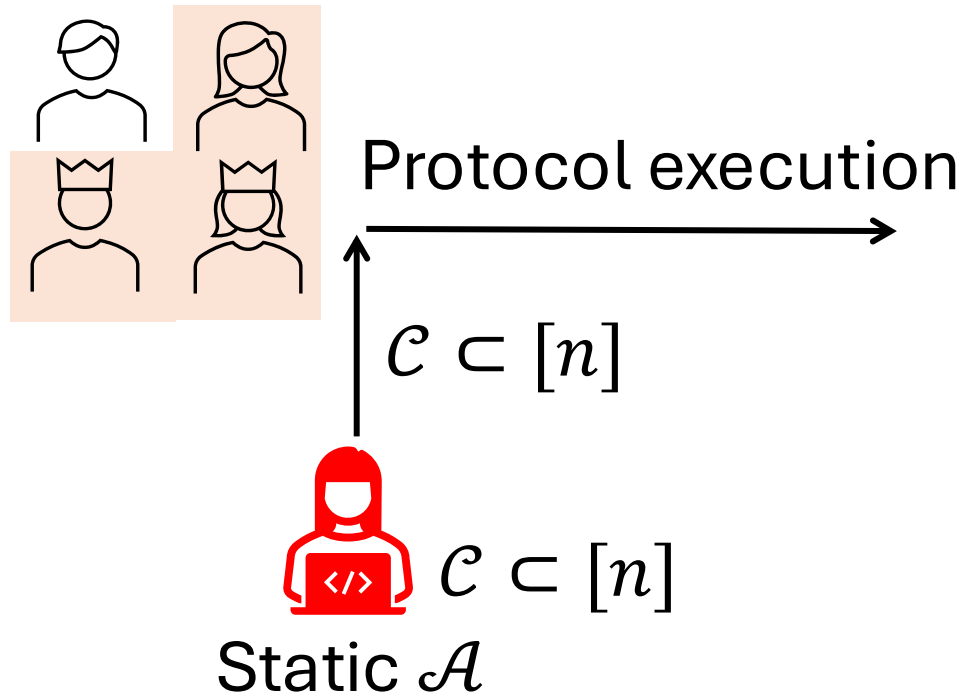
Decide corrupt parties
before the protocol

Threat model: Static vs Adaptive Corruption



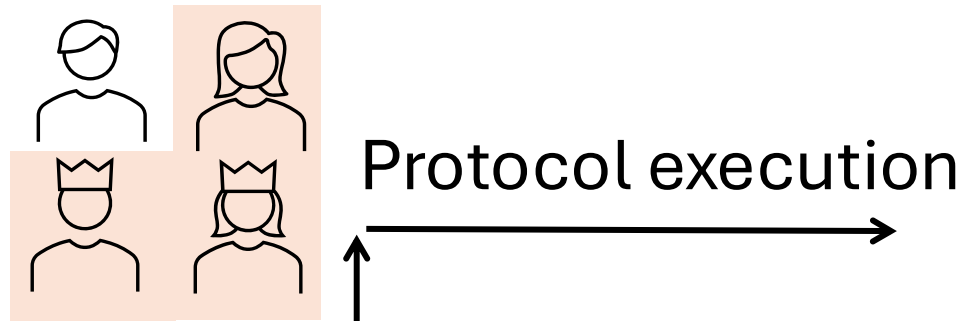
Decide corrupt parties
before the protocol

Threat model: Static vs Adaptive Corruption



Decide corrupt parties
before the protocol

Threat model: Static vs Adaptive Corruption



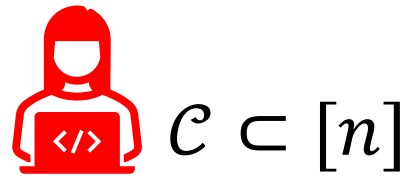
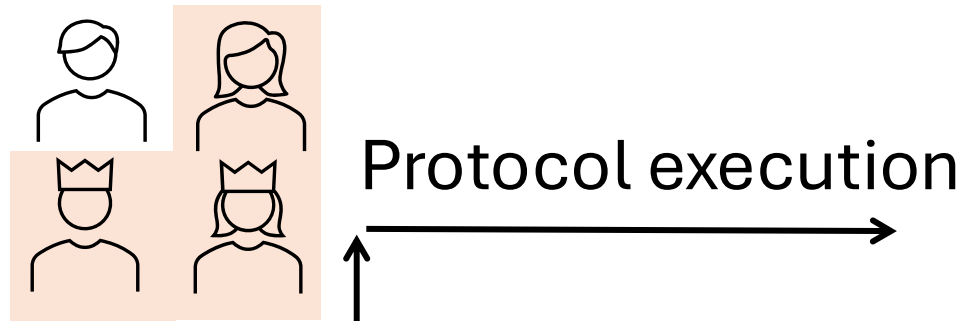
Static \mathcal{A}



Adaptive \mathcal{A}

Decide corrupt parties
before the protocol

Threat model: Static vs Adaptive Corruption



Static \mathcal{A}

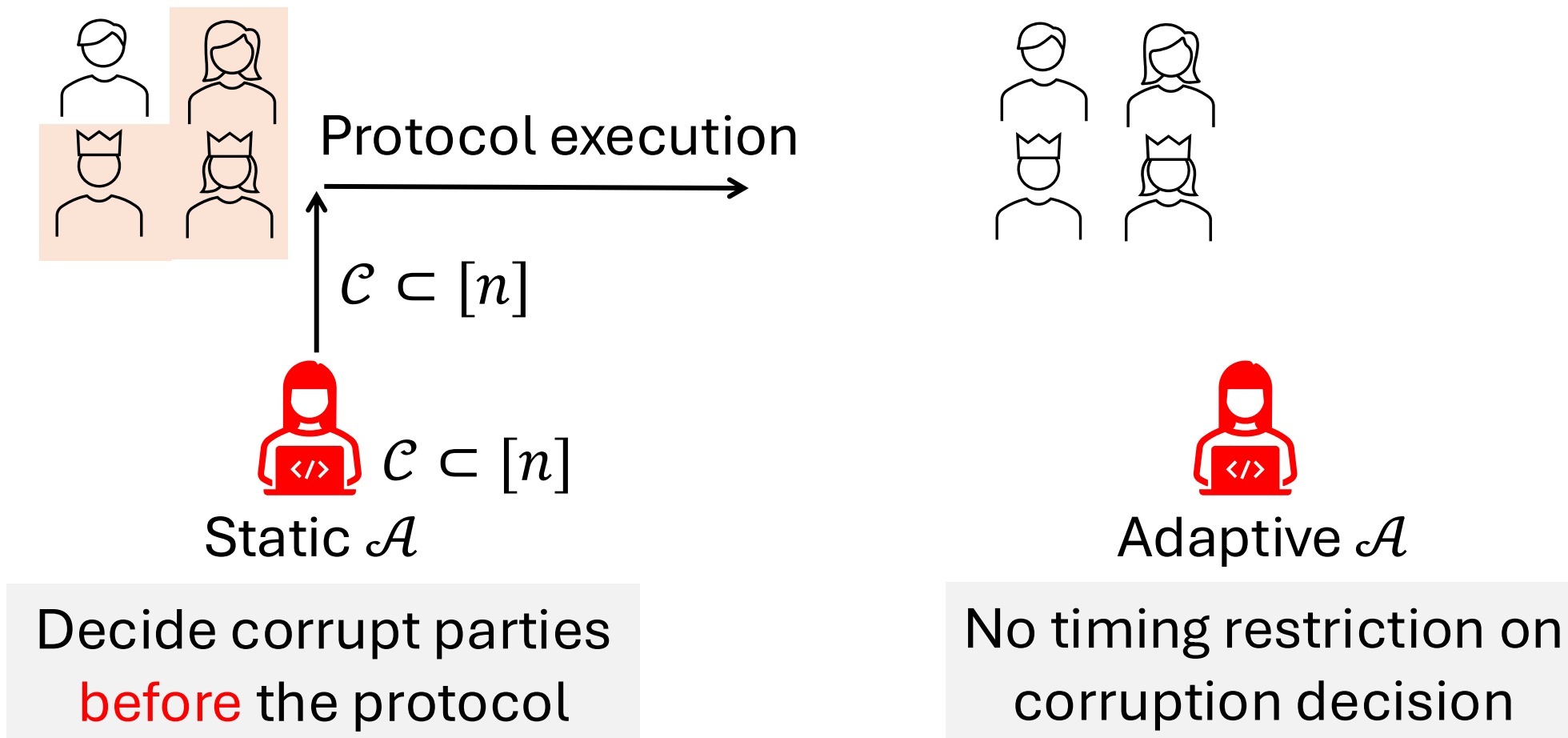
Decide corrupt parties
before the protocol



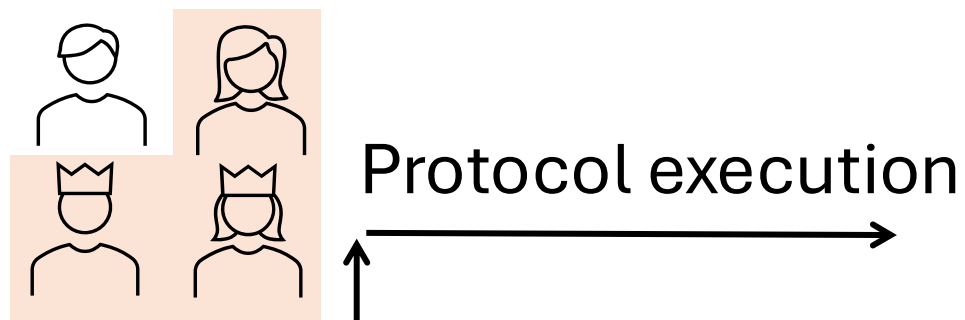
Adaptive \mathcal{A}

No timing restriction on
corruption decision

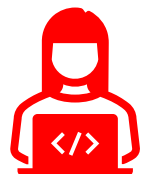
Threat model: Static vs Adaptive Corruption



Threat model: Static vs Adaptive Corruption



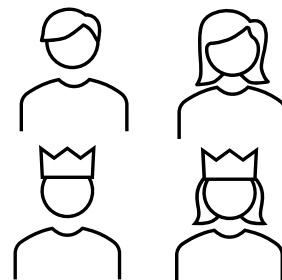
$$\mathcal{C} \subset [n]$$



$$\mathcal{C} \subset [n]$$

Static \mathcal{A}

Decide corrupt parties
before the protocol

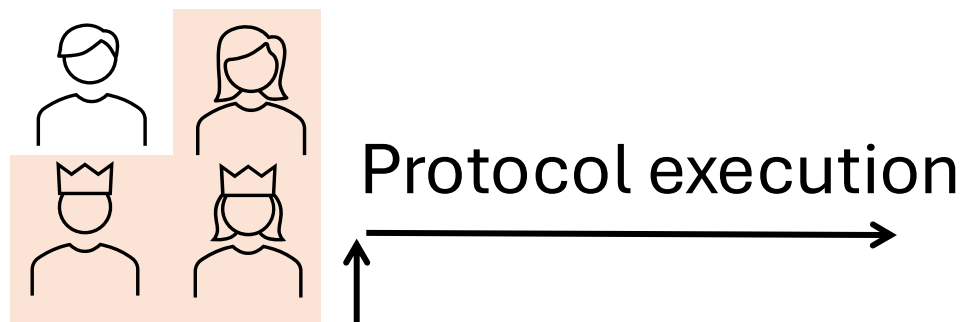


$$\mathcal{C} = \{3\}$$

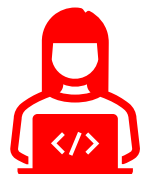
Adaptive \mathcal{A}

No timing restriction on
corruption decision

Threat model: Static vs Adaptive Corruption



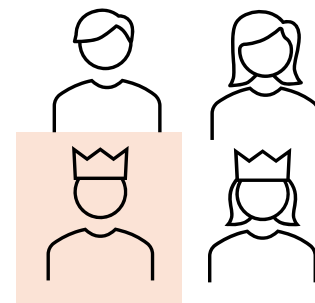
$\mathcal{C} \subset [n]$



$\mathcal{C} \subset [n]$

Static \mathcal{A}

Decide corrupt parties
before the protocol



3

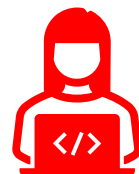
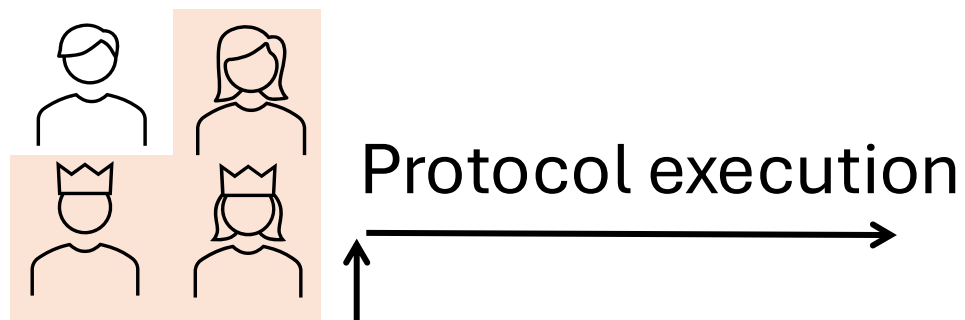


$\mathcal{C} = \{3\}$

Adaptive \mathcal{A}

No timing restriction on
corruption decision

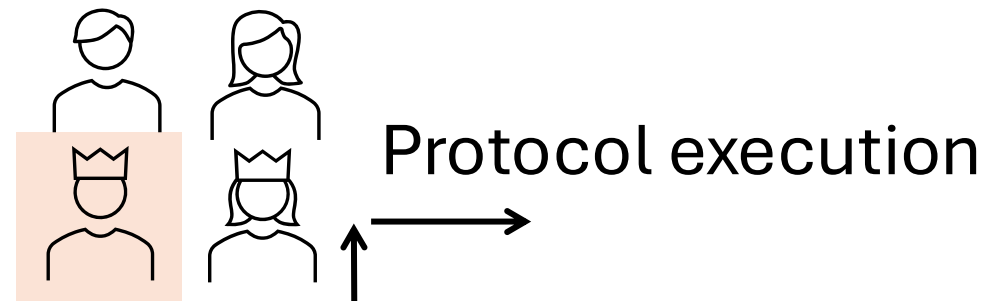
Threat model: Static vs Adaptive Corruption



