AIMer Standard Format

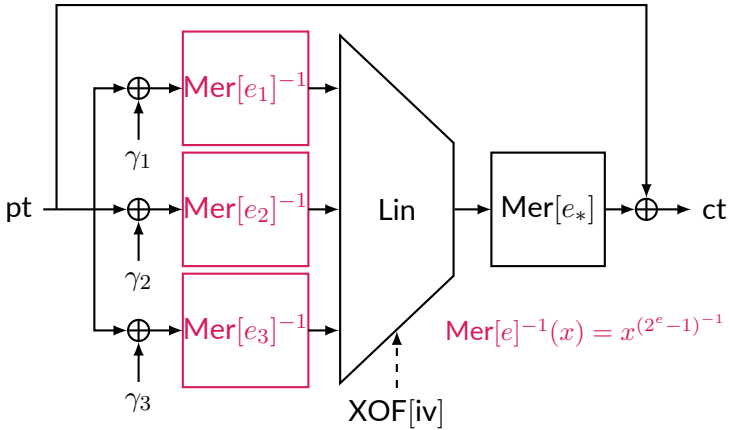
KpqC Conference (2025.11)

Seongkwang Kim

Samsung SDS

Recap of AIMer

AIM2


pt
γ_1
γ_2
γ_3
$\mathrm{Mer}[e_1]^{-1}$
$\mathrm{Mer}[e_2]^{-1}$
$\mathrm{Mer}[e_3]^{-1}$
Lin
$\mathrm{Mer}[e_*]$
ct
XOF[iv]
$\mathrm{Mer}[e]^{-1}(x) = x^{(2^e-1)^{-1}}$

AIMer KeyGen

1. Sample $\mathsf{pt}, \mathsf{iv} \leftarrow_{\$} \{0,1\}^{\lambda}$
2. Compute $\mathsf{AIM2}(\mathsf{pt}, \mathsf{iv}) = \mathsf{ct}$
3. Set $\mathsf{sk} = (\mathsf{pt}, \mathsf{iv}, \mathsf{ct})$, $\mathsf{pk} = (\mathsf{iv}, \mathsf{ct})$

Structure of AIMer Sign

- 5-round Challenge-Response structure
- Challenges are generated using Fiat-Shamir transform
- Phase 1, 3, 5 are repeated τ times, and each repetition computes views of N parties
- In Phase 2, 4, a single digest is generated by hashing all τ responses, and is expanded to challenges of specific length

AIMer Sign

0. Instantiate AIM2 (generating linear layer)
1. Generate views of each parties
    - Generate each party's seed by using GGM tree
    - Random views are genereted from PRG fed by the seeds
    - Compute the corrections between the random views and the real values
    - Commit to the seeds

AIMer Sign

0. Instantiate AIM2 (generating linear layer)
1. Generate views of each parties
2. Generate the first challenge by hashing the commitments and the corrections

AIMer Sign

0. Instantiate AIM2 (generating linear layer)
1. Generate views of each parties
2. Generate the first challenge by hashing the commitments and the corrections
3. Check whether the views correctly check multiplications
    - Get output shares of multiplication check protocol

AIMer Sign

0. Instantiate AIM2 (generating linear layer)
1. Generate views of each parties
2. Generate the first challenge by hashing the commitments and the corrections
3. Check whether the views correctly check multiplications
4. Generate the second challenge by hashing output share

AIMer Sign

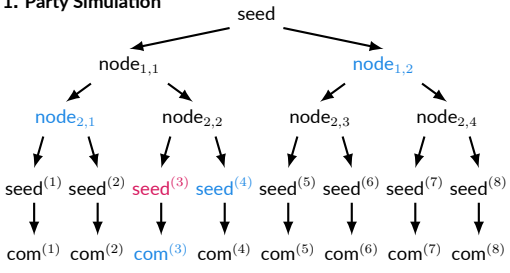
0. Instantiate AIM2 (generating linear layer)
1. Generate views of each parties
2. Generate the first challenge by hashing the commitments and the corrections
3. Check whether the views correctly check multiplications
4. Generate the second challenge by hashing output share
5. Reveal all seeds except the challenge seed (GGM copath)

AIMer Sign

0. Instantiate AIM2 (generating linear layer)
1. Generate views of each parties
2. Generate the first challenge by hashing the commitments and the corrections
3. Check whether the views correctly check multiplications
4. Generate the second challenge by hashing output share
5. Reveal all seeds except the challenge seed (GGM copath)
6. Signature is (GGM copath, commitment to the hidden seeds, corrections, challenge hashes, mult. check masking value)

AIMer Sign

1. Party Simulation



2. Multiplication triple generation

$\mathrm{PRG}(\mathrm{seed}^{(1)}) =$ Witness$^{(1)}$ | Mult. triple$^{(1)}$

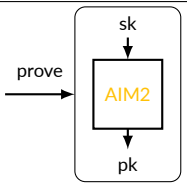
$\vdots$

$\mathrm{PRG}(\mathrm{seed}^{(N)}) =$ Witness$^{(N)}$ | Mult. triple$^{(N)}$

$+\Delta$Witness | $+\Delta$Mult. triple

3. Proof

MultCheck($, ,) = Output shares



4. Party Opening

Choose $i = H(\ )$

Structure of AIMer Verification

- Recompute Phase 1-4 and check whether the second challenge is same
- Commitments to the hidden seeds are in the signature

