AIMer Signature Scheme

Principal Submitter : Seongkwang Kim

HSU TING YU
Supervisor: Wakaha Ogata, Fukang Liu

Summary

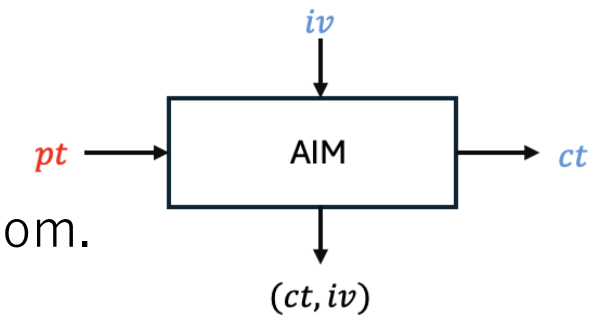
- Explain the basic knowledge of MPC-in-the-head, Fiat-Shamir Transform and BN++ proof system.
- Illustrate the AIMer signature scheme and how to accomplish the algorithm with MPCitH and BN++ proof system.

Overview

- Overview of AIMer Signature Scheme
- Background
  - MPC-in-the-head
  - Fiat-Shamir Transform
  - BN++ Proof System
- AIMer Signature Scheme
  - Key Generation
  - Signing Algorithm
  - Verification Algorithm

Overview of AIMer Signature Scheme

- AIMer Signature Scheme is a post-quantum cryptography(PQC), which is a candidate of NIST competition now.
- Key Generation
  - Key pair: $sk=pt$, $pk=(iv,ct)$
    - A tweak iv and a plaintext pt are sampled uniformly at random.
    - $ct=AIM(iv,pt)$
  - AIM is a “tweakable” one-way function.



- Signing Algorithm
  - The signer prove that it knows the secret pt that satisfied $ct=AIM(iv,pt)$ without revealing the secret to verifier.
- Verification Algorithm
  - The verifier verify the signer knows the secret pt by non-interactive proof.

Background

- MPC-in-the-head
  - Zero-knowledge from Secure Multiparty Computation [IKOS07]
- Fiat-Shamir Transform
- BN++ Proof System
  - Efficient Lifting for Shorter Zero-Knowledge Proofs and Post-Quantum Signatures [KZ22]
  - Amortized Complexity of Information-Theoretically Secure MPC Revisited [CCXY18]

Background

Multi-party Computation → Zero-Knowledge Proof → Signature Scheme

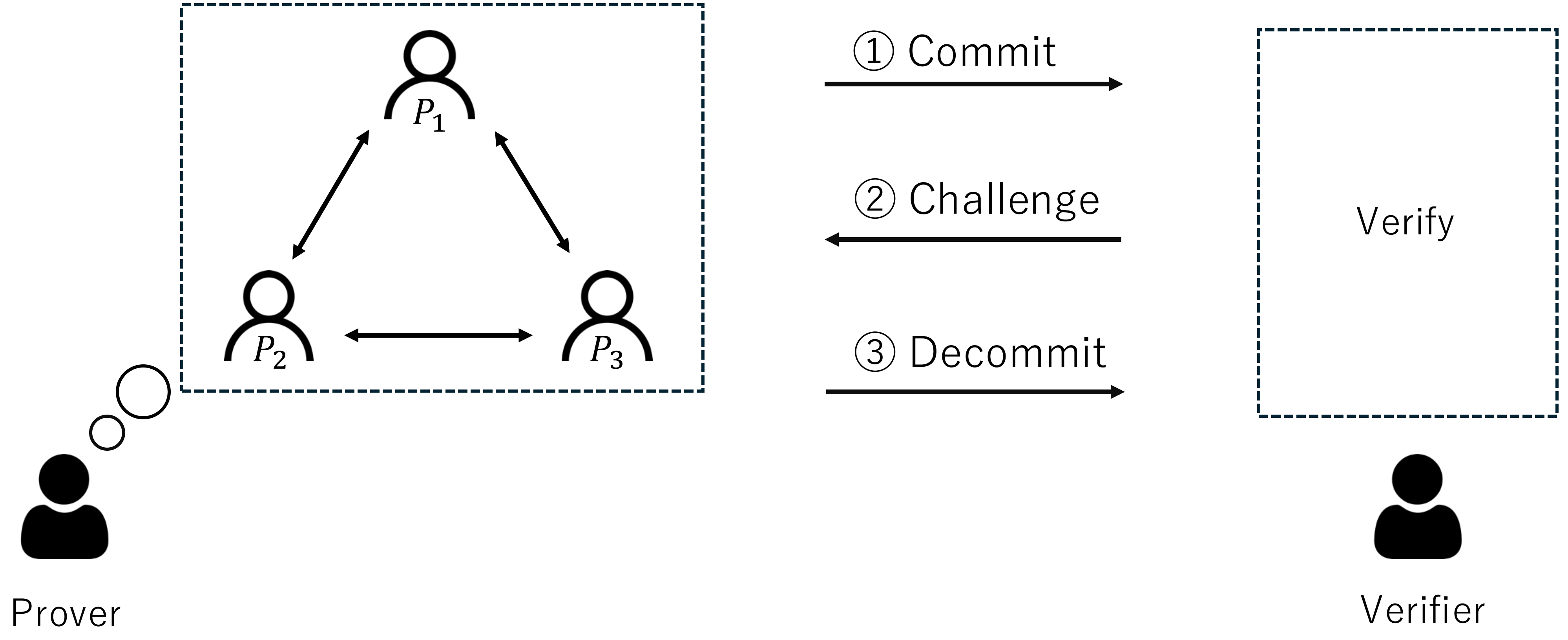
BN++ Proof System

Fiat-Shamir Transform

MPC-in-the-head

MPC-in-the-head (MPCitH) [IKOS07]

- Consider one to construct a zero-knowledge proof(ZKP) from a multi-party computation (MPC) protocol.



Multi-party Computation(MPC)