$\mathcal{C} \subset [n]$

Static \mathcal{A}

Decide corrupt parties
before the protocol

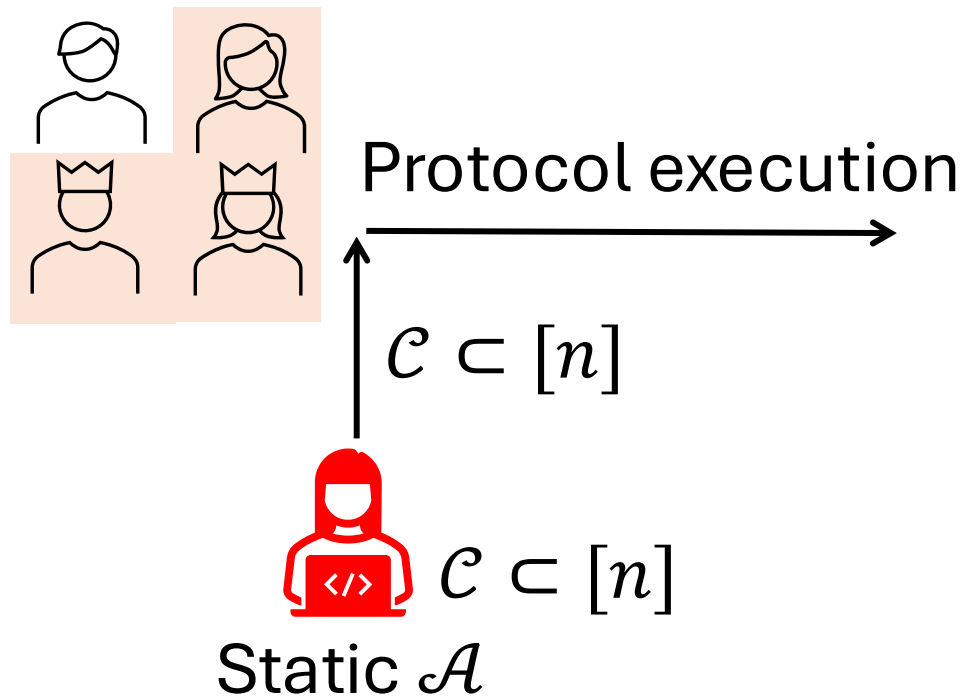


$\mathcal{C} = \{3\}$

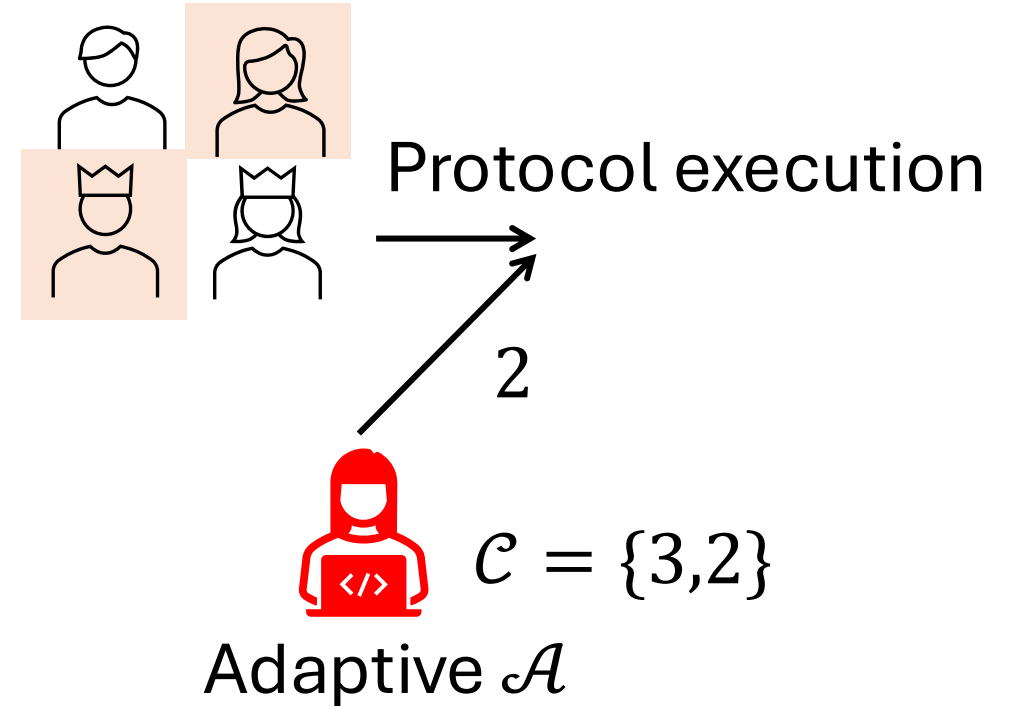
Adaptive \mathcal{A}

No timing restriction on
corruption decision

Threat model: Static vs Adaptive Corruption

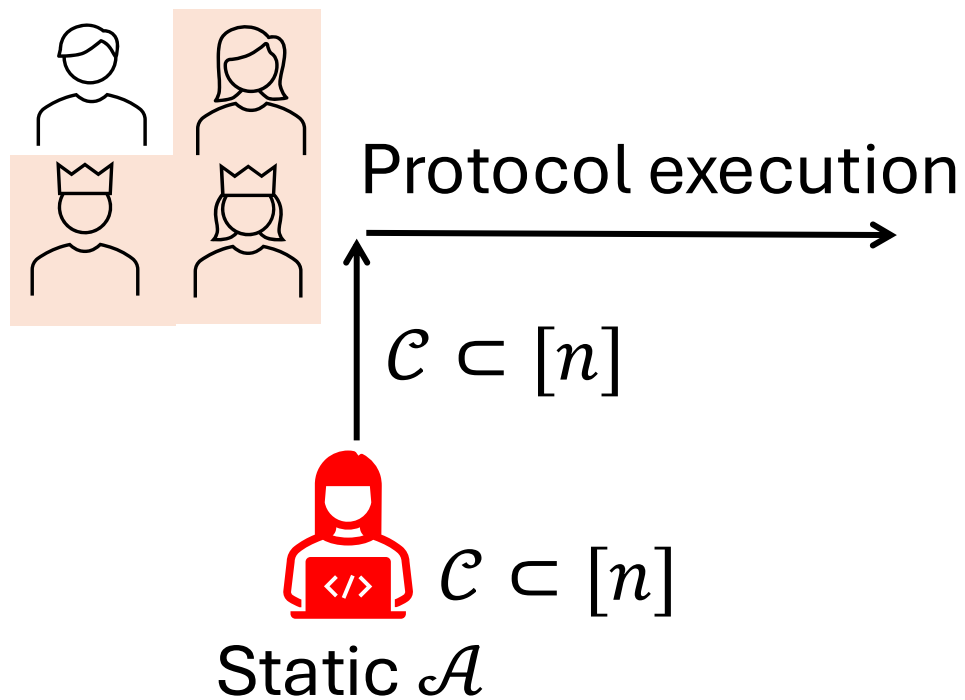


Decide corrupt parties
before the protocol

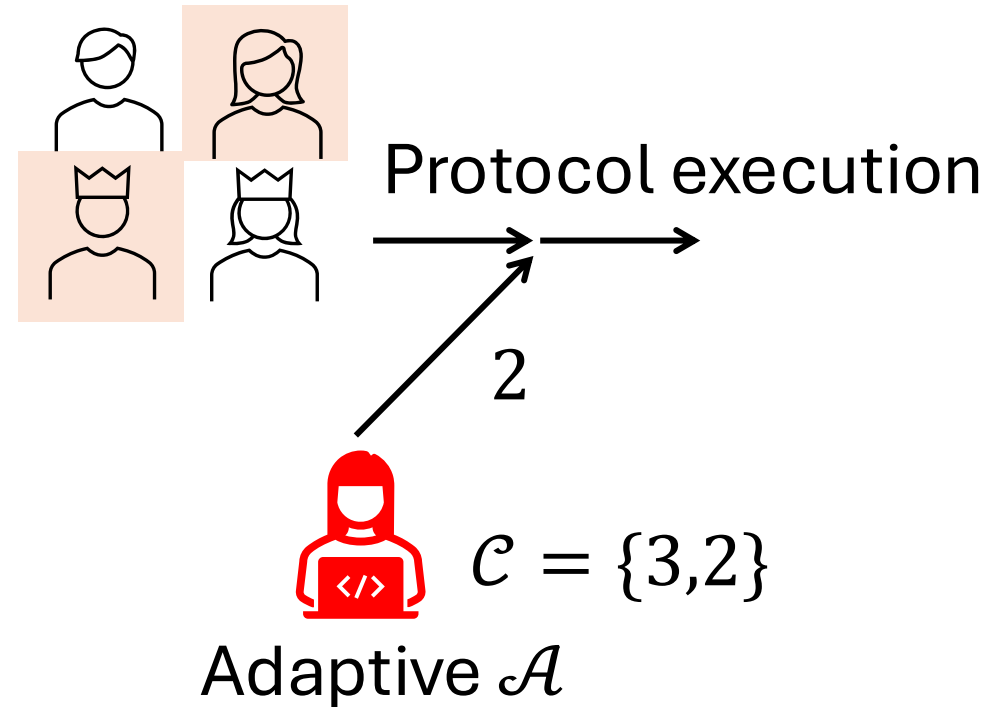


No timing restriction on
corruption decision

Threat model: Static vs Adaptive Corruption

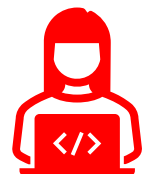
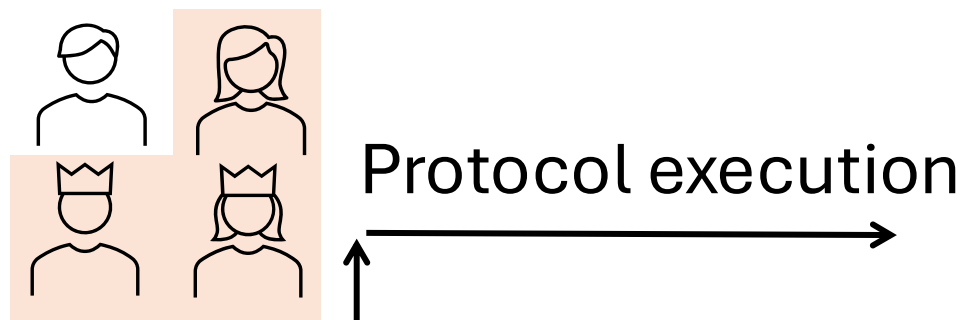


Decide corrupt parties
before the protocol



No timing restriction on
corruption decision

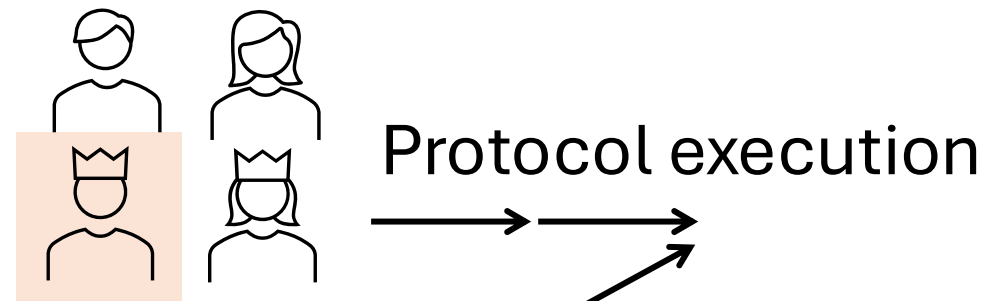
Threat model: Static vs Adaptive Corruption



$\mathcal{C} \subset [n]$

Static \mathcal{A}

Decide corrupt parties
before the protocol

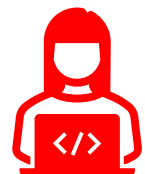
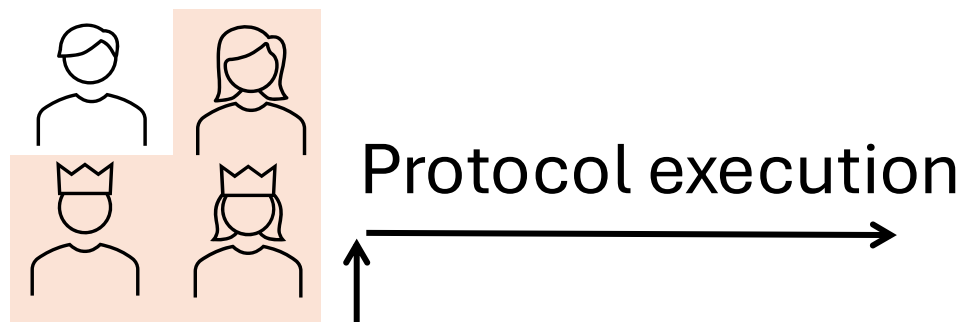


$\mathcal{C} = \{3,2,4\}$

Adaptive \mathcal{A}

No timing restriction on
corruption decision

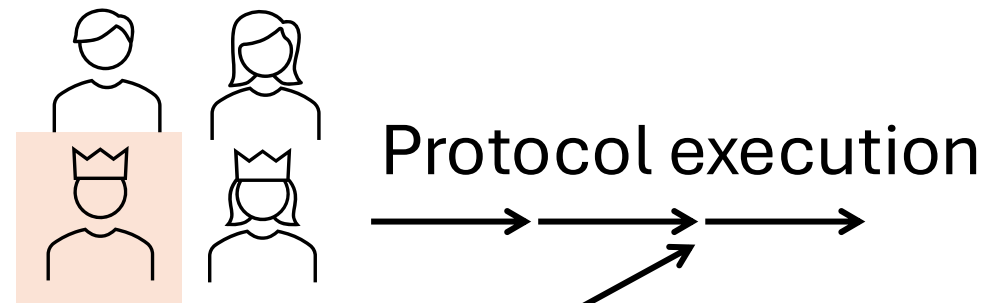
Threat model: Static vs Adaptive Corruption