Definition

P_n: party n

$rand_n$: random tape

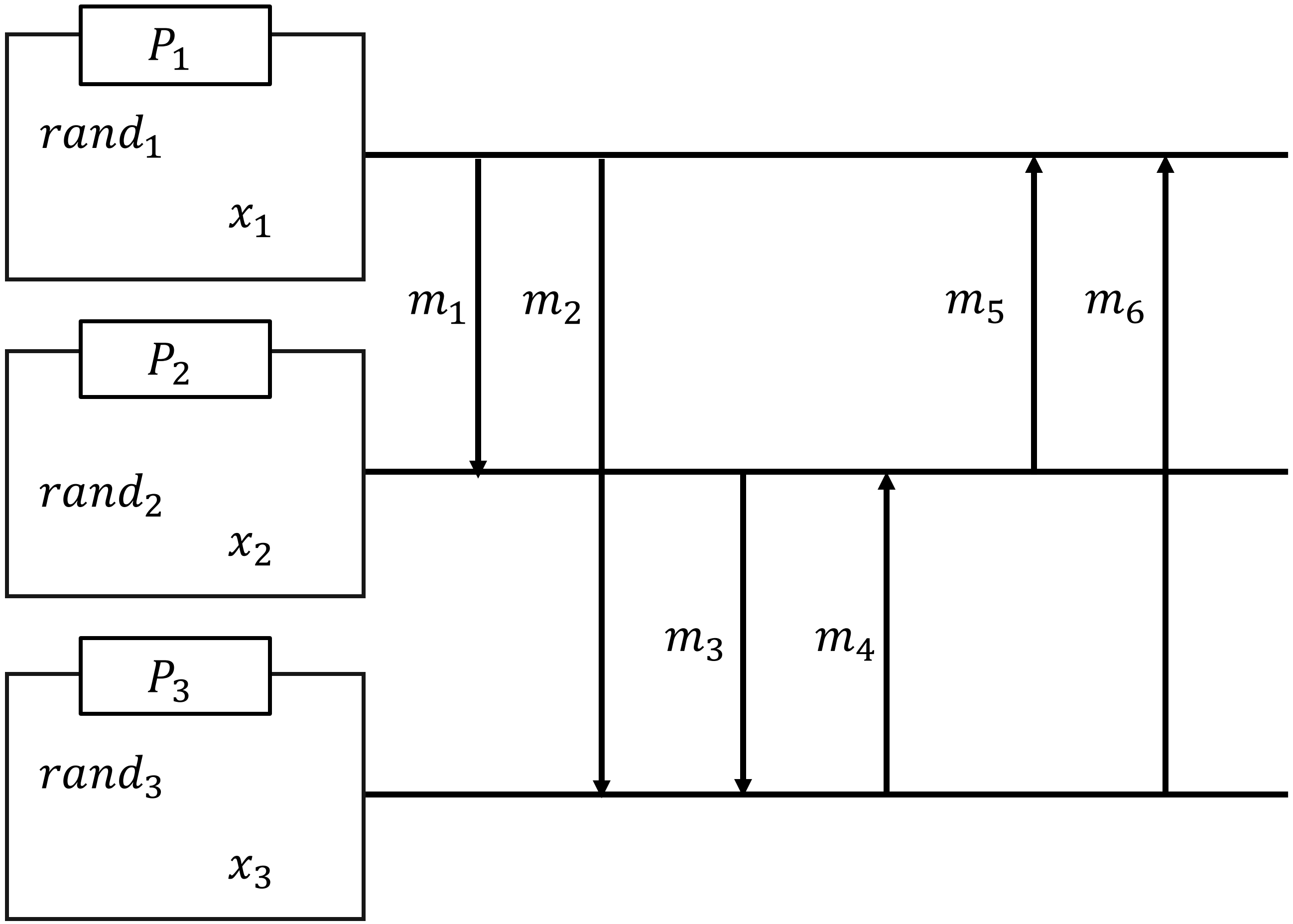
x: secret

x_n: secret input share

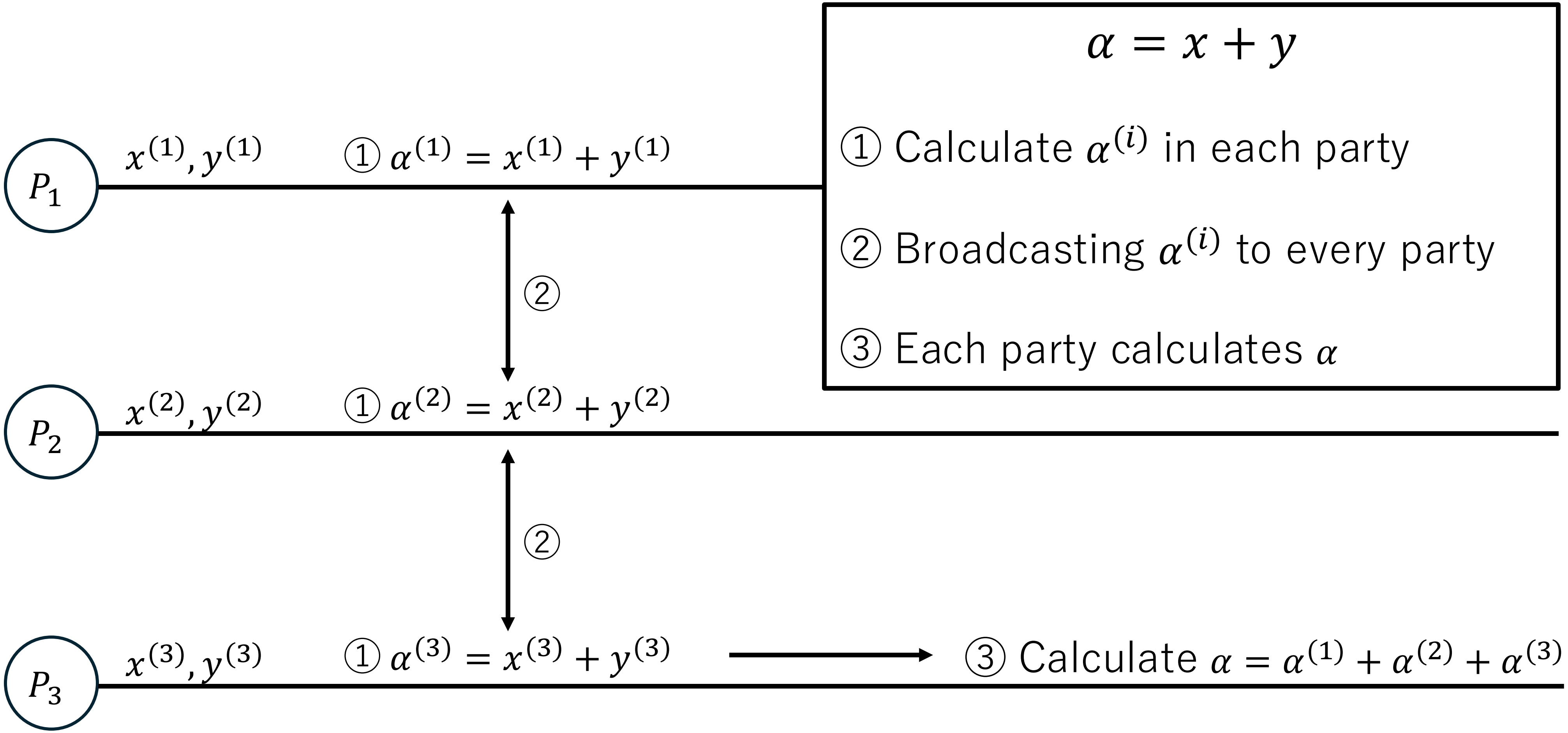
m_n: communicated messages from and to each party

$$x = x_1 + x_2 + x_3$$

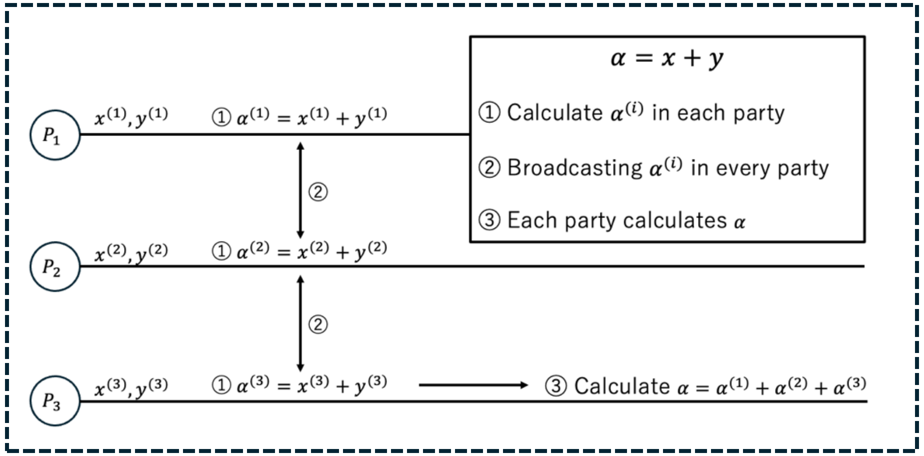
$\Longrightarrow$ Additive share of x



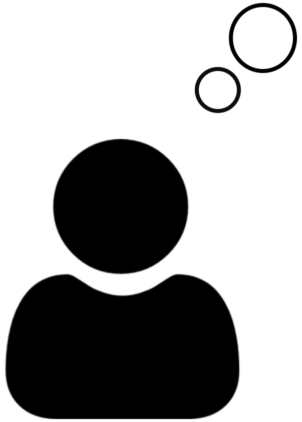
Concrete example of MPC



MPC-in-the-head



Change party into view in MPCitH



Prover

MPC vs. MPCitH

MPC: Operate in actual parties.
MPCitH: virtually done only in prover’s head.

MPC-in-the-Head

Step1. The prover simulates MPC.
Step2. Operate Σ-protocol between the verifier and the prover.

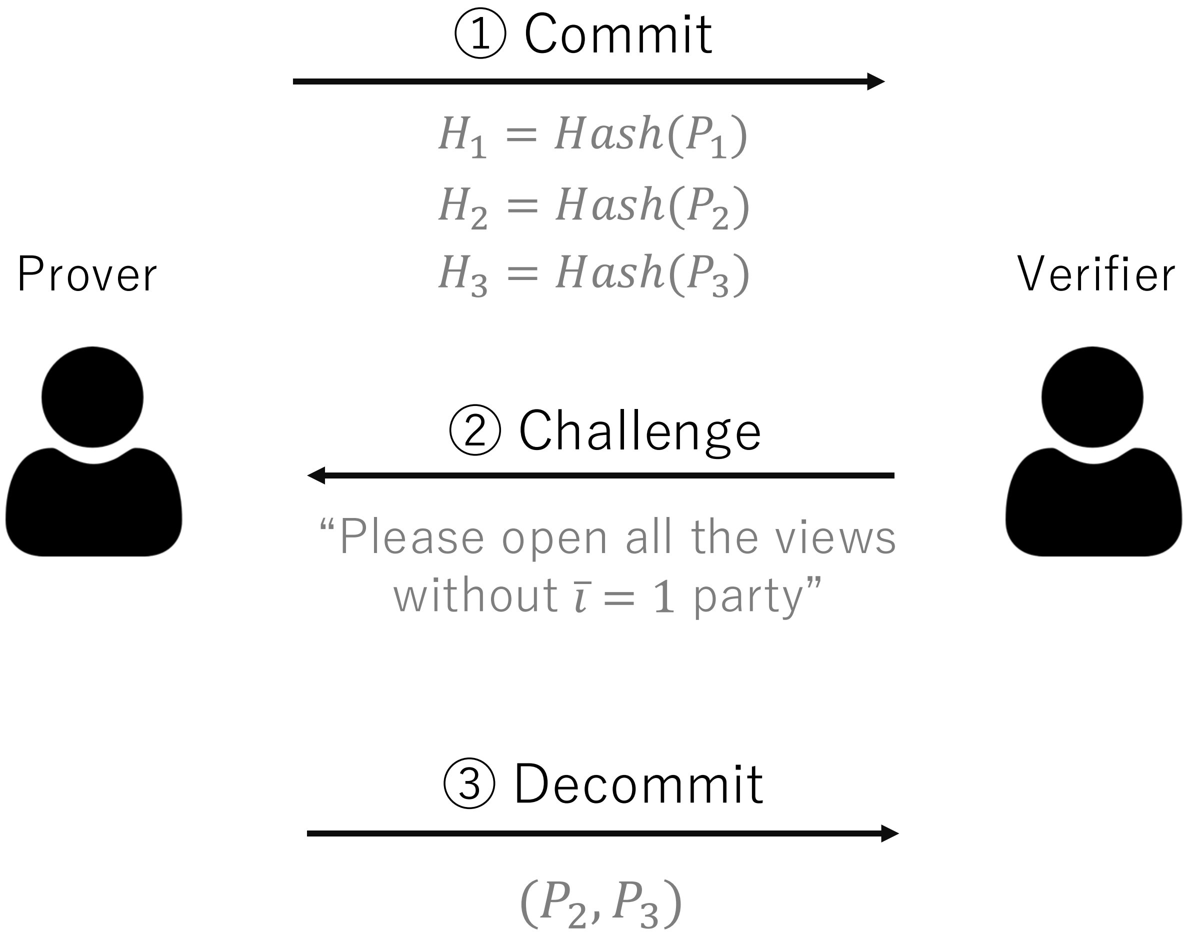
MPC-in-the-head (MPCitH) [IKOS07]

Σ-protocol

① Commit
After committing, the prover cannot change the values.

② Challenge
The verifier will send the requirement for the key of the randomly party.

③ Decommit
The verifier can make sure the prover is not cheating based on the known information.



Fiat-Shamir Transform

Fiat-Shamir Transform

- The technique for taking an interactive proof of knowledge and creating a non-interactive counterpart.

BN++ Proof System

BN++ Proof System

- In the signature scheme, the signer want to show they know the secret that satisfying an equation. $\Longrightarrow z_j = x_j \cdot y_j$
- In order to check, we need two triples to verify, the multiplication triple $(x_j, y_j, z_j = x_j \cdot y_j)_{j=1}^{C}$ and the helping triples $((a_j, b_j)_{j=1}^{C}, c)$, which $b_j = y_j$, $\mathrm{c} = \sum_{j=1}^{C} a_j \cdot b_j$. Additionally, each party holds secret share of the multiplication triples and the helping triples.
- BN++ Protocol Overview:
  - The i-th party locally sets $\alpha_j^{(i)} = \epsilon_j \cdot x_j^{(i)} + a_j^{(i)}$, which ϵ_j is random challenge given by the prover.
  - The parties open the $\alpha_1, \ldots, \alpha_c$ by broadcasting their shares.
  - The i-th party locally sets $v^{(i)} = \sum_{j=1}^{c} \epsilon_j \cdot z_j^{(i)} - \sum_{j=1}^{c} \alpha_j \cdot b_j^{(i)} + c^{(i)}$.
  - The parties open v by broadcasting their shares and output **Accept** if $v = 0$.