$\mathcal{C} \subset [n]$

Static \mathcal{A}

Decide corrupt parties
before the protocol

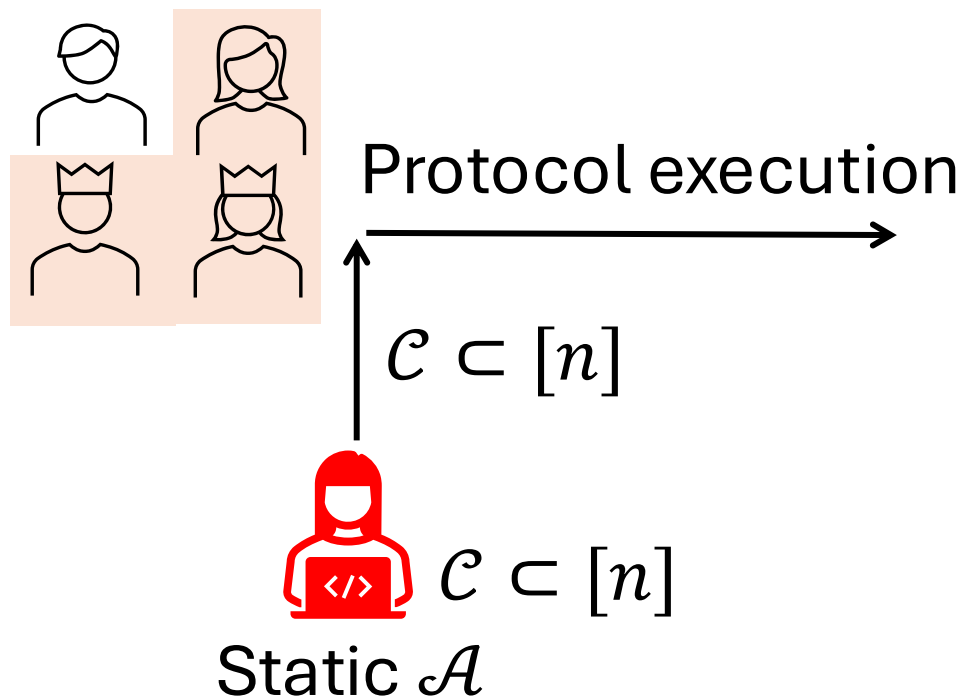


$\mathcal{C} = \{3,2,4\}$

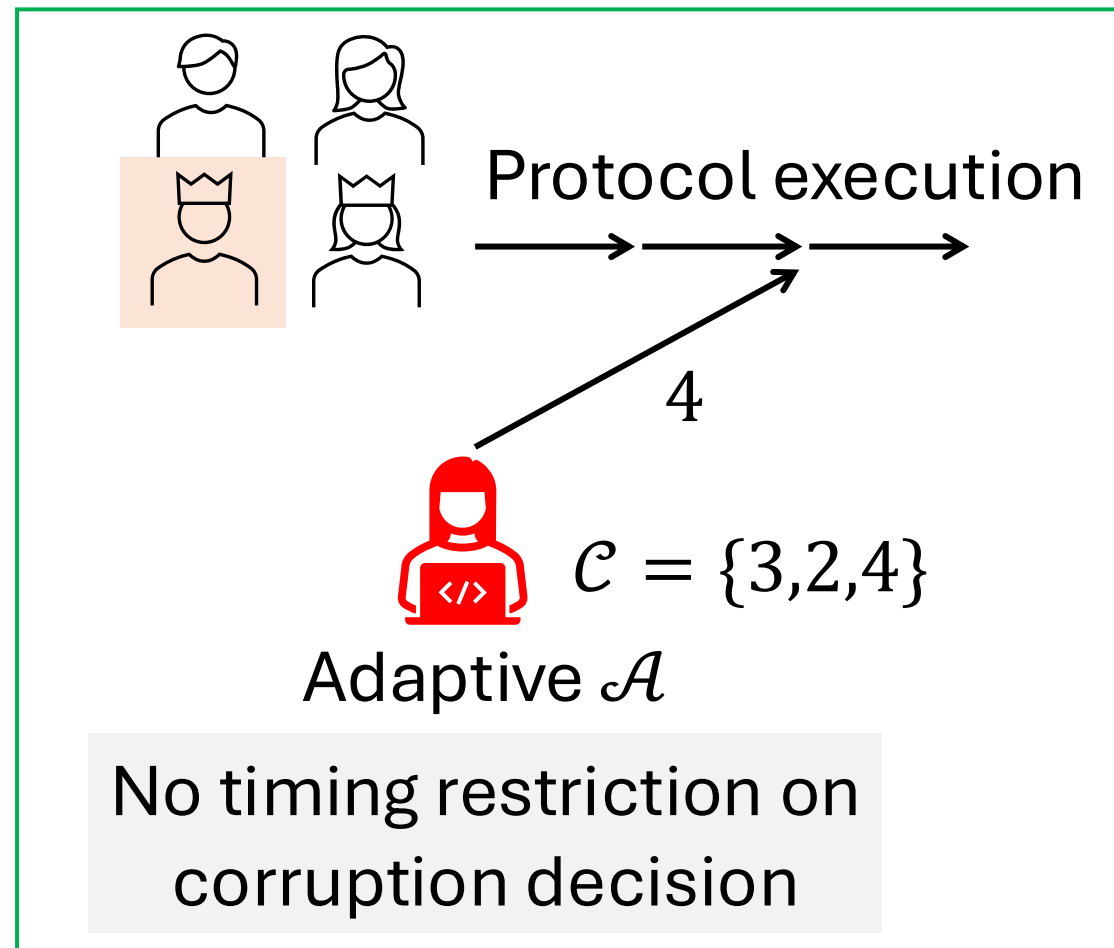
Adaptive \mathcal{A}

No timing restriction on
corruption decision

Threat model: Static vs Adaptive Corruption

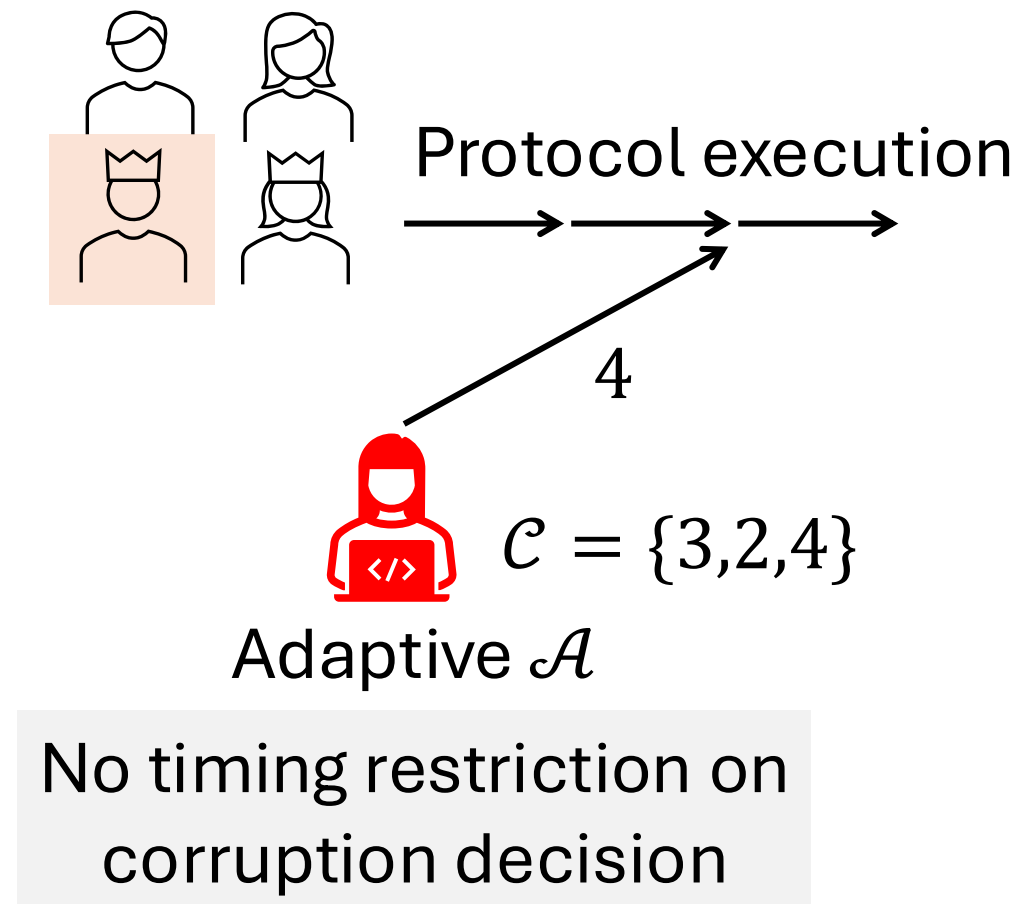
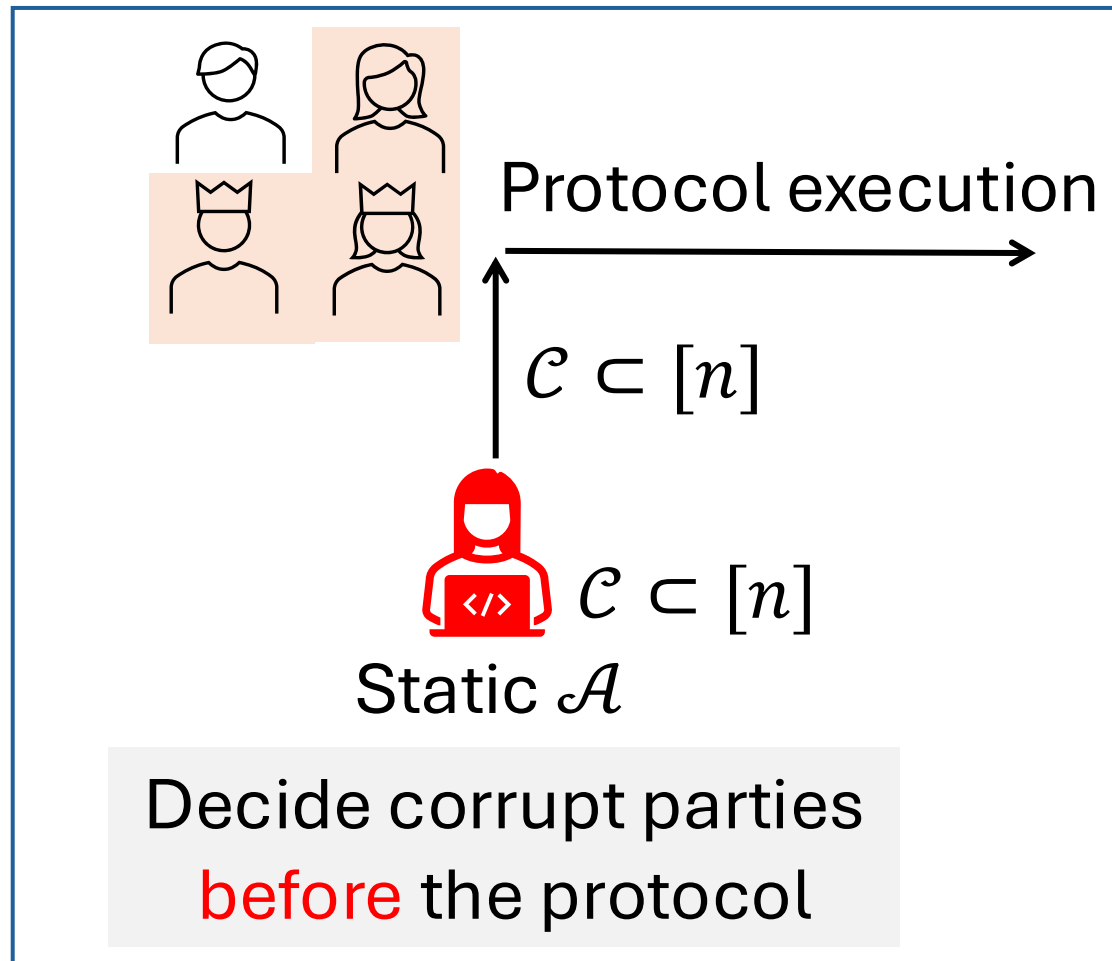


Decide corrupt parties
before the protocol



No timing restriction on
corruption decision

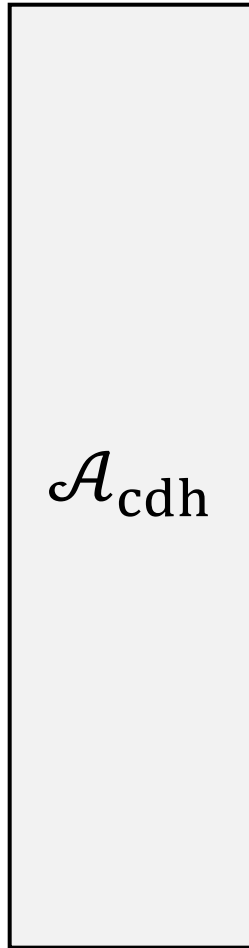
Threat model: Static vs Adaptive Corruption



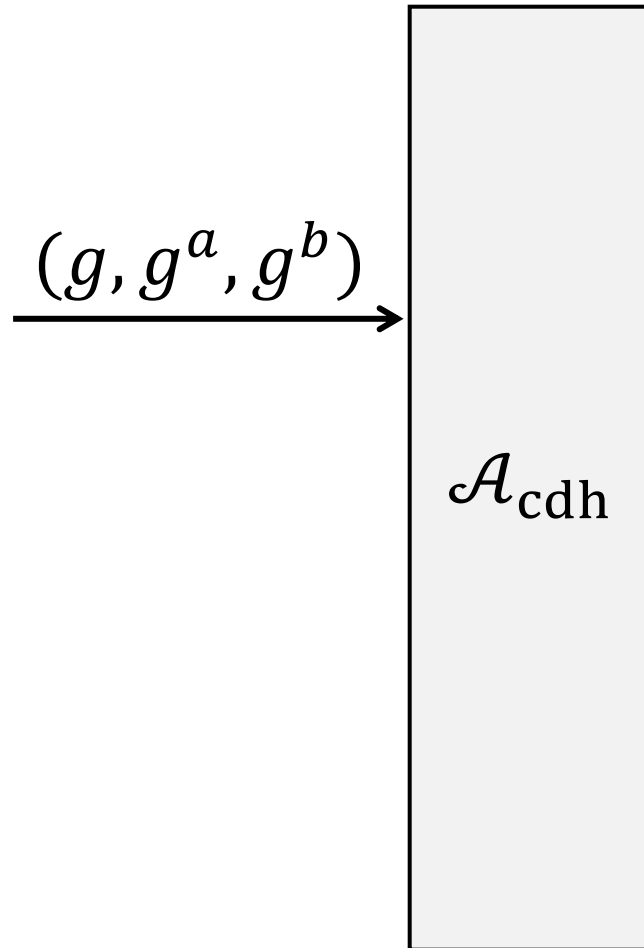
Static Security of Threshold BLS Signature