Improvement of BN to BN++ [KZ22]

- BN protocol overview
  - The parties locally set $\alpha_j^{(i)} = \epsilon_j \cdot x_j^{(i)} + a_j^{(i)}, \beta_j^{(i)} = \epsilon_j \cdot y_j^{(i)} + b_j^{(i)}$, which ϵ_j is random challenge given by the prover.
  - The parties open α and β by broadcasting their shares.
  - The parties locally set $v^{(i)} = \epsilon \cdot z^{(i)} - c^{(i)} + \alpha \cdot b^{(i)} + \beta \cdot a^{(i)} - \alpha \cdot \beta$.
  - The parties open v by broadcasting their shares and output **Accept** if $v = 0$.
- Goal
  - To optimize proof size from $5C\ \log_2(|\mathbb{F}|)$ to $(2C+1)\ \log_2(|\mathbb{F}|)$
    - BN: $\left(\Delta c_{e,l}, \Delta z_{e,l}, \alpha_{e,l}^{(\bar{\iota}_e)}, \beta_{e,l}^{(\bar{\iota}_e)}, v_{e,l}^{(\bar{\iota}_e)}\right)$
    - BN++: $\left(\Delta c_{e,l}, \Delta z_{e,l}, \alpha_{e,l}^{(\bar{\iota}_e)}, \beta_{e,l}^{(\bar{\iota}_e)}, v_{e,l}^{(\bar{\iota}_e)}\right) \rightarrow \left(\Delta z_{e,l}, \alpha_{e,l}^{(\bar{\iota})}\right)$ and one Δc_e per repetition

BN++ Proof System

- Efficient Lifting for Shorter Zero-Knowledge Proofs and Post-Quantum Signatures [KZ22]
  - Improvement of BN to BN++
  - Handling Small Fields Efficiently
  - Reverse Multiplication Friendly Embedding(RMFE)

- Amortized Complexity of Information-Theoretically Secure MPC Revisited [CCXY18]
  - Concrete Example of RMFE

Handling Small Fields Efficiently [KZ22]

- Problem
  - The BN++ protocol performs well for circuits defined over large fields, for example, AIMer implements in $\mathbb{F}_{2^{128}}$, which is no need for considering this problem, but we have to check several multiplications in small fields, such as $\mathbb{F}_{2^8}$, to maintain the soundness.
  - The soundness possibility is $1/|\mathbb{F}|$, which indicates the probability the verifier is cheated.

Solution for small fields inefficiency [KZ22]

- Simple Lifting
  - This method provides a small improvement to the next smallest field for example, $\mathbb{F}_{2^8} \rightarrow \mathbb{F}_{2^{16}}$.
- Multiple Checks Per Repetition
  - Instead of simple lifting $\mathbb{F}_2 \rightarrow \mathbb{F}_{2^M}$($M$: Checking protocol times per repetition, but checking M times on $\mathbb{F}_2$.
- Lifting with RMFEs
  - Reverse Multiplication Friendly Embedding(RMFE) allows us to encode multiple bits into a field extension with better rate.
- Use a circuit defined in $\mathbb{F}_{2^M}$

Reverse Multiplication Friendly Embedding(RMFE) [KZ22]

- RMFE is a pair of linear maps (ϕ, ψ) which allows to perform coordinate-wise multiplication over small fields by operating overextension fields.

$$(a_1, a_2, \ldots, a_k) * (b_1, b_2, \ldots, b_k) = (a_1 \cdot b_1, a_2 \cdot b_2, \ldots, a_k \cdot b_k)$$

$\in \mathbb{F}_2^k$ $\quad$ $\in \mathbb{F}_2^k$

Encoded by ϕ $\quad$ Encoded by ϕ $\quad$ Decoded by ϕ

$$\phi(a) \cdot \phi(b) = \phi(a) \cdot \phi(b)$$

$\in \mathbb{F}_{2^m}$ $\quad$ $\in \mathbb{F}_{2^m}$

- For example, $(3,5)_2$, with rate is 5/3=1.6

$$(x_1, x_2, x_3) * (y_1, y_2, y_3) = (x_1 \cdot y_1, x_2 \cdot y_2, x_3 \cdot y_3)$$

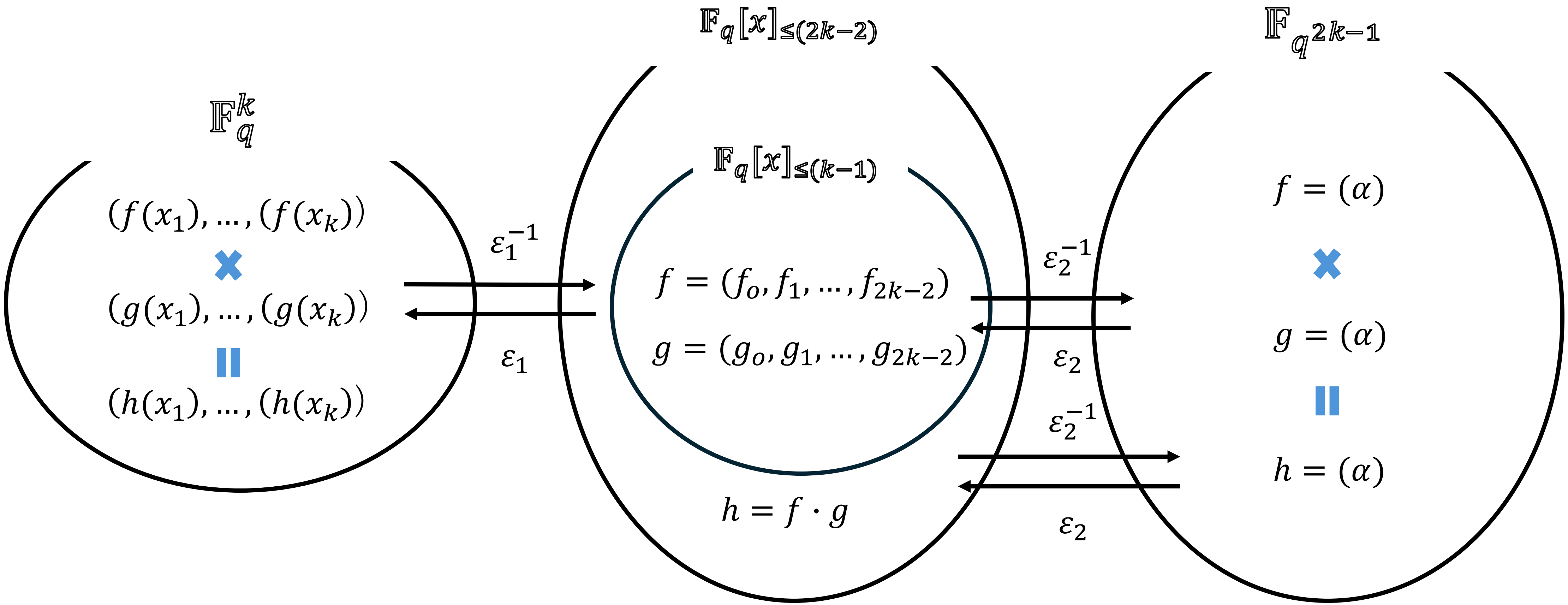
$\in \mathbb{F}_2^3$ $\quad$ $\in \mathbb{F}_2^3$