High-level framework

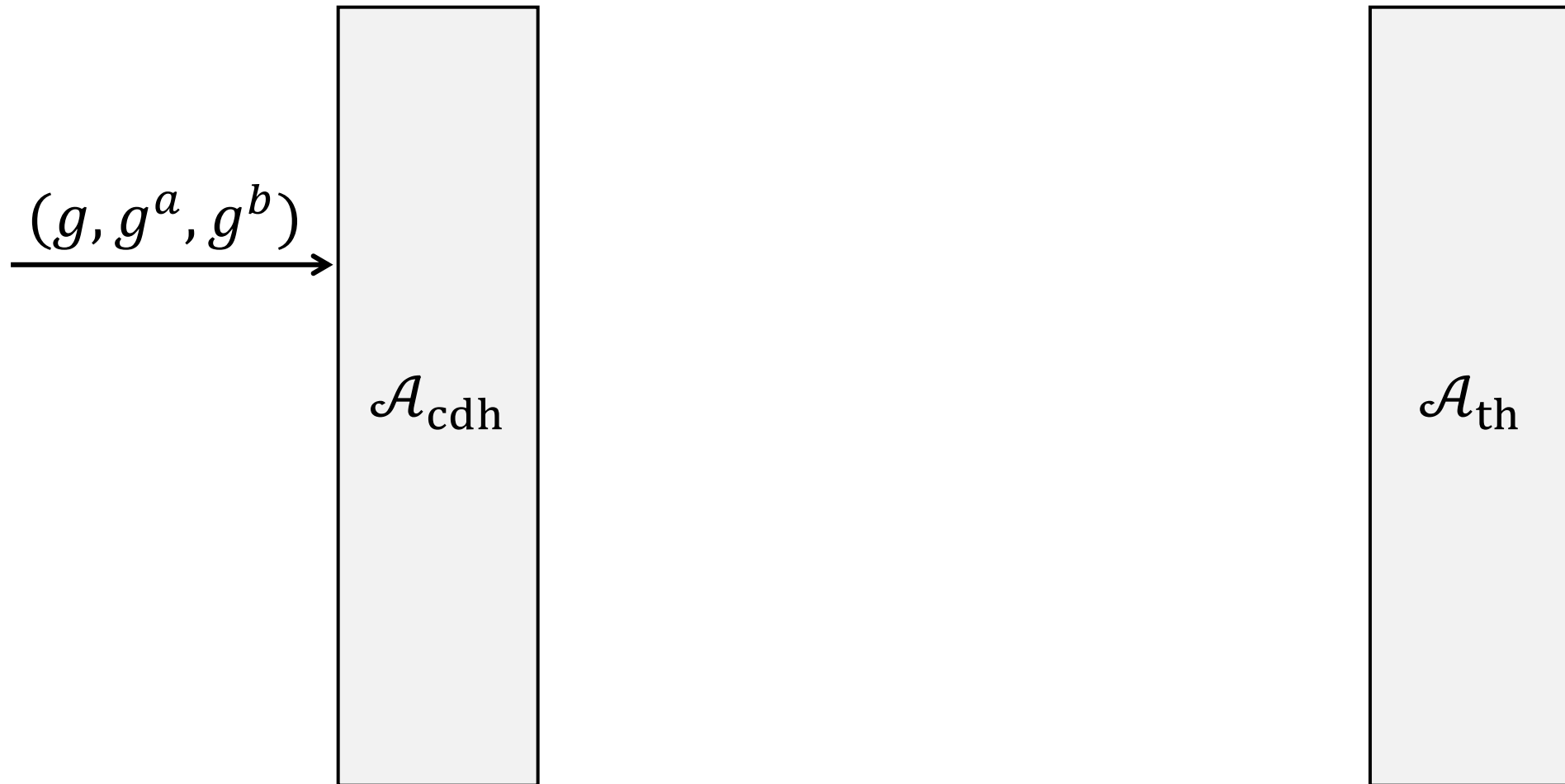
High-level framework



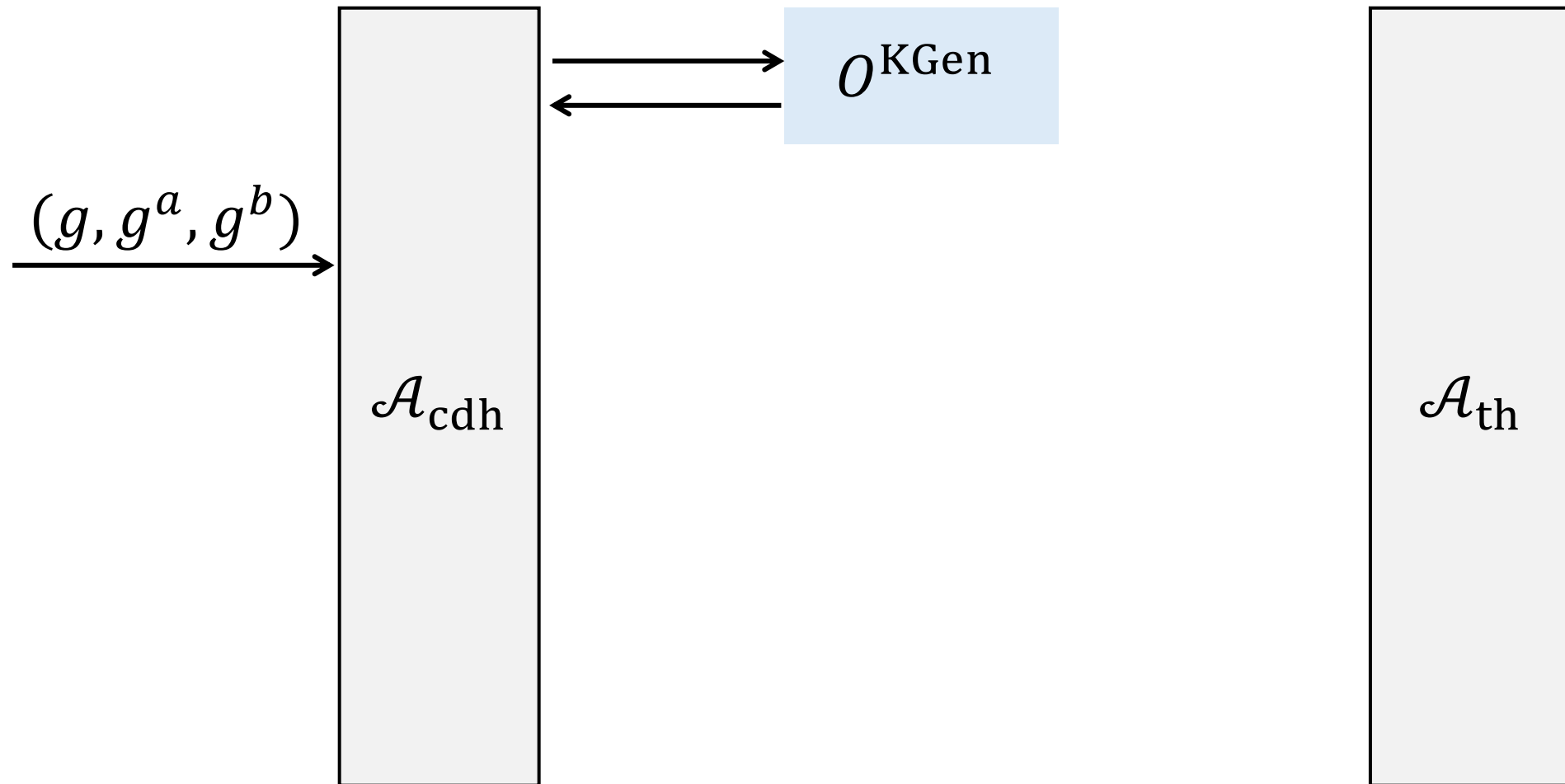
High-level framework



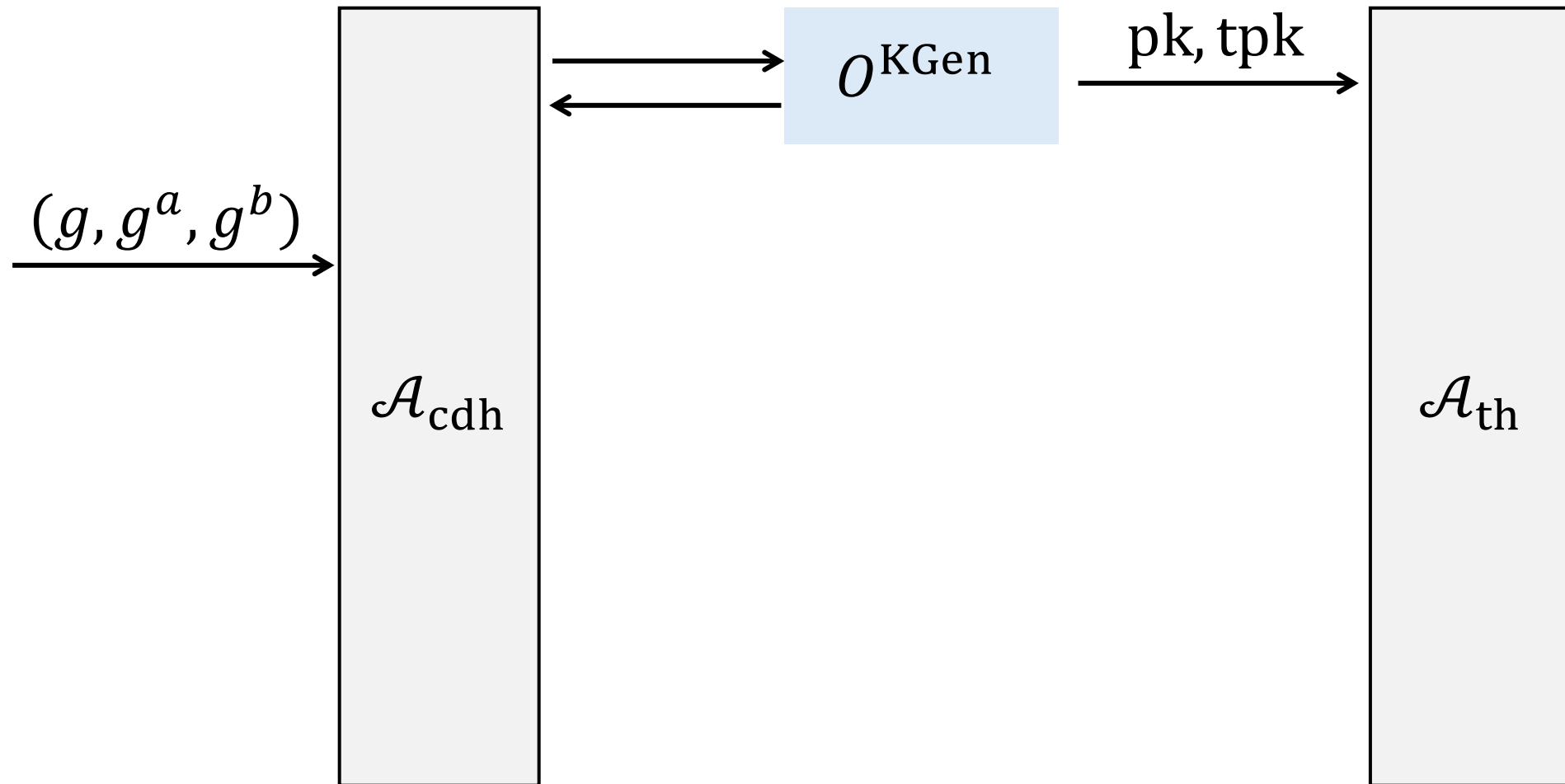
High-level framework



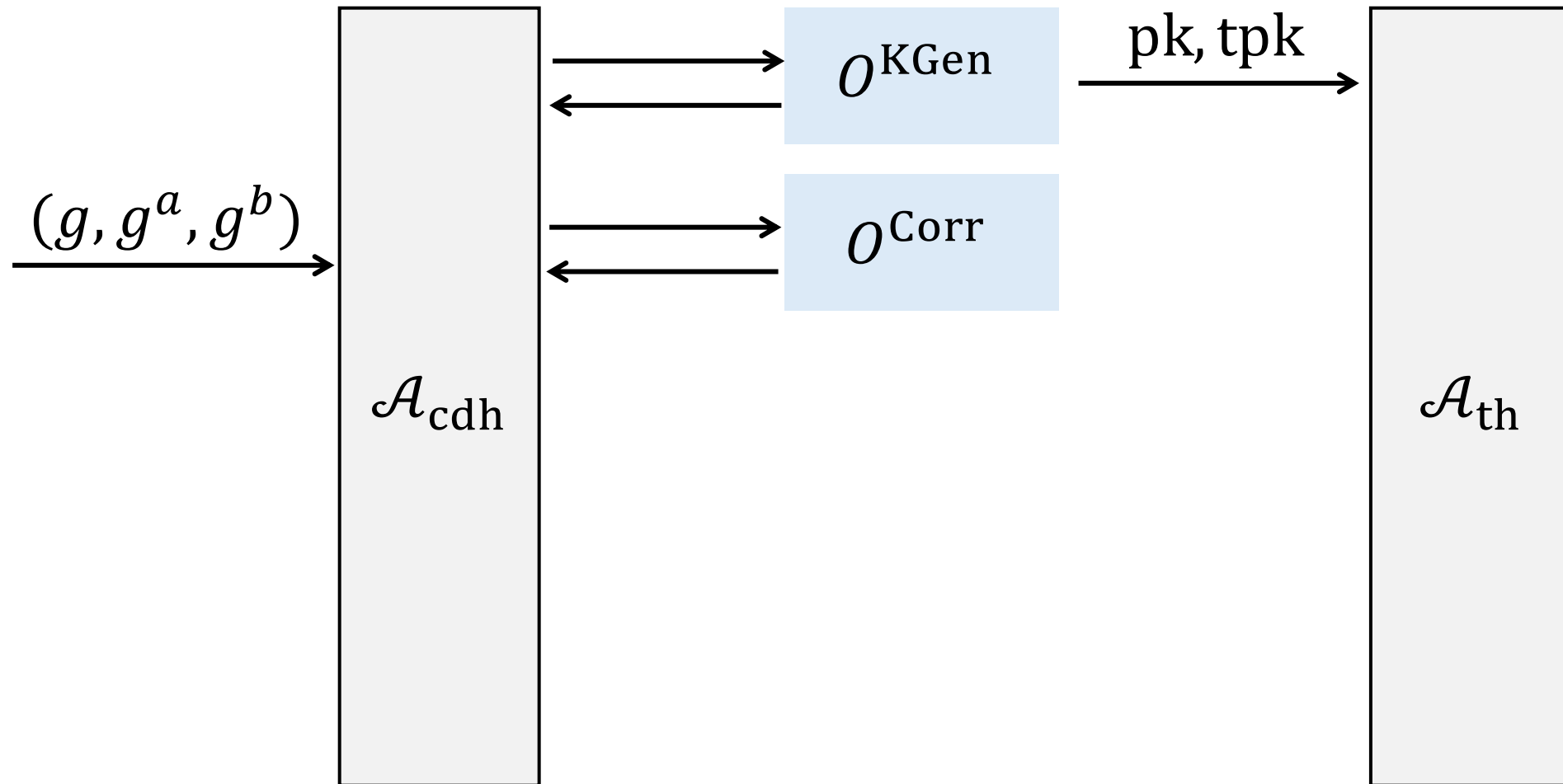
High-level framework



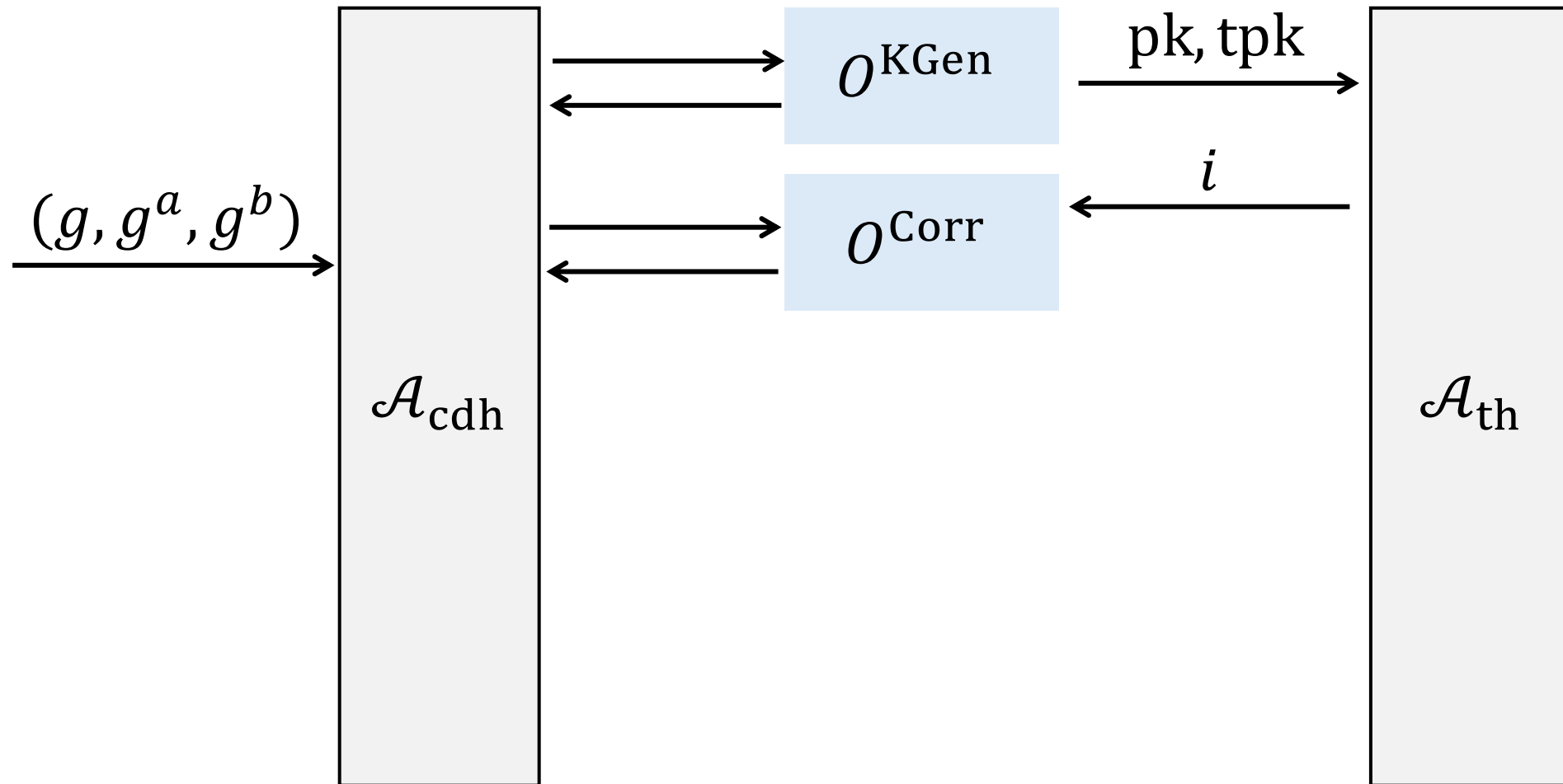
High-level framework



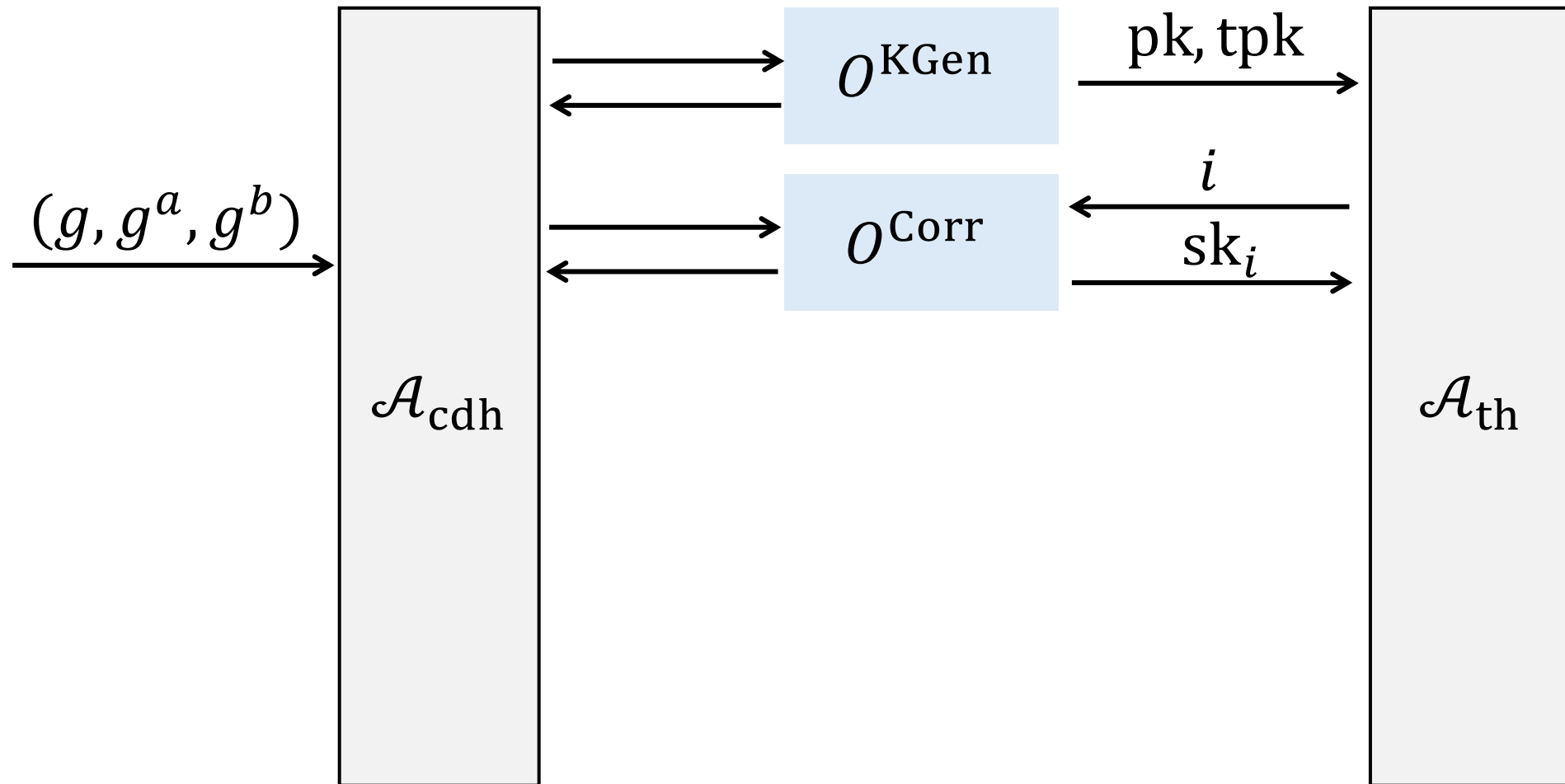
High-level framework



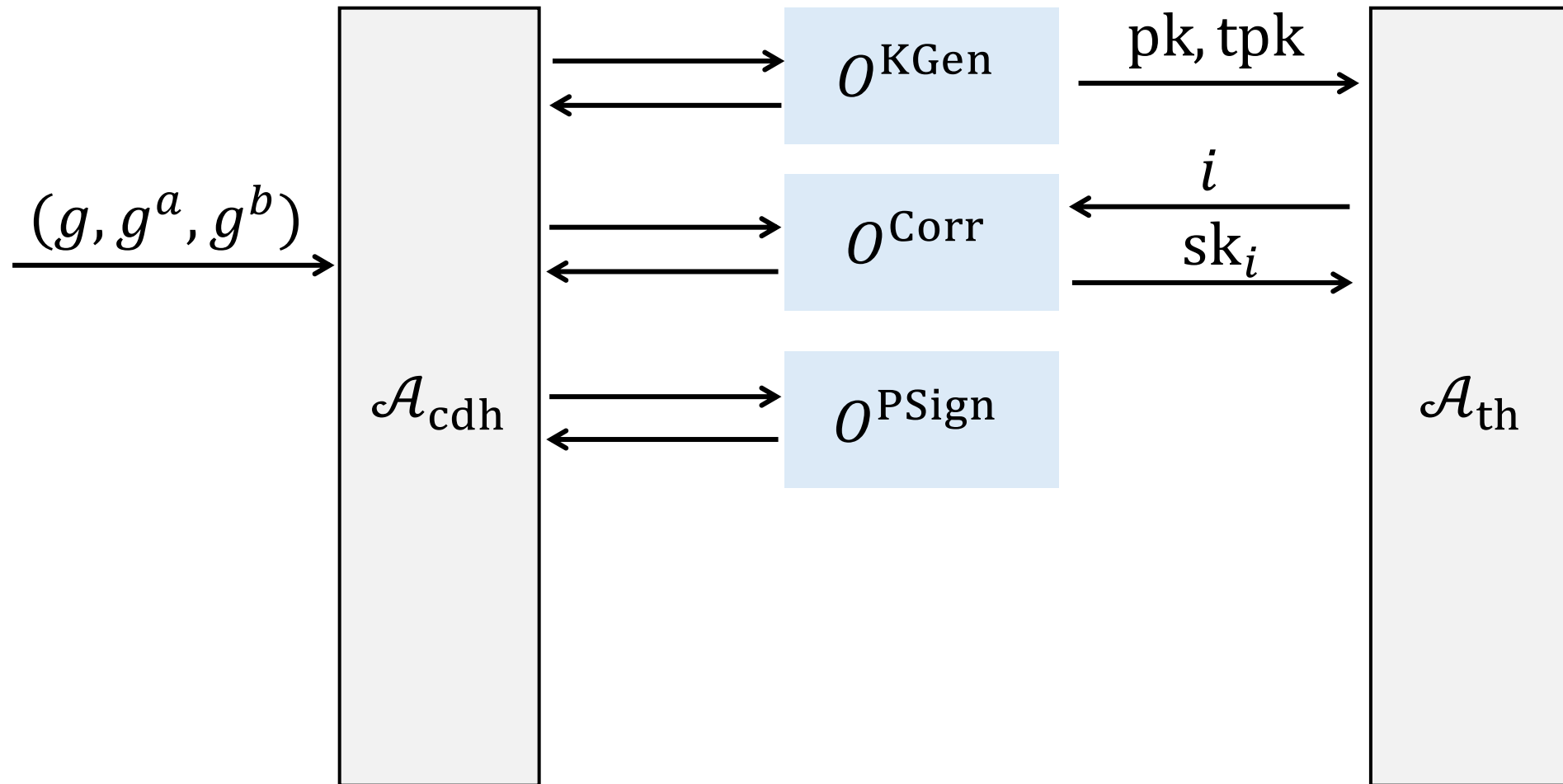
High-level framework



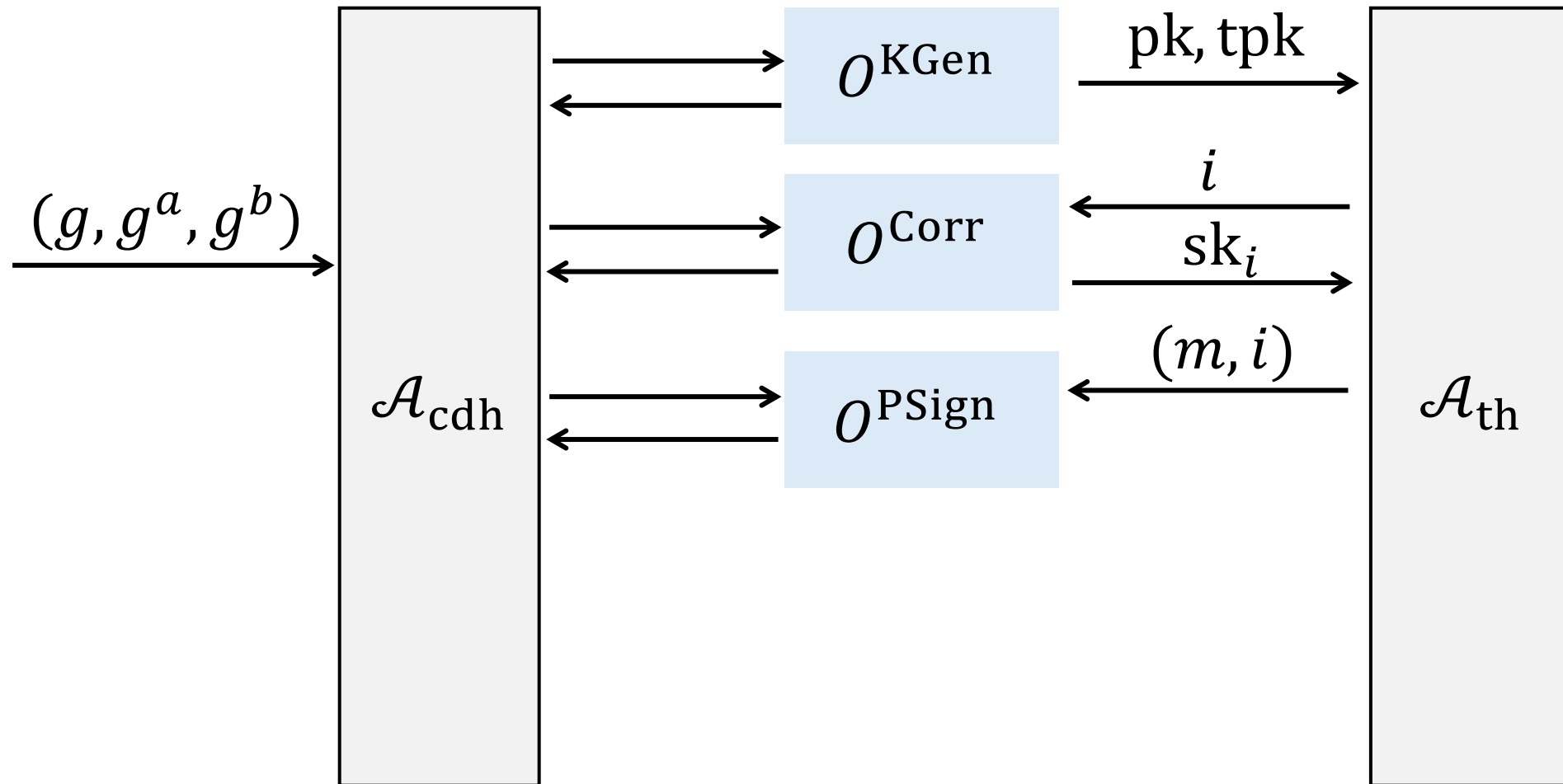
High-level framework



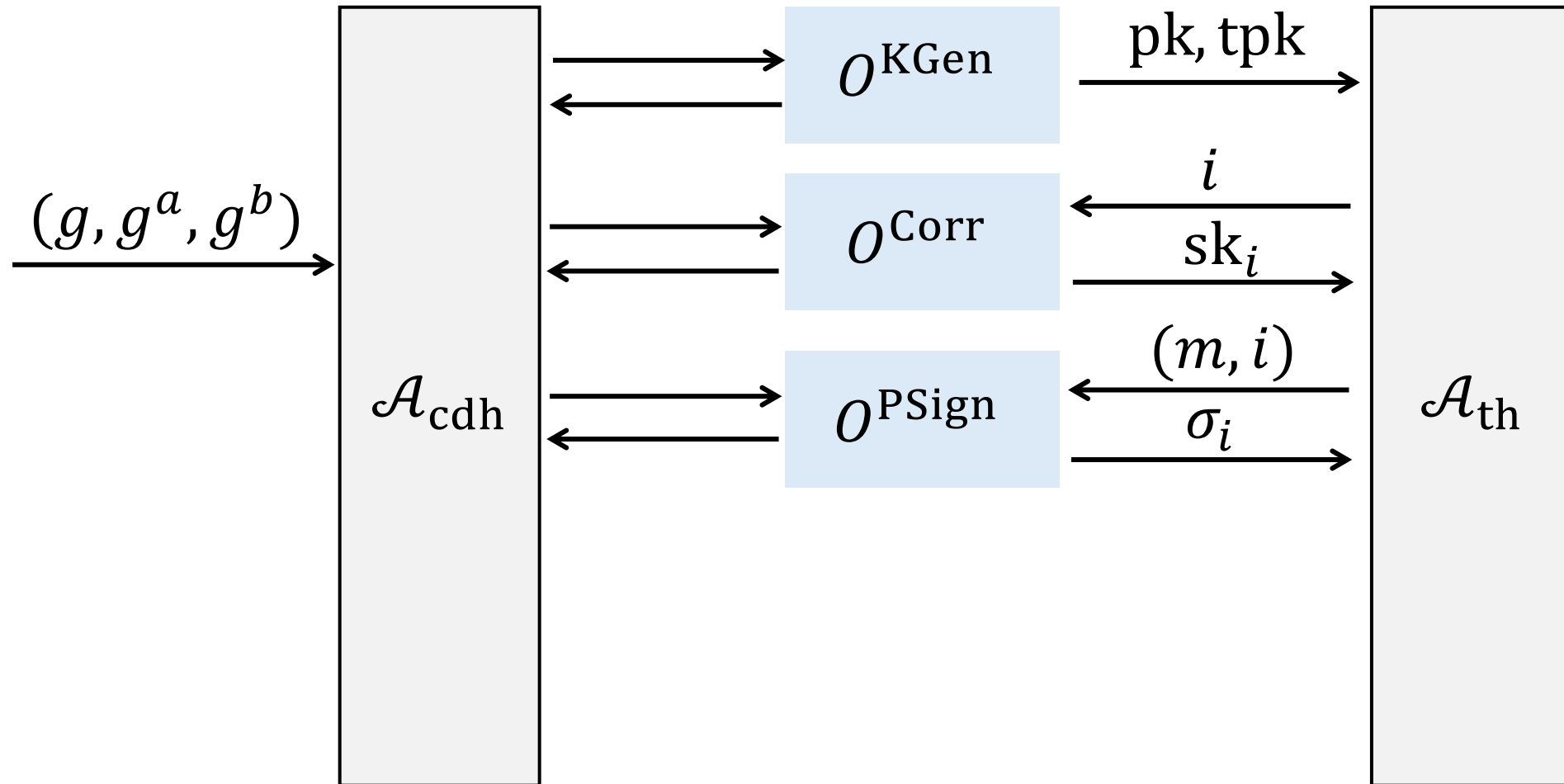
High-level framework



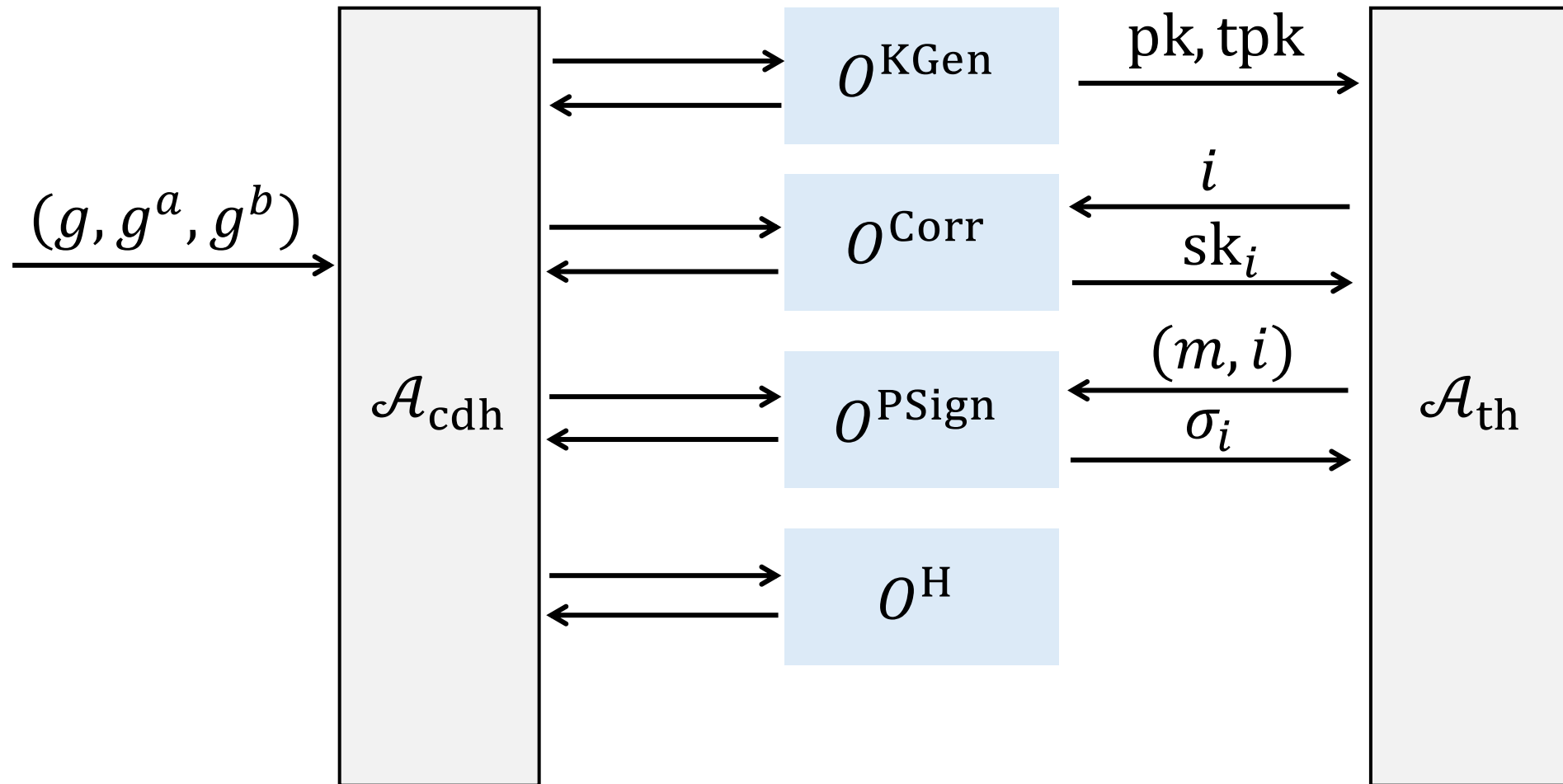
High-level framework



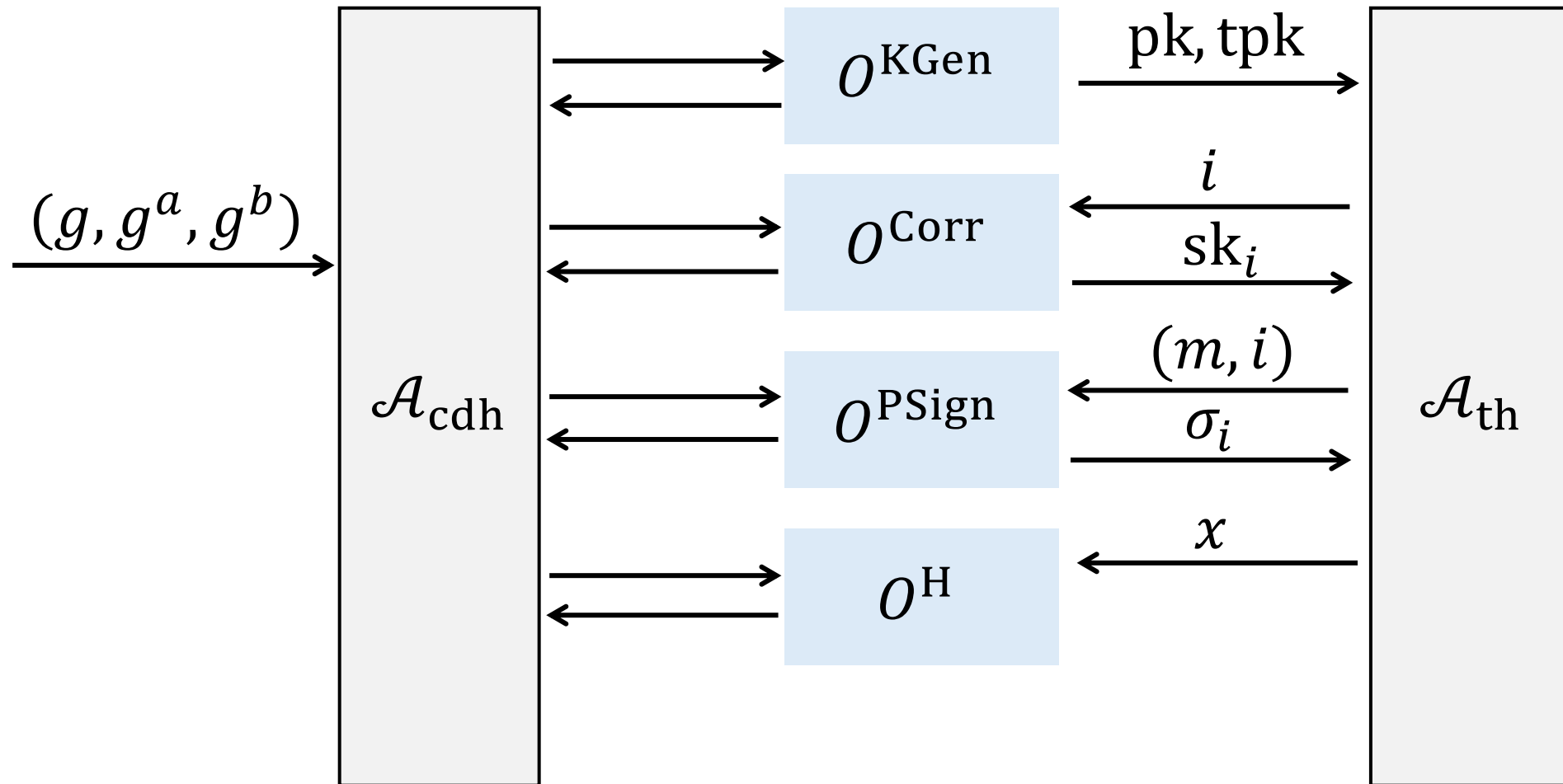
High-level framework



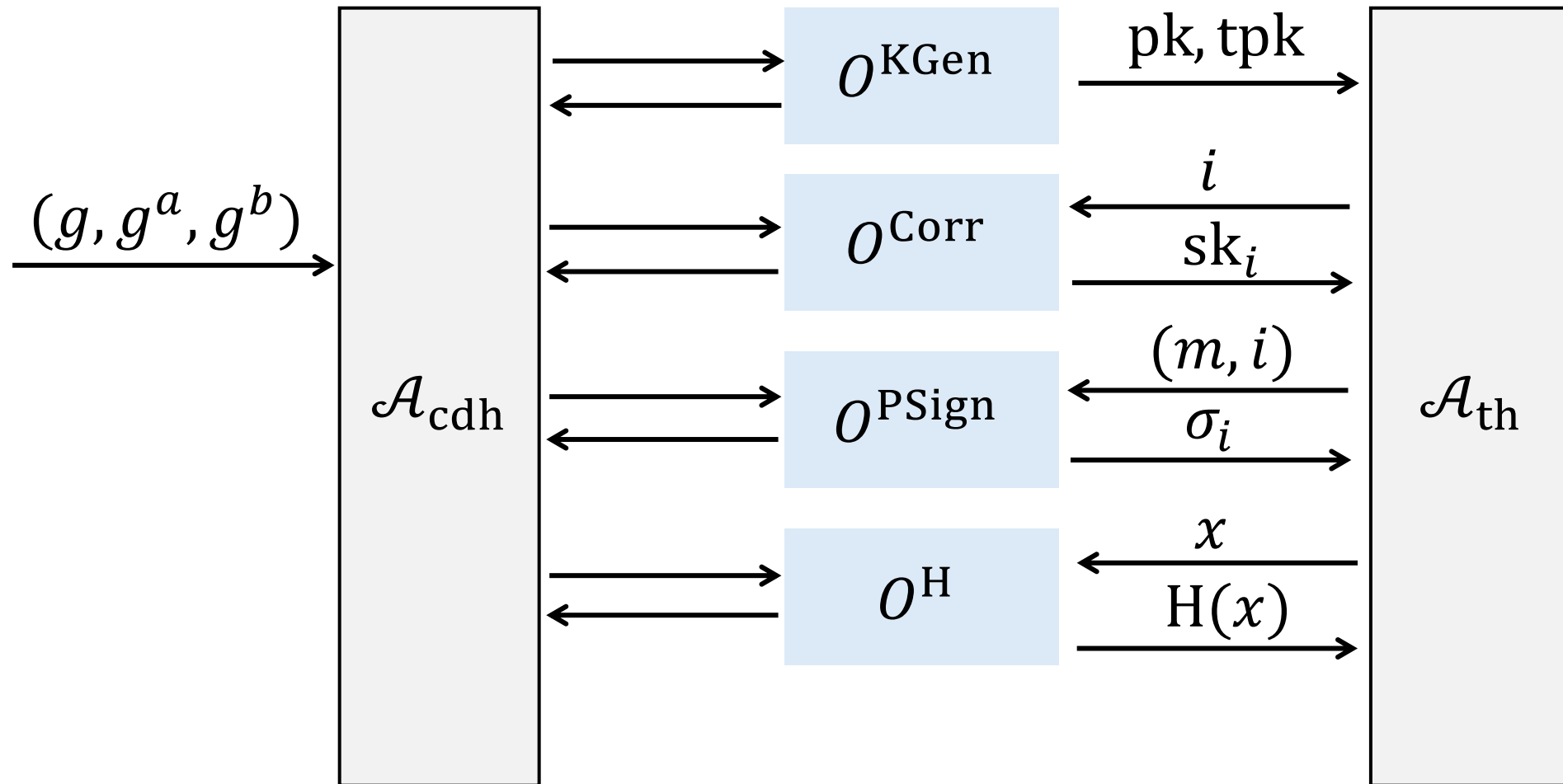
High-level framework



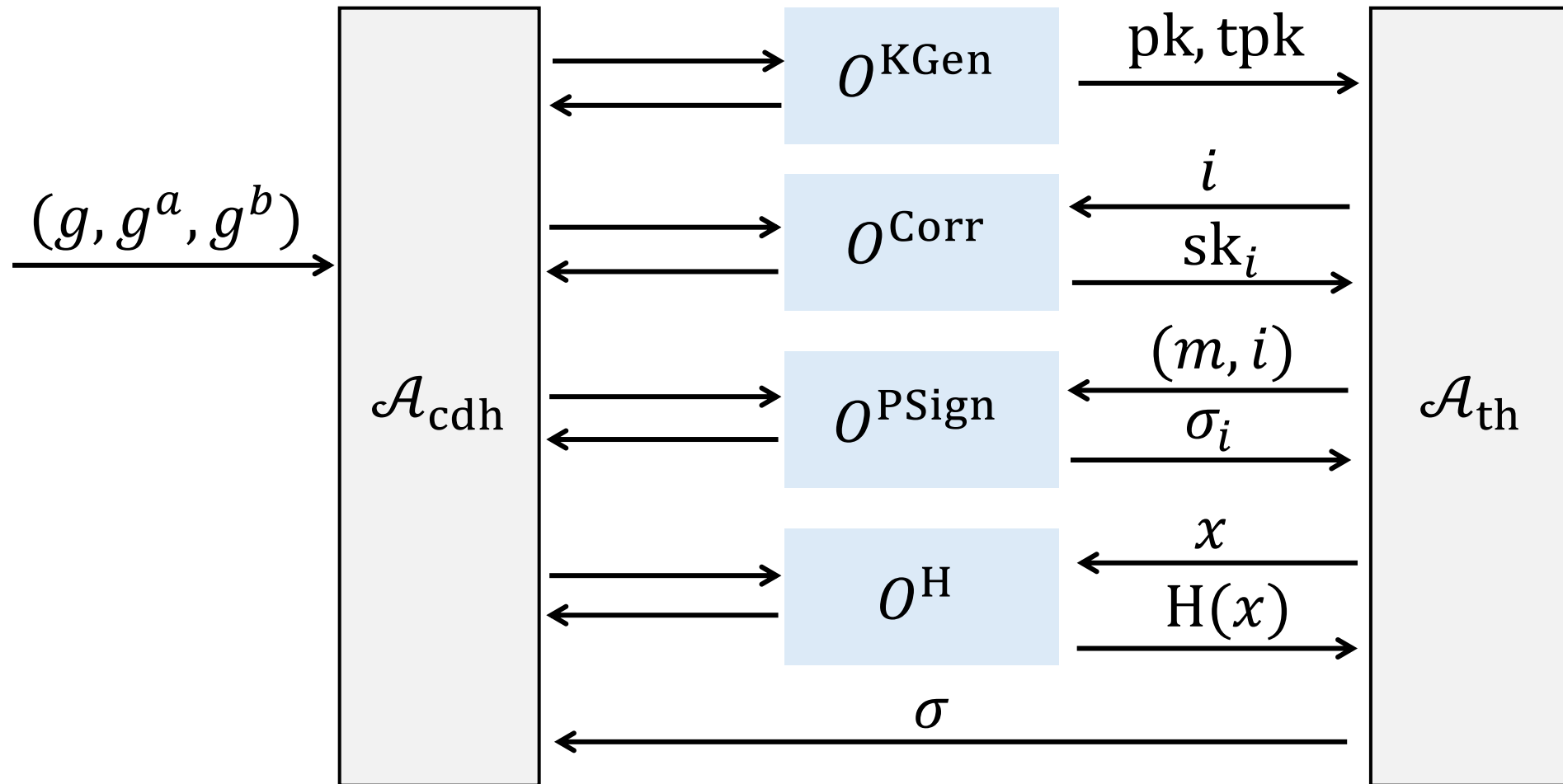
High-level framework