Encoded by ϕ $\quad$ Encoded by ϕ $\quad$ Decoded by ϕ

$$\phi(x) \cdot \phi(y) = \phi(x) \cdot \phi(y)$$

$\in \mathbb{F}_{2^5}$ $\quad$ $\in \mathbb{F}_{2^5}$ $\quad$ $\in \mathbb{F}_{2^5}$

Concrete Example of RMFE [CCXY18]



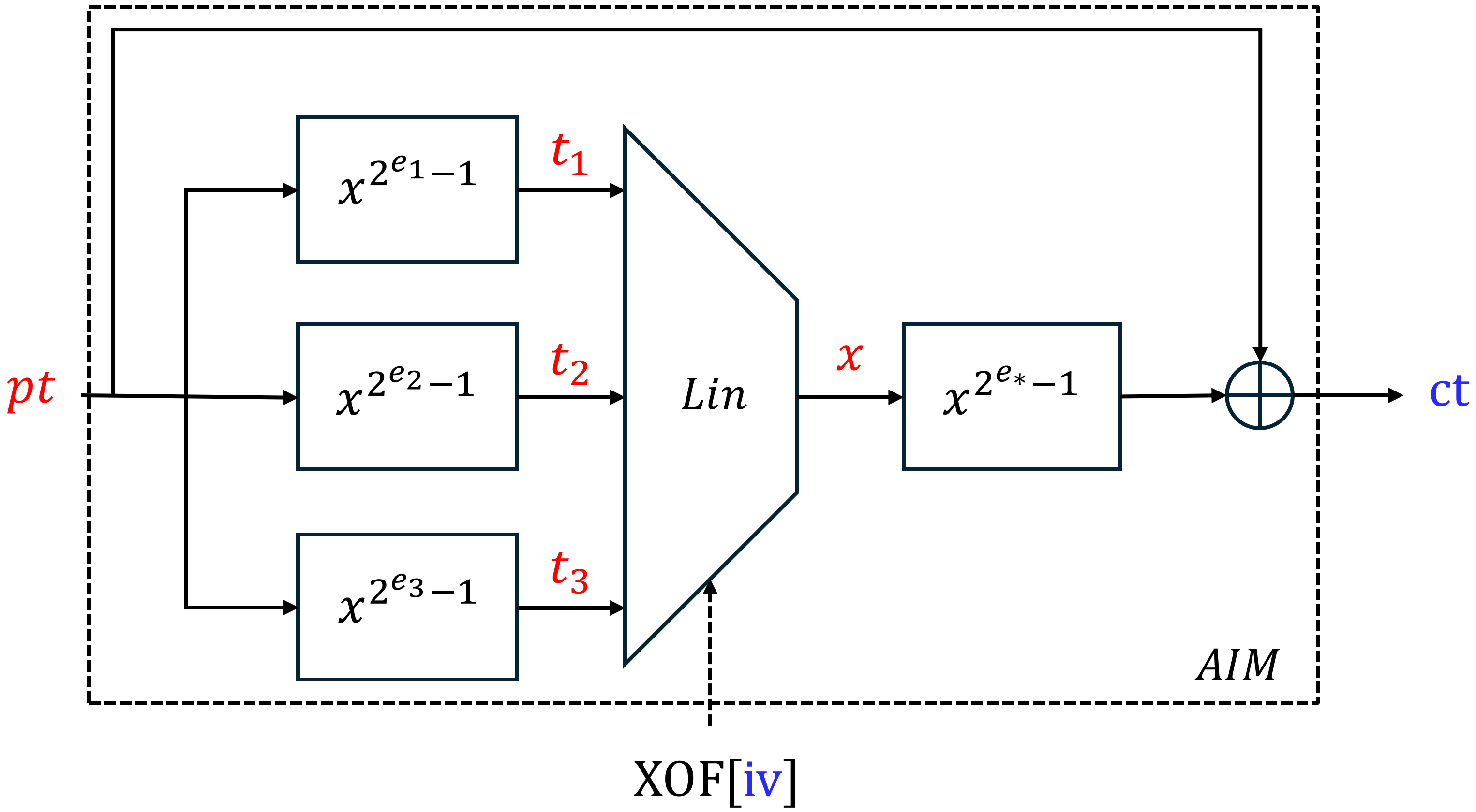
AIMer Signature Scheme

AIMer Signature Scheme

- Key Generation
- Signing Algorithm
- Verification Algorithm

Key Generation

Key Generation

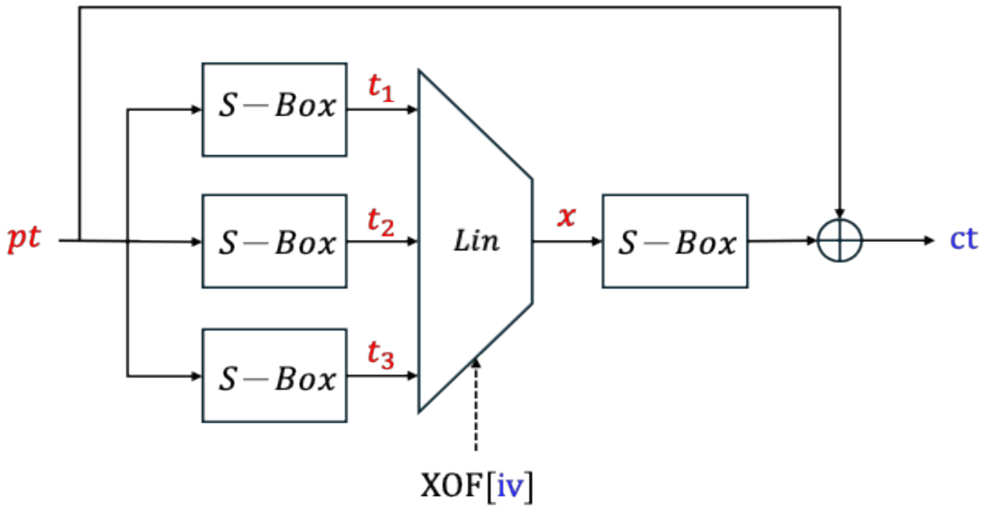


Public Key: (iv, ct)
Secret Key: pt

$ct = AIM(iv, pt)$

All calculations for AIM are over a finite field $\mathbb{F}$ where $\mathbb{F} \in \{\mathbb{F}_{2^{128}}, \mathbb{F}_{2^{192}}, \mathbb{F}_{2^{256}}\}$.

Feature of the AIM function

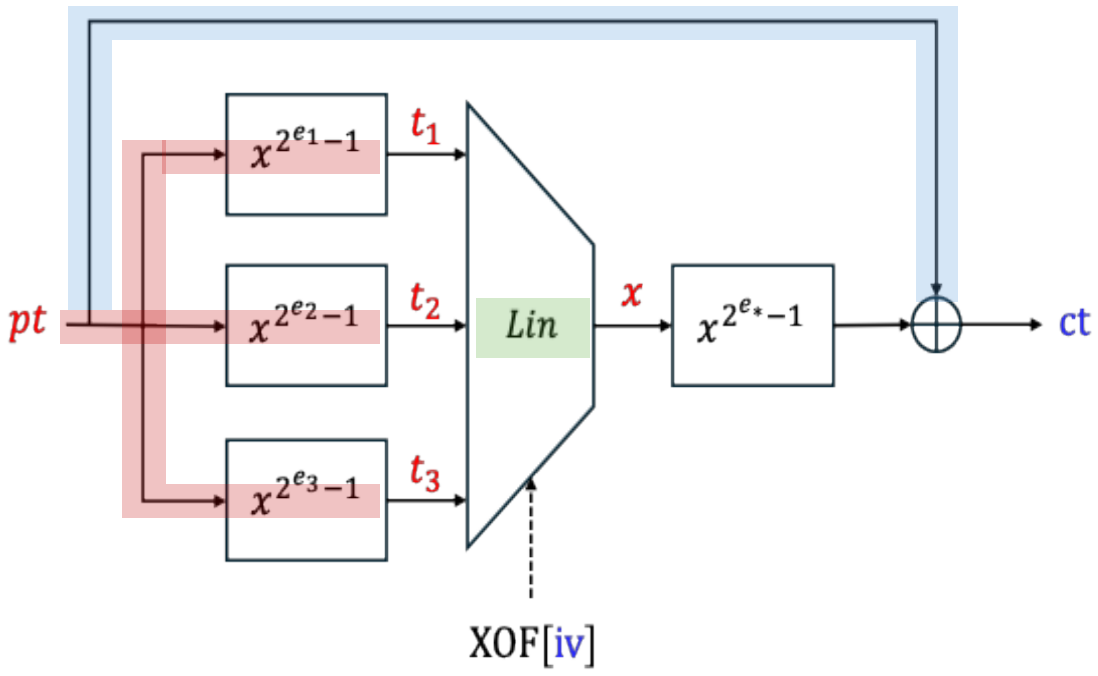


- S-box
  - S-boxes are exponentiation by Mersenne numbers over a large field.

$$Mer[e](x) = x^{2^e-1}$$

- Linear Components
  - Random binary matrix $A_{iv} = [A_{iv,1} | \ldots | A_{iv,l}] \in (\mathbb{F}_2^{n\times n})^l$
  - Random constant vector b_{iv}.
  - The matrix A_{iv} and the vector b_{iv} are generated by an extendable-output function (XOF) with the initial vector iv.

Feature of the AIM function



Take **j = 3** as an example,

① $pt^{2^{e_j}-1} = t_j \Leftrightarrow t_j \cdot pt = pt^{2^{e_j}}, j = 1,2,3$

② $x^{2^{e_*}-1} = pt + ct \Leftrightarrow x \cdot pt = x \cdot ct + x^{2^{e_*}}$

③ $x = Lin_{iv}\,(t_1, t_2, t_3)$

Blue: Known by the verifier,

hence, not share in MPC, which can be seen as constant.

Red: Calculate as the secret share of MPC.

Signing Algorithm

Signing Algorithm

MPCitH

(x_1, pt, z_1)
(x_2, pt, z_2)
(x_3, pt, z_3)

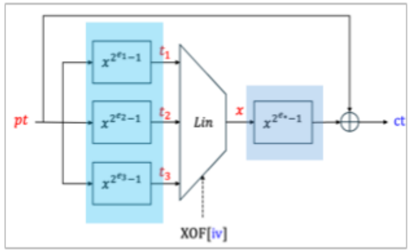
Change the value into share by **Random Seed**

1. Generate every share by random: $pt_k^{(i)} \leftarrow \text{Sample}(tape_k^{(i)})$
2. Adjust first share: $\Delta\, pt_k^{(i)} \leftarrow pt - \sum_i pt_k^{(i)}$, $pt_k^{(1)} \leftarrow pt_k^{(1)} + \Delta pt_k$.

$(x_1^{(i)}, pt^{(i)}, z_1^{(i)})$
$(x_2^{(i)}, pt^{(i)}, z_2^{(i)})$
$(x_3^{(i)}, pt^{(i)}, z_3^{(i)})$

AIM Set $x_{k,j}^{(i)}, z_{k,j}^{(i)}$

	$x_{k,j}^{(i)}$	$z_{k,j}^{(i)}$
$j \leq l$	$t_{k,j}^{(i)}$	$((pt)_k^{(i)})^{2^{e_j}}$
$j = l+1$	$Lin_{iv}(t_1, t_2, t_3)$	$ct \cdot x_{k,j}^{(i)} + (x_{k,j}^{(i)})^{2^{e_j}}$



BN++

Multiplication Triple $(x_{k,j}^{(i)}, pt_k^{(i)}, z_{k,j}^{(i)})$ → Generate → Helping Triple $(a_k^{(i)}, b_k^{(i)}, c_k^{(i)})$ → Locally calculating $\alpha_j^{(i)}, v^{(i)}$ $(\alpha_j^{(i)}, v^{(i)})$

Using a multiplication triple to generate a helping triple that results in $v = 0$

According to the checking protocol,
$\alpha_j^{(i)} = \epsilon_j \cdot x_j^{(i)} + a_j^{(i)}$
$v^{(i)} = \sum_{j=1}^{c} \epsilon_j \cdot z_j^{(i)} - \sum_{j=1}^{c} \alpha_j \cdot b_j^{(i)} + c^{(i)}$

Hash

Generate the Challenge: $\bar{\iota}_k$ party

① Hash
$h_1 = H(\alpha^{(1)}, v^{(1)}, \alpha^{(2)}, v^{(2)}, \dots)$

Connected $\alpha^{(i)}, v^{(i)}$ from every party and hashed as an element of output.

③ Signature
$\sigma = (\dots, h_1, \alpha_k^{(\bar{\iota}_k)}, (x_{k,j}^{(i)}, pt_k^{(i)}, z_{k,j}^{(i)})_{i=N\setminus\{\bar{\iota}_k\}}, (a_k^{(i)}, b_k^{(i)}, c_k^{(i)}) \dots)$

$\bar{\iota}_k$ is the only party verifier will not get the multiplication triple and the helping triple, instead of that, the signature includes α from $\bar{\iota}_k$ party.