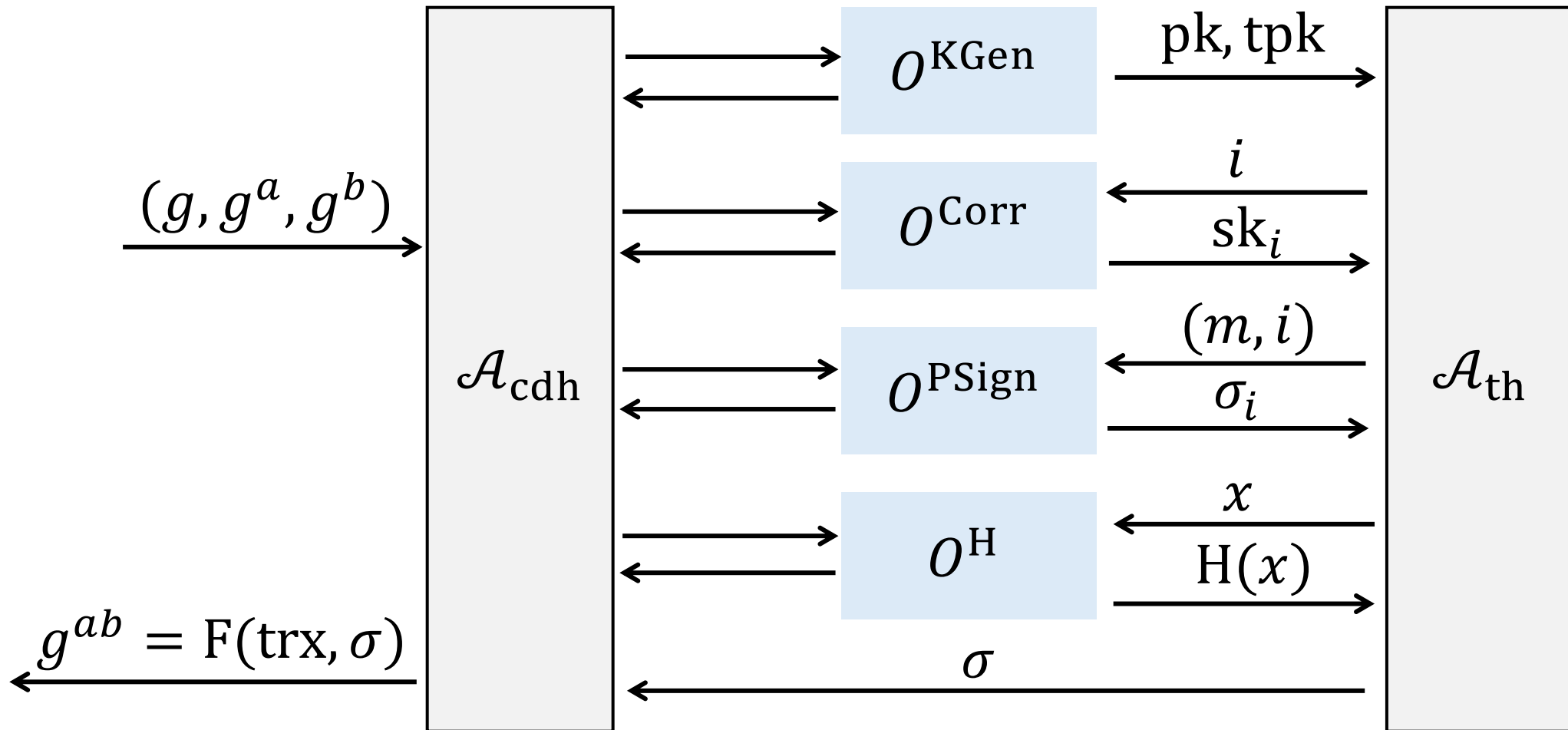
High-level framework



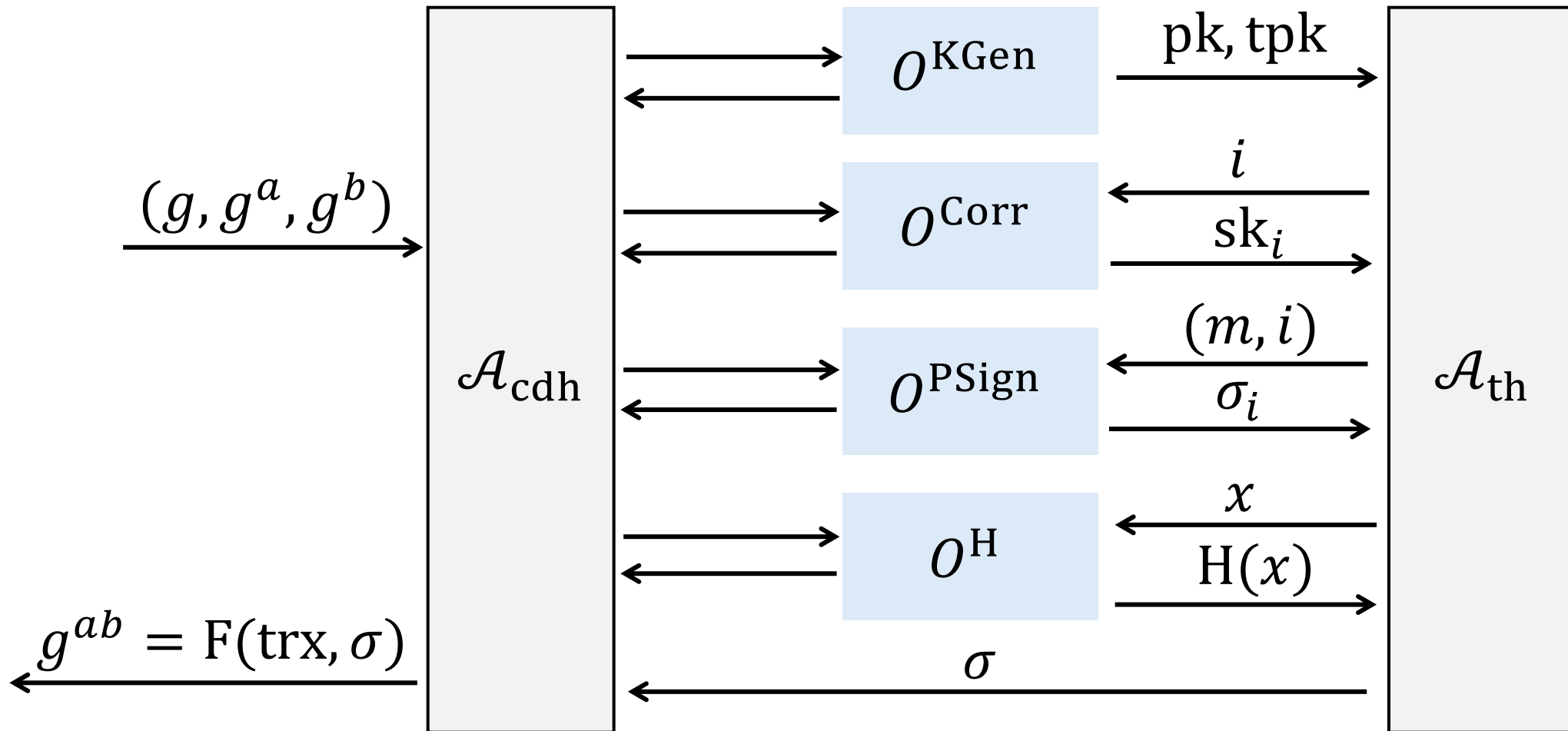
High-level framework



High-level framework



High-level framework



O^{Corr} is the trickiest to simulate

Static Security of Threshold BLS: Breaking CDH

Static Security of Threshold BLS: Breaking CDH

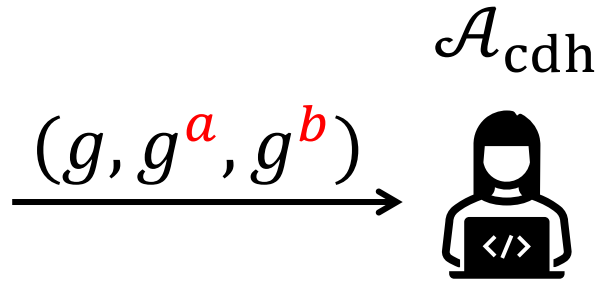
A_{cdh}



A_{th}

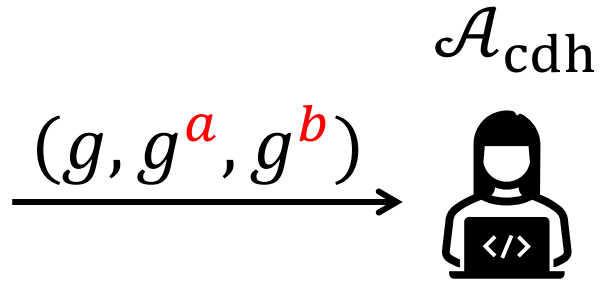


Static Security of Threshold BLS: Breaking CDH

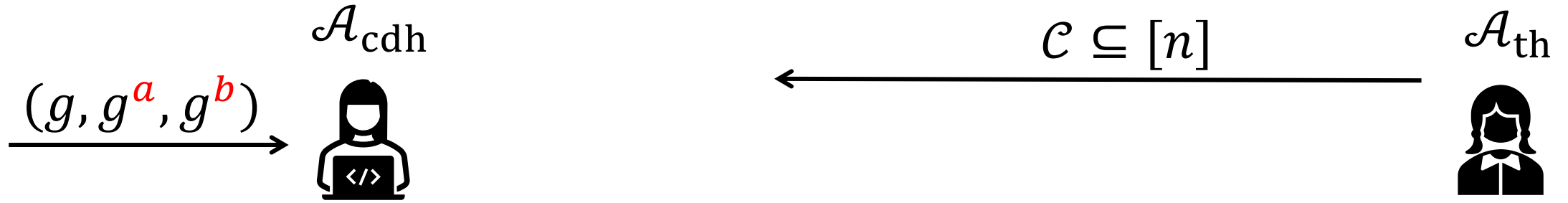


Simulating Corruption Queries

Simulating Corruption Queries

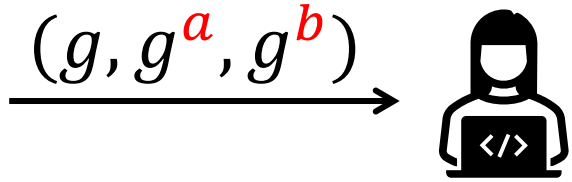


Simulating Corruption Queries



Simulating Corruption Queries

\mathcal{A}_{cdh}



- Let $\mathcal{C} = \{1, 2, \dots, t\}$

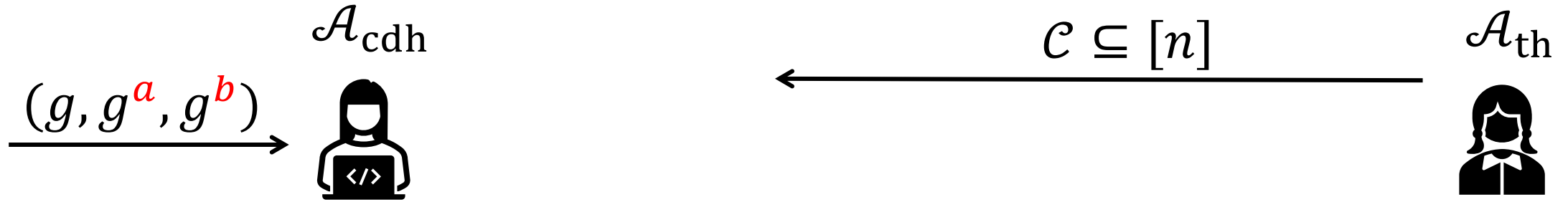
$\mathcal{C} \subseteq [n]$



\mathcal{A}_{th}

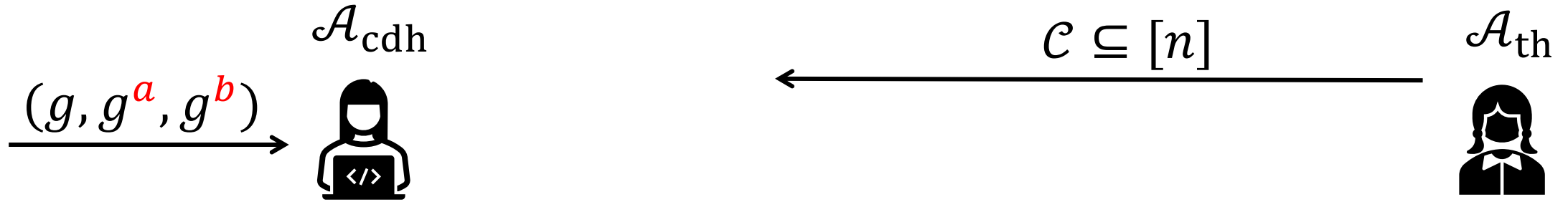


Simulating Corruption Queries



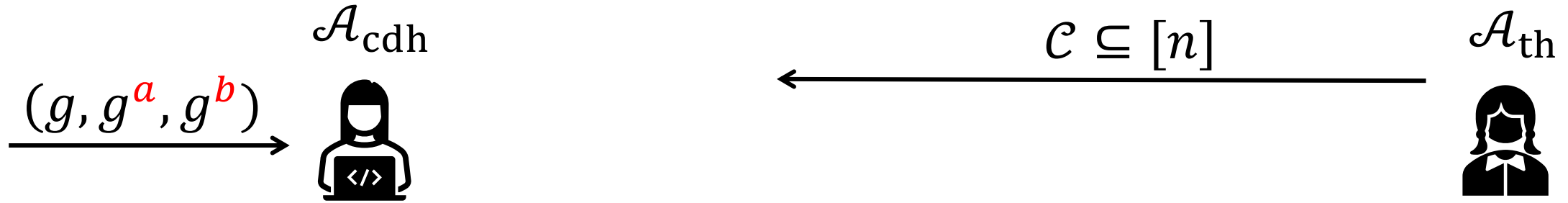
- Let $\mathcal{C} = \{1, 2, \dots, t\}$
- Sample $s(1), \dots, s(t) \leftarrow \mathbb{F}$

Simulating Corruption Queries



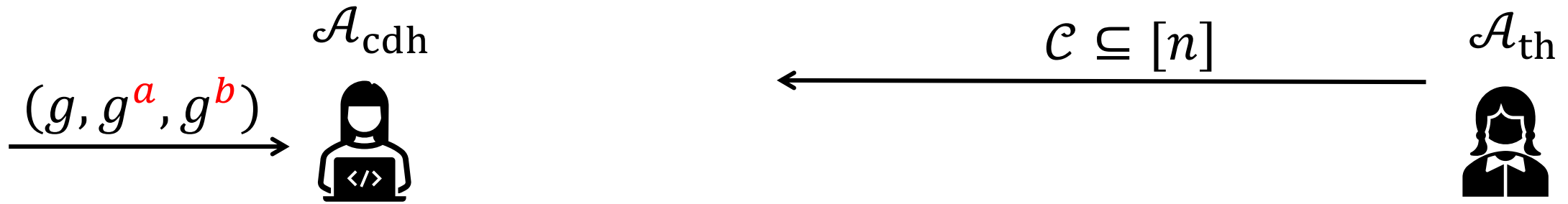
- Let $\mathcal{C} = \{1, 2, \dots, t\}$
- Sample $s(1), \dots, s(t) \leftarrow \mathbb{F}$
- Let $s(0) = a$

Simulating Corruption Queries



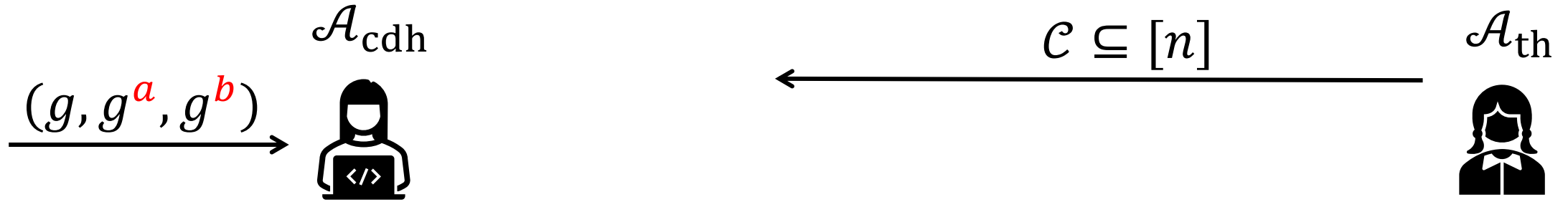
- Let $\mathcal{C} = \{1, 2, \dots, t\}$
- Sample $s(1), \dots, s(t) \leftarrow \mathbb{F}$
- Let $s(0) = a$
- Interpolate $\{g^a, g^{s(1)}, \dots, g^{s(t)}\}$

Simulating Corruption Queries



- Let $\mathcal{C} = \{1, 2, \dots, t\}$
- Sample $s(1), \dots, s(t) \leftarrow \mathbb{F}$
- Let $s(0) = a$
- Interpolate $\{g^a, g^{s(1)}, \dots, g^{s(t)}\}$
to compute $\text{tpk} = [g^{s(1)}, \dots, g^{s(n)}]$