Verifier

Signing Algorithm – MPCitH

Definition

$$\forall k: \sum_{i=1}^{N} x_{k,j}^{(i)} = x_j$$

$$\forall k: \sum_{i=1}^{N} pt_k^{(i)} = pt$$

$$\forall k: \sum_{i=1}^{N} z_{k,j}^{(i)} = z_j$$

(x_1, pt, z_1)

(x_2, pt, z_2)

(x_3, pt, z_3)

Change the value into share by **Random Seed**

1. Generate every share by random:

$$pt_k^{(i)} \leftarrow \text{Sample}(tape_k^{(i)})$$

2. Adjust first share:

$$\Delta\, pt_k^{(i)} \leftarrow pt - \textstyle\sum_i pt_k^{(i)}\ ,\ \ pt_k^{(1)} \leftarrow pt_k^{(1)} + \Delta pt_k.$$

$k = 1,2,3,\ldots,\tau$

$i = 1,2,3,\ldots,N$

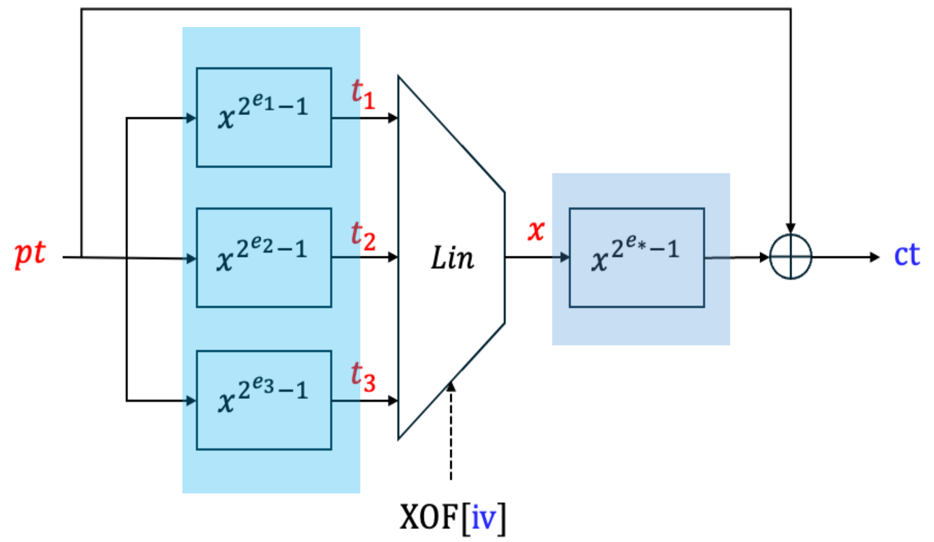
$(x_{k,1}^{(i)}, pt_k^{(i)}, z_{k,1}^{(i)})$

$(x_{k,2}^{(i)}, pt_k^{(i)}, z_{k,2}^{(i)})$

$(x_{k,3}^{(i)}, pt_k^{(i)}, z_{k,3}^{(i)})$

The k –th iteration

Signing Algorithm – AIM



Set $x_{k,j}^{(i)}, z_{k,j}^{(i)}$

	$x_{k,j}^{(i)}$	$z_{k,j}^{(i)}$
$j \leq l$	$t_{k,j}^{(i)}$	$((pt)_k^{(i)})^{2^{e_j}}$
$j = l + 1$	$Lin_{iv}(t_1, t_2, t_3)$	$ct \cdot x_{k,j}^{(i)} + (x_{k,j}^{(i)})^{2^{e_j}}$

Feature of AIM

$x_j \cdot y_j = z_j$

$t_j \cdot pt = pt^{2^{e_j}}$

$x \cdot pt = x \cdot ct + x^{2^{e_*}}$

Signing Algorithm – BN++

BN++ Protocol Overview

- Goal: verify $x_i \cdot y_i = z_i$ for $1 \le i \le C$, for secret triples (x_i, y_i, z_i)
- Generate helping triple (a_i, b_i, c) such that $b_i = y_i$, a_i is randomly chosen from $\mathbb{F}$, and $\mathrm{c} = \sum_{j=1}^{C} a_j \cdot\ b_j$ for $1 \le i \le C$.
- The i-th party locally sets $\alpha_j^{(i)} = \epsilon_j \cdot x_j^{(i)} + a_j^{(i)}$, ϵ_j is a random challenge from the verifier.
- The parties open the $\alpha_1, \dots, \alpha_c$ by broadcasting their shares.
- The i-th party locally sets $v^{(i)} = \sum_{j=1}^{c} \epsilon_j \cdot\ z_j^{(i)} - \sum_{j=1}^{c} \alpha_j \cdot\ b_j^{(i)} + c^{(i)}$.
- The parties open v by broadcasting their shares and output **Accept** if $v = 0$.

Multiplication Triple $(x_{k,j}^{(i)}, pt_k^{(i)}, z_{k,j}^{(i)})$

Generate

Helping Triple $(a_k^{(i)}, b_k^{(i)}, c_k^{(i)})$

Using a multiplication triple to generate a helping triple that results in $v = 0$

Locally calculating $(\alpha_j^{(i)}, v^{(i)})$

Conclusion

If the triples satisfy that $x_i \cdot y_i = z_i$ and $\mathrm{c} = \sum_{i=1}^{C} a_i \cdot\ b_i$, then v must be 0. Otherwise, $v = 0$ with a negligible probability $1/|\mathbb{F}|$, where $|\mathbb{F}|$ denotes the field size.

Signing Algorithm – Hash

MPCitH

<u>Σ-protocol</u>

① Commit
② Challenge
③ Decommit

② Generate the Challenge: $\bar{\imath}_k$ party

① Hash

$$h_1 = H\left(\alpha^{(1)}, v^{(1)}, \alpha^{(2)}, v^{(2)}, \ldots\right)$$

Connected $\alpha^{(i)}, v^{(i)}$ from every party and hashed as an element of output.

③ Signature

$$\sigma = \Big(\ldots, h_1, \alpha_k^{(\bar{\imath}_k)},$$
$$\left(x_{k,j}^{(i)}, pt_k^{(i)}, z_{k,j}^{(i)}\right)_{i=N\setminus\{\bar{\imath}_k\}},$$
$$\left(a_k^{(i)}, b_k^{(i)}, c_k^{(i)}\right)_{i=N\setminus\{\bar{\imath}_k\}} \ldots\Big)$$

$\bar{\imath}_k$ is the only party verifier will not get the multiplication triple and the helping triple, instead of that, the signature need to includes α from $\bar{\imath}_k$ party.