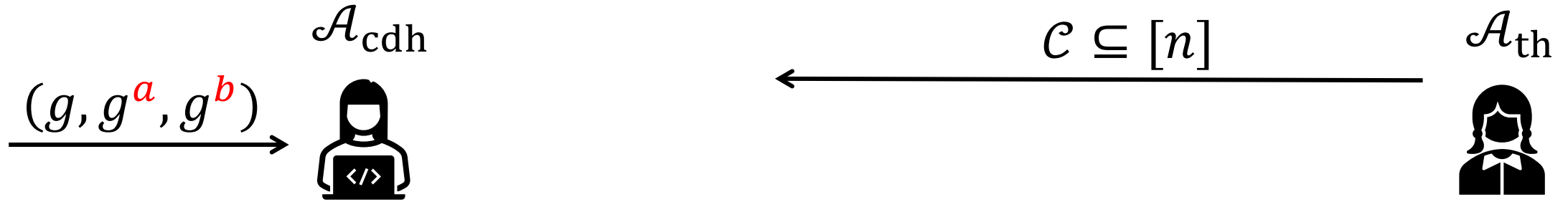
Simulating Corruption Queries



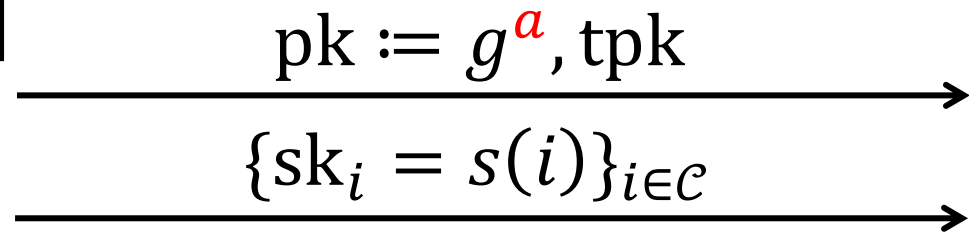
- Let $\mathcal{C} = \{1, 2, \dots, t\}$
- Sample $s(1), \dots, s(t) \leftarrow \mathbb{F}$
- Let $s(0) = a$
- Interpolate $\{g^a, g^{s(1)}, \dots, g^{s(t)}\}$
to compute $\text{tpk} = [g^{s(1)}, \dots, g^{s(n)}]$

$\text{pk} := g^a, \text{tpk}$ $\xrightarrow{\quad}$

Simulating Corruption Queries

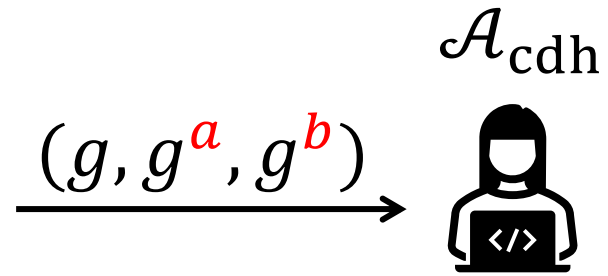


- Let $\mathcal{C} = \{1, 2, \dots, t\}$
- Sample $s(1), \dots, s(t) \leftarrow \mathbb{F}$
- Let $s(0) = a$
- Interpolate $\{g^a, g^{s(1)}, \dots, g^{s(t)}\}$
to compute $\text{tpk} = [g^{s(1)}, \dots, g^{s(n)}]$

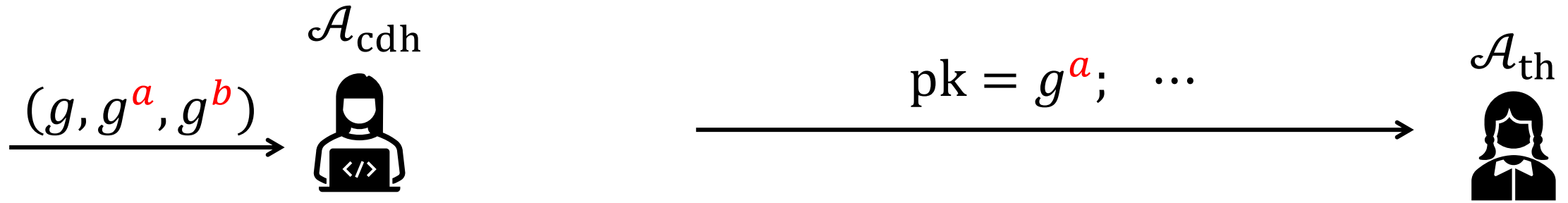


Simulating Signing Queries

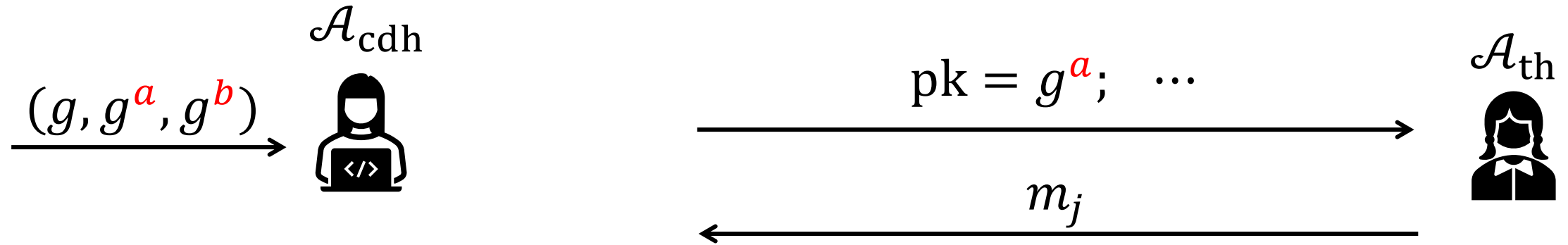
Simulating Signing Queries



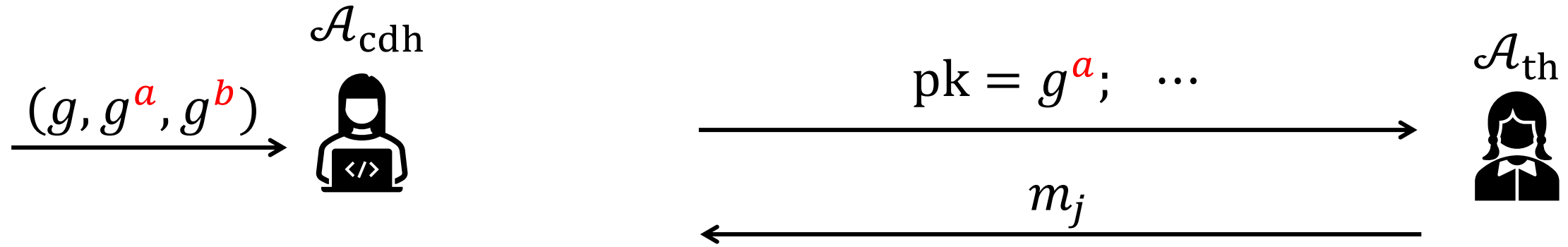
Simulating Signing Queries



Simulating Signing Queries

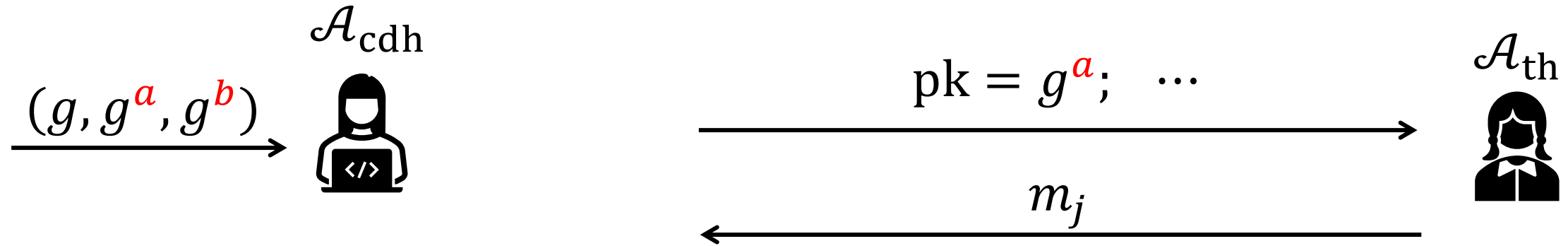


Simulating Signing Queries



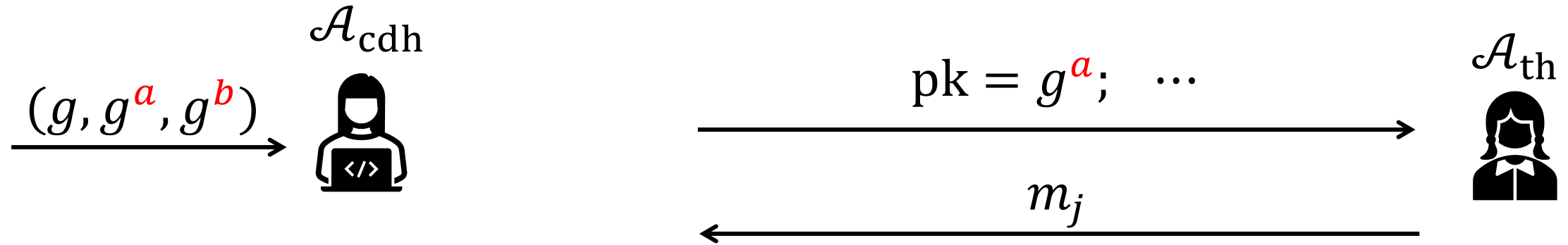
1. Sample $x_j \leftarrow \mathbb{F}$

Simulating Signing Queries



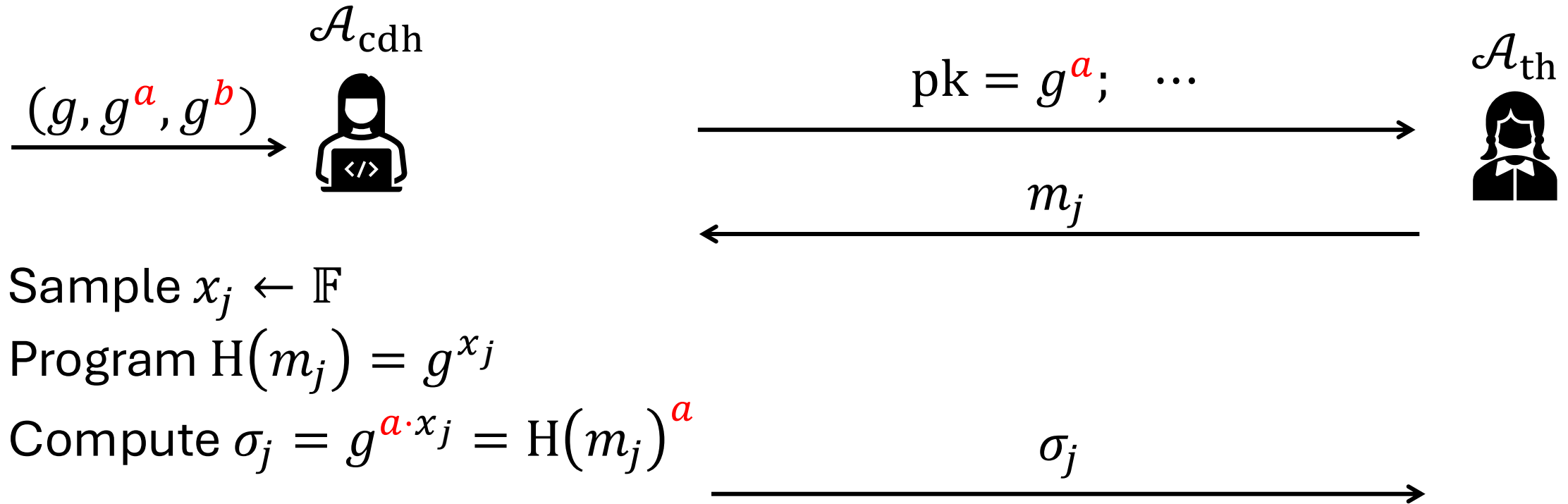
1. Sample $x_j \leftarrow \mathbb{F}$
2. Program $H(m_j) = g^{x_j}$