Verifier

Verification Algorithm

Verification Algorithm

Verifier

Signature
$\sigma = (\ldots, h_1, \alpha_k^{(\bar{\iota}_k)},$
$\left(x_{k,j}^{(i)}, pt_k^{(i)}, z_{k,j}^{(i)}\right)_{i=N\setminus\{\bar{\iota}_k\}}, (a_k^{(i)}, b_k^{(i)}, c_k^{(i)}) \ldots)$

Calculate without $\bar{\boldsymbol{\iota}}_{\boldsymbol{k}}$ party

Set $x_{k,j}^{(i)}, z_{k,j}^{(i)}$
$j \le l \Rightarrow x_{k,j}^{(i)} = t_{k,j}^{(i)}, z_{k,j}^{(i)} = ((pt)_k^{(i)})^{2^{e_j}}$
$j = l+1 \Rightarrow x_{k,j}^{(i)} = Lin_{iv}(t_1, t_2, t_3), z_{k,j}^{(i)} = (x_{k,j}^{(i)})^{2^{e_j}} + ct \cdot x_{k,j}^{(i)}$

Multiplication Triple
$(x_{k,j}^{(i)}, pt_k^{(i)}, z_{k,j}^{(i)})$

Generate

Helping Triple
$(a_k^{(i)}, b_k^{(i)}, c_k^{(i)})$

Locally calculating $\alpha_j^{(i)}$, $v^{(i)}$
$(\alpha_j^{(i)}, v^{(i)})$

Compute $v_k^{(\bar{\iota}_k)}$
$v_k^{(\bar{\iota}_k)} = 0 - \sum_{i \ne \bar{\iota}_k} v_k^{(i)}$

Hash
$\acute{h}_1 = Hash(\alpha^{(1)}, v^{(1)}, \alpha^{(2)}, v^{(2)}, \alpha^{(\bar{\iota}_k)}. v^{(\bar{\iota}_k)} \ldots)$

Output
Accept if $h_1 = \acute{h}_1$, otherwise **Reject**

Verification Algorithm

Verifier

Signature

$\sigma = (\ldots, h_1, \alpha_k^{(\bar{\imath}_k)},$
$\left(x_{k,j}^{(i)}, pt_k^{(i)}, z_{k,j}^{(i)}\right)_{i=N\setminus\{\bar{\imath}_k\}},$
$\left(a_k^{(i)}, b_k^{(i)}, c_k^{(i)}\right)_{i=N\setminus\{\bar{\imath}_k\}} \ldots)$

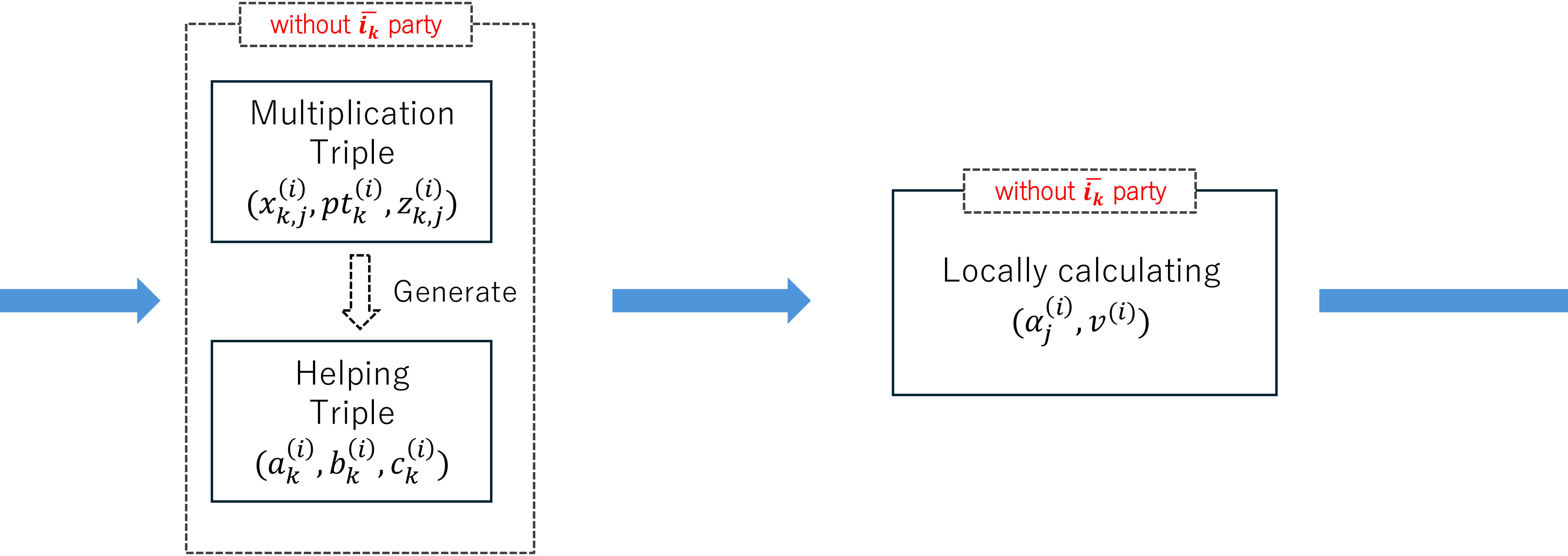
without $\bar{\boldsymbol{\imath}}_{\boldsymbol{k}}$ party

Set $x_{k,j}^{(i)}, z_{k,j}^{(i)}$

$j \leq l \Rightarrow x_{k,j}^{(i)} = t_{k,j}^{(i)}, \; z_{k,j}^{(i)} = ((pt)_k^{(i)})^{2^{e_j}}$

$j = l + 1 \Rightarrow x_{k,j}^{(i)} = Lin_{iv}(t_1, t_2, t_3), \; z_{k,j}^{(i)} = (x_{k,j}^{(i)})^{2^{e_j}} + ct \cdot x_{k,j}^{(i)}$

Verification Algorithm - BN++



Verification Algorithm - Verify

Compute $v_k^{(\bar{i_k})}$

$$v_k^{(\bar{i_k})} = 0 - \sum_{i \neq \bar{\imath}_k} v_k^{(i)}$$

Hash

$$\acute{h_1} = Hash\left(\alpha^{(1)}, v^{(1)}, \alpha^{(2)}, v^{(2)}, \alpha^{\overline{(i_k)}}. v^{\overline{(i_k)}} \ldots\right)$$

We know $\alpha_k^{(\bar{\imath}_k)}$ from the signature and calculate $v_k^{(\bar{i_k})}$ by assuming $\boldsymbol{v = 0}$, which indicates that the equation is correct.

Output

Accept if $h_1 = \acute{h_1}$, otherwise **Reject**

The End

Presenter: HSU TING YU, National Cheng Kung University

Date: 09/2024 at Ogata Lab, Tokyo Tech

Reference

[1] Kim, S., Cho, J., Cho, M., Ha, J., Kwon, J., Lee, B., Lee, J., Lee, J., Lee, S., Moon, D., & Yoon, H. (n.d.). The AIMer Signature Scheme Submission to the NIST PQC project Version 1.0. Retrieved September 17, 2024, from https://aimer-signature.org/docs/AIMer-NIST-Document.pdf

[2] Yuval Ishai, Eyal Kushilevitz, Ostrovsky, R., & Sahai, A. (2007). Zero-knowledge from secure multiparty computation. Symposium on the Theory of Computing. https://doi.org/10.1145/1250790.1250794

[3] Kales, D., & Zaverucha, G. (2022). Efficient Lifting for Shorter Zero-Knowledge Proofs and Post-Quantum Signatures. Cryptology EPrint Archive. https://eprint.iacr.org/2022/588

[4] Cascudo, I., Cramer, R., Xing, C., & Yuan, C. (2018). Amortized Complexity of Information-Theoretically Secure MPC Revisited. Cryptology EPrint Archive. https://eprint.iacr.org/2018/429