Simulating Signing Queries

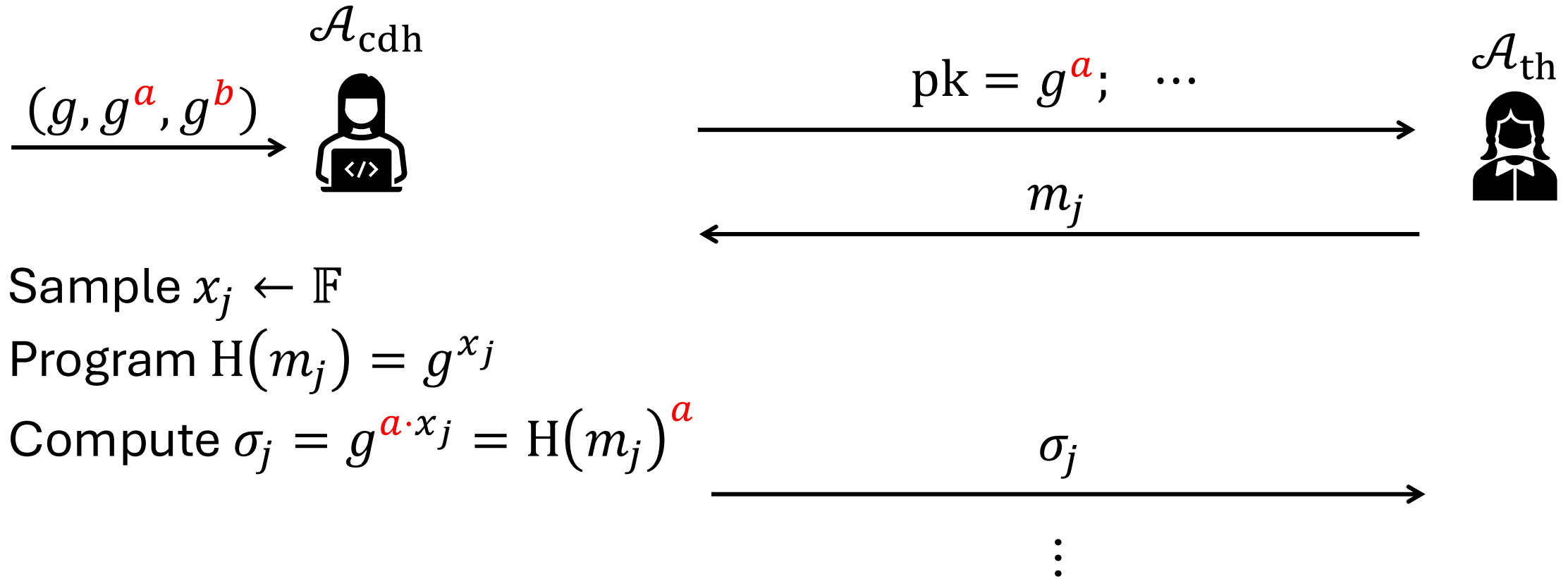


1. Sample $x_j \leftarrow \mathbb{F}$
2. Program $H(m_j) = g^{x_j}$
3. Compute $\sigma_j = g^{a \cdot x_j} = H(m_j)^a$

Simulating Signing Queries

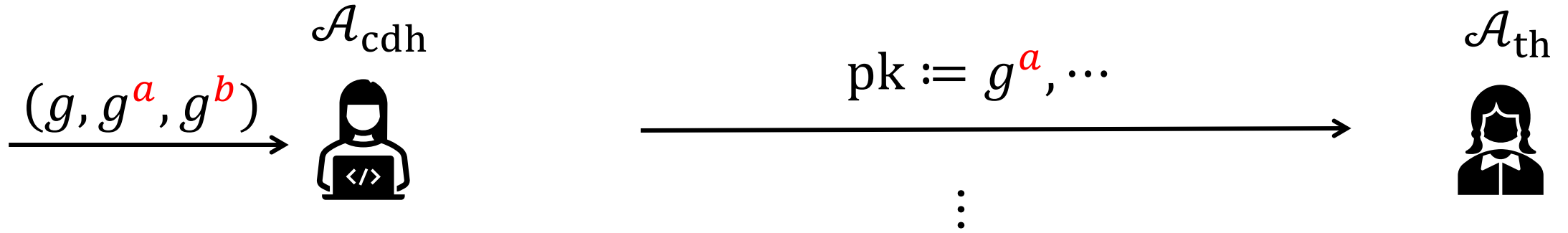


Simulating Signing Queries

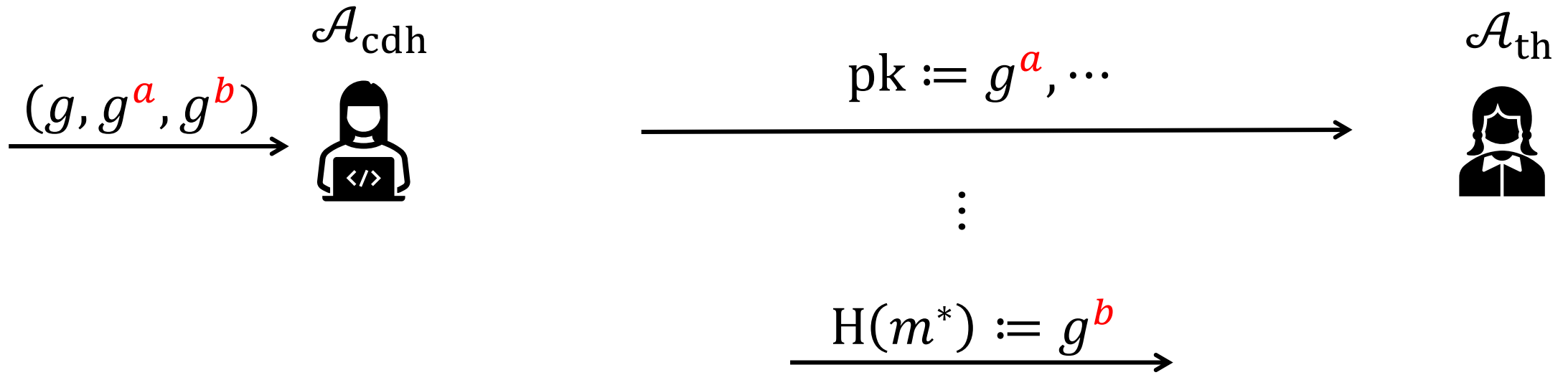


Breaking CDH

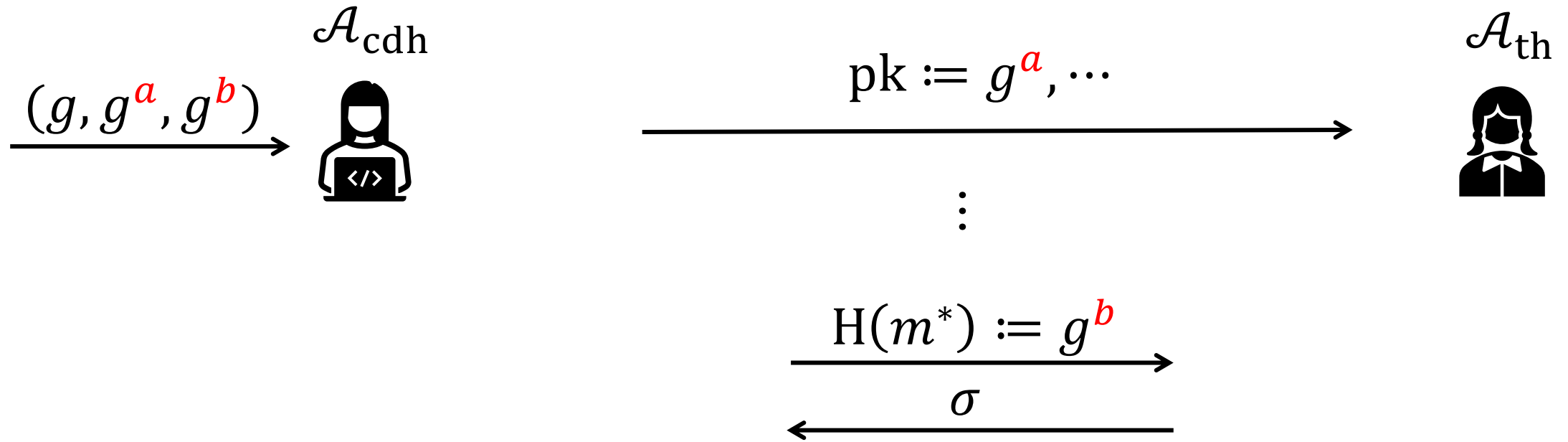
Breaking CDH



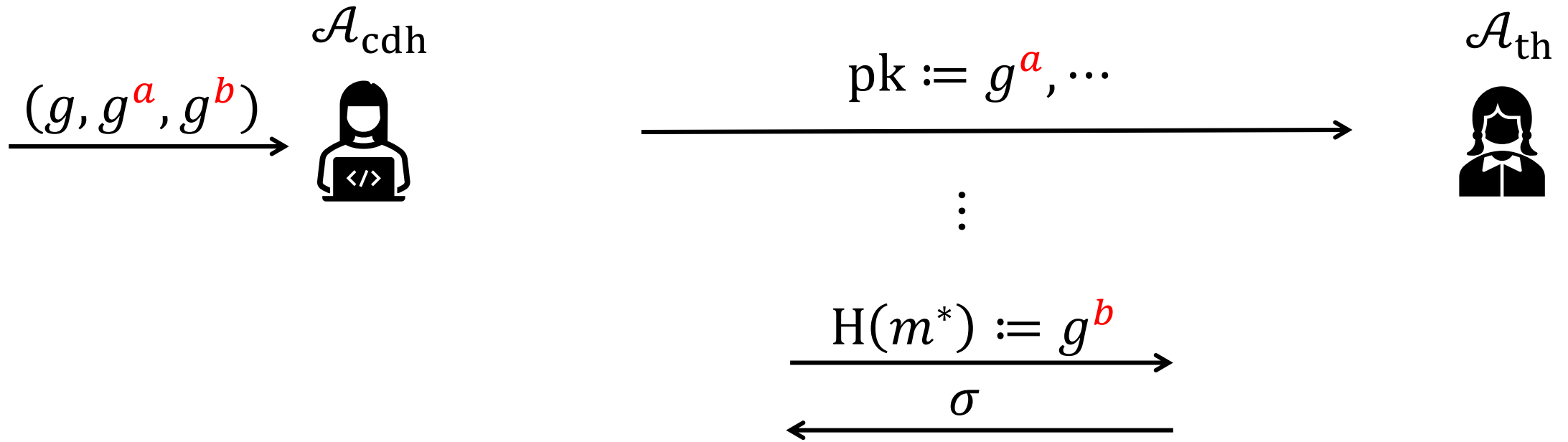
Breaking CDH



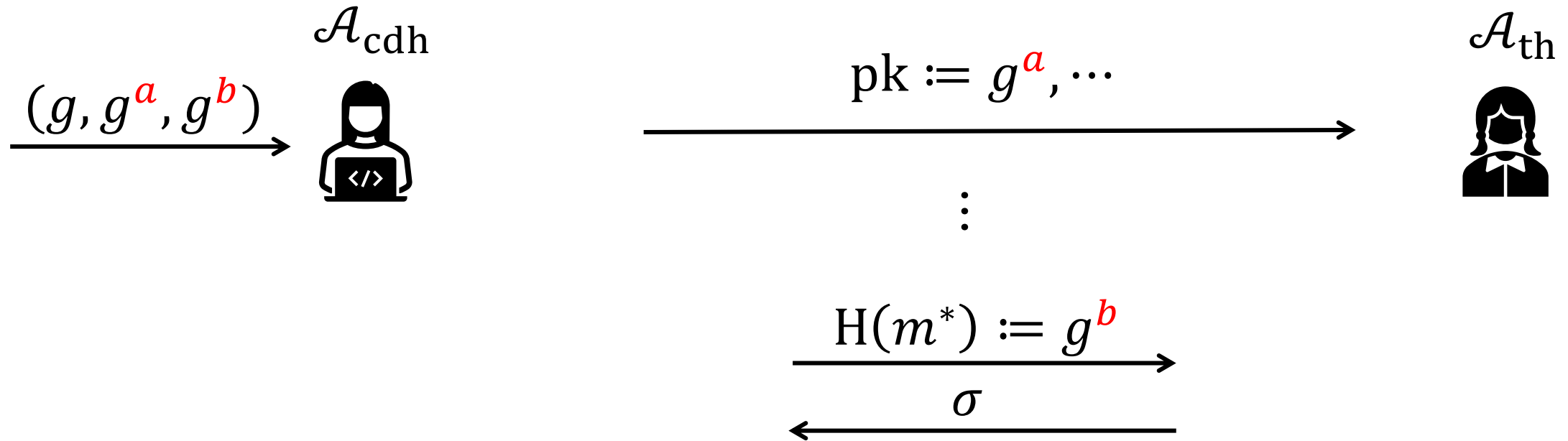
Breaking CDH



Breaking CDH

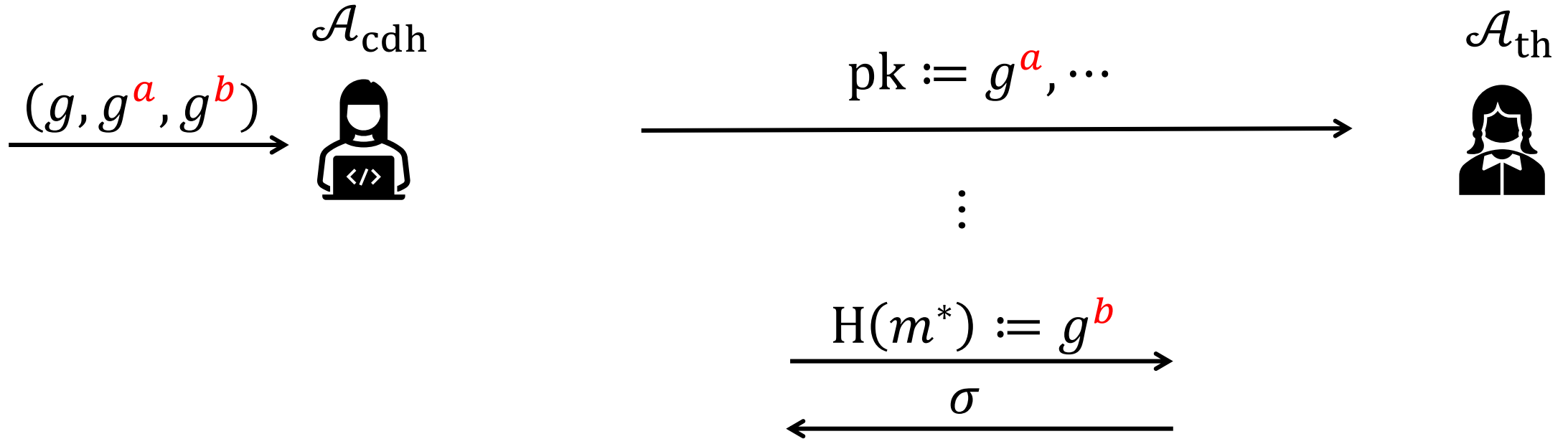


Breaking CDH



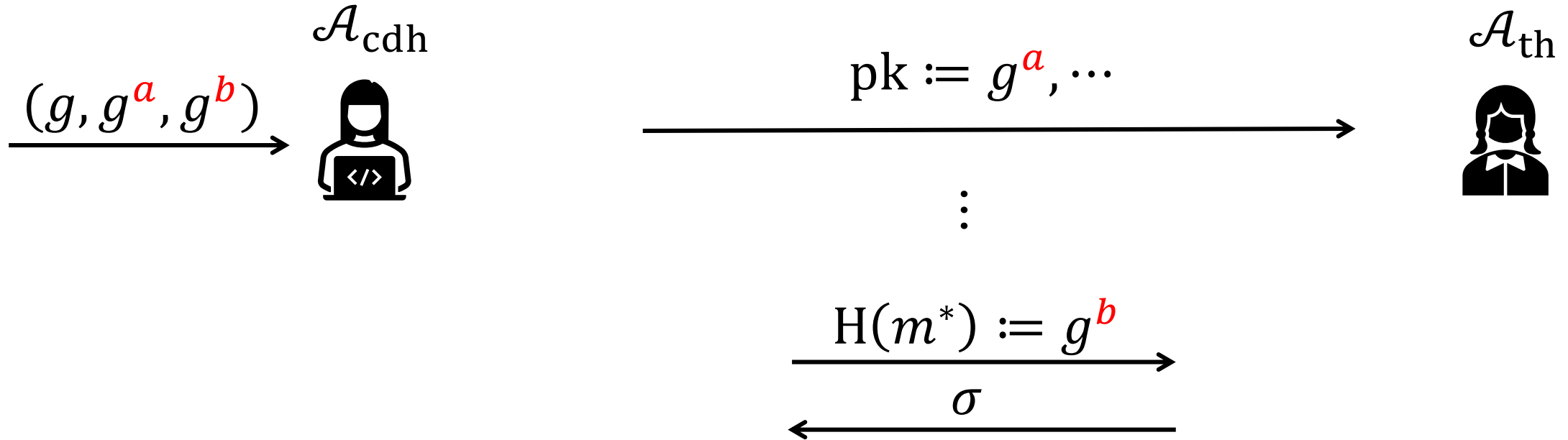
$$e(\text{pk}, \text{H}(m^*)) = e(g, \sigma)$$

Breaking CDH

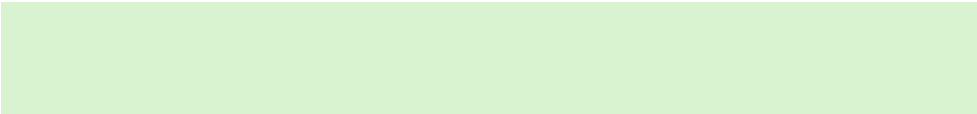


$$\begin{aligned} e(\text{pk}, \text{H}(m^*)) &= e(g, \sigma) \\ \Rightarrow e(g^a, g^b) &= e(g, \sigma) \Rightarrow \sigma = g^{ab} \end{aligned}$$

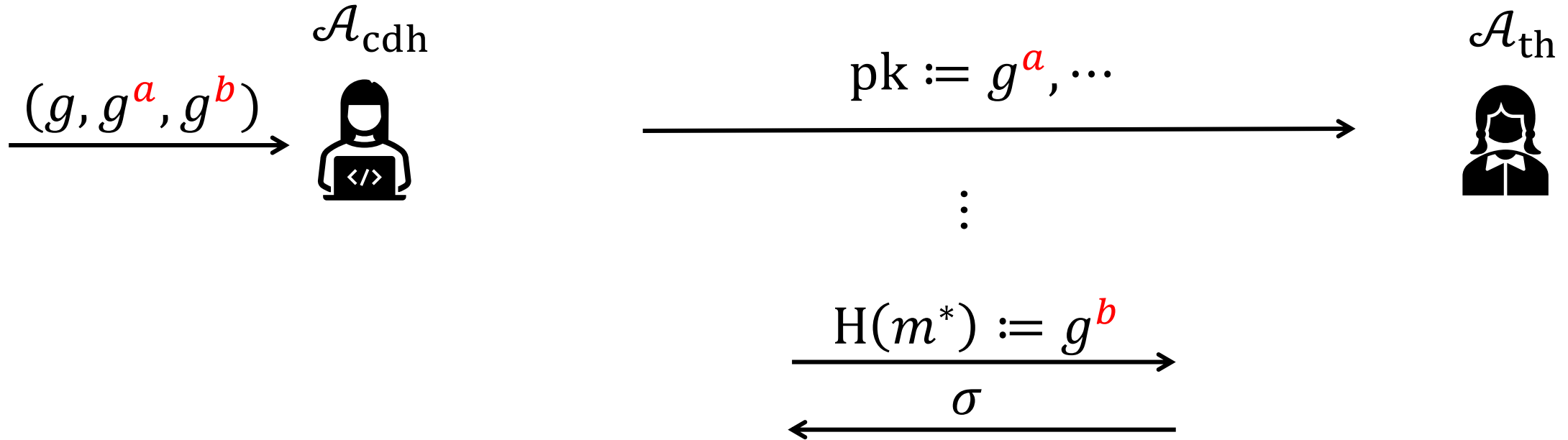
Breaking CDH



$$e(\text{pk}, \text{H}(m^*)) = e(g, \sigma)$$
$$\Rightarrow e(g^a, g^b) = e(g, \sigma) \Rightarrow \sigma = g^{ab}$$



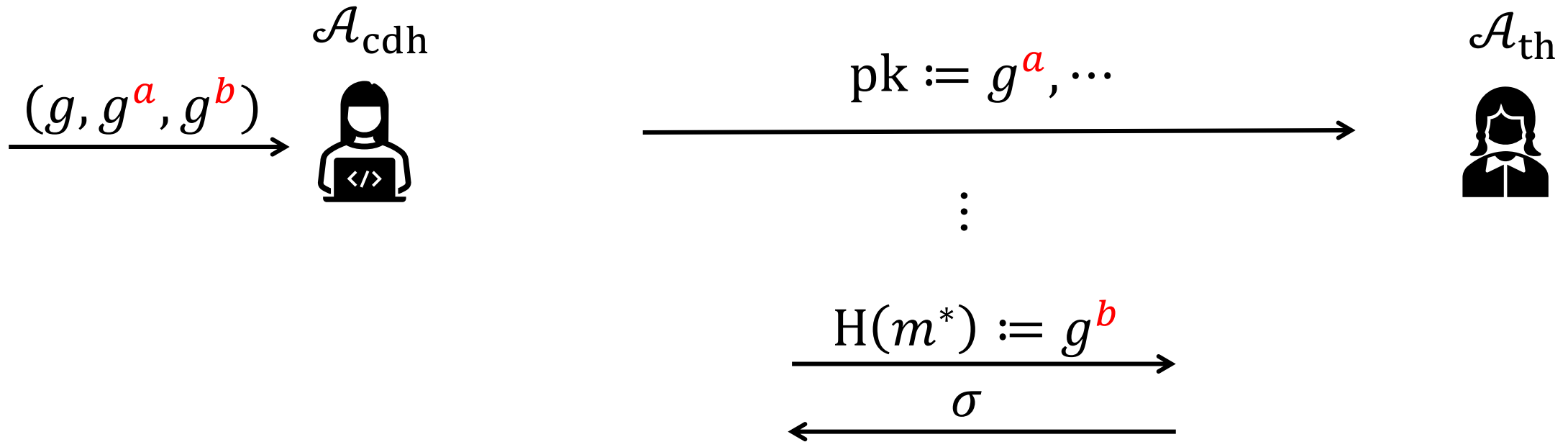
Breaking CDH



$$e(\text{pk}, \text{H}(m^*)) = e(g, \sigma)$$
$$\Rightarrow e(g^a, g^b) = e(g, \sigma) \Rightarrow \sigma = g^{ab}$$

$\sigma = g^{ab}$ is the CDH solution!

Breaking CDH



$$e(\text{pk}, \text{H}(m^*)) = e(g, \sigma)$$
$$\Rightarrow e(g^a, g^b) = e(g, \sigma) \Rightarrow \sigma = g^{ab}$$

$\sigma = g^{ab}$ is the CDH solution!

See [BL22] for adaptive security proof (OMDL in the AGM).

Evaluation Results

Some evaluation results

Some evaluation results

- Implementation in Golang (gnark-crypto)

Some evaluation results

- Implementation in Golang (gnark-crypto)
- bls12381 elliptic curve

Some evaluation results

- Implementation in Golang (gnark-crypto)
- bls12381 elliptic curve
- t3.2xlarge AWS, 32 GB RAM, 8 virtual cores, 2.50GHz CPU

Some evaluation results

- Implementation in Golang (gnark-crypto)
- bls12381 elliptic curve
- t3.2xlarge AWS, 32 GB RAM, 8 virtual cores, 2.50GHz CPU

Some evaluation results

- Implementation in Golang (gnark-crypto)
- bls12381 elliptic curve
- t3.2xlarge AWS, 32 GB RAM, 8 virtual cores, 2.50GHz CPU

Scheme	Signing time (ms)	Partial signature verification time (ms)	Partial signature size (bytes)
Boldyreva-I	0.81	1.12	96
Boldyreva-II	1.20	0.76	160

Source: <https://github.com/sourav1547/adaptive-bls>

Some evaluation results

- Implementation in Golang (gnark-crypto)
- bls12381 elliptic curve
- t3.2xlarge AWS, 32 GB RAM, 8 virtual cores, 2.50GHz CPU

Scheme	Signing time (ms)	Partial signature verification time (ms)	Partial signature size (bytes)
Boldyreva-I	0.81	1.12	96
Boldyreva-II	1.20	0.76	160

Common case aggregation time (for $t=64$) is 7.7 ms for both schemes!

Source: <https://github.com/sourav1547/adaptive-bls>

Summary

Summary

- Background on Bilinear pairing

Summary

- Background on Bilinear pairing
- Boneh-Lynn Shacham (BLS) Signature

Summary

- Background on Bilinear pairing
- Boneh-Lynn Shacham (BLS) Signature
- Threshold Secret Sharing

Summary

- Background on Bilinear pairing
- Boneh-Lynn Shacham (BLS) Signature
- Threshold Secret Sharing
- Design of Threshold BLS Signature

Summary

- Background on Bilinear pairing
- Boneh-Lynn Shacham (BLS) Signature
- Threshold Secret Sharing
- Design of Threshold BLS Signature
- Security of Threshold BLS Signature

Summary

- Background on Bilinear pairing
- Boneh-Lynn Shacham (BLS) Signature
- Threshold Secret Sharing
- Design of Threshold BLS Signature
- Security of Threshold BLS Signature
- Evaluation Results



Summary

- Background on Bilinear pairing
- Boneh-Lynn Shacham (BLS) Signature
- Threshold Secret Sharing
- Design of Threshold BLS Signature
- Security of Threshold BLS Signature
- Evaluation Results



Thank you! (<https://sourav1547.github.io/>)