AIMer Verification

- Instantiate AIM2
- Recompute Phase 1&2
  - Reconstruct opened seeds by using GGM copath
  - Reconstruct views of the opened parties using opened seeds
  - Recomputed opened commitments, and recompute the first challenge hash

AIMer Verification

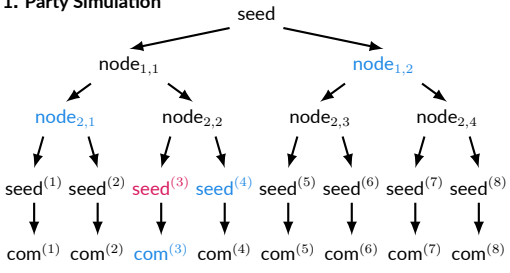
- Instantiate AIM2
- Recompute Phase 1&2
  - Reconstruct opened seeds by using GGM copath
  - Reconstruct views of the opened parties using opened seeds
  - Recomputed opened commitments, and recompute the first challenge hash
- Recompute Phase 3&4
  - Recompute multiplication check protocol for opened parties and get output share of hidden party
  - Recompute the second challenge

AIMer Verification

- Instantiate AIM2
- Recompute Phase 1&2
  - Reconstruct opened seeds by using GGM copath
  - Reconstruct views of the opened parties using opened seeds
  - Recomputed opened commitments, and recompute the first challenge hash
- Recompute Phase 3&4
  - Recompute multiplication check protocol for opened parties and get output share of hidden party
  - Recompute the second challenge
- Accept if the second challenges are same, reject otherwise

AIMer Verification

1. Party Simulation



2. Multiplication triple generation

$\text{PRG}(\text{seed}^{(1)}) =$ Witness$^{(1)}$ Mult. triple$^{(1)}$

$\vdots$

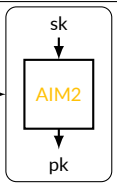
$\text{PRG}(\text{seed}^{(N)}) =$ Witness$^{(N)}$ Mult. triple$^{(N)}$

$+\Delta$Witness $+\Delta$Mult. triple

3. Proof

MultCheck($, ,) = Output shares

prove



4. Party Opening

Choose $i = H(\ )$

Notes on Standard Document

Function Call Map: KeyGen

- AIMer_keygen - AIMer_keygen_internal
  - AIM2
    - AIM2_GenerateLinear

Function Call Map: Sign/Verify

- AIMer_sign - AIMer_sign_internal
  - Hash: $H_0, \ldots, H_5$, ExpandH1, ExpandH2
  - AIM2_SboxOutputs
  - AIM2_GenerateLinear
  - ExpandTree / ReconstructTree
    - Hash function H_4
  - AIM2_MPC
  - RevealAllBut

Some Added Details

Some Added Details

- Explicitly written exponents and constants

Some Added Details

- Explicitly written exponents and constants
- Output size and format of hash functions

Some Added Details

- Explicitly written exponents and constants
- Output size and format of hash functions
- How to generate random numbers: DRBG

Some Added Details

- Explicitly written exponents and constants
- Output size and format of hash functions
- How to generate random numbers: DRBG
- Data conversion (bitstring $\leftrightarrow$ integer)

Some Added Details

- Explicitly written exponents and constants
- Output size and format of hash functions
- How to generate random numbers: DRBG
- Data conversion (bitstring $\leftrightarrow$ integer)
- Finite field, matrix-vector multiplication pseudocode

Some Added Details

- Explicitly written exponents and constants
- Output size and format of hash functions
- How to generate random numbers: DRBG
- Data conversion (bitstring $\leftrightarrow$ integer)
- Finite field, matrix-vector multiplication pseudocode
- AIM2_SboxOutputs function

Some Added Details

- Explicitly written exponents and constants
- Output size and format of hash functions
- How to generate random numbers: DRBG
- Data conversion (bitstring $\leftrightarrow$ integer)
- Finite field, matrix-vector multiplication pseudocode
- AIM2_SboxOutputs function
- Multiplication check protocol details

Some Notes

- Standard document is mainly written using bitstring, which may be quite different in word-based implementation
- We don’t use “byte” in the document. When using bytes in the real implementation, please be careful on byte order

Some Notes

- Standard document is mainly written using bitstring, which may be quite different in word-based implementation
- We don't use "byte" in the document. When using bytes in the real implementation, please be careful on byte order
- As in Specification v2.1, we only use SHAKE rather than SHA2/3 for hash functions

Some Notes

- Standard document is mainly written using bitstring, which may be quite different in word-based implementation
- We don't use "byte" in the document. When using bytes in the real implementation, please be careful on byte order
- As in Specification v2.1, we only use SHAKE rather than SHA2/3 for hash functions
- Unlike ML-DSA/SLH-DSA, we don't have pre-hash variant

Changes

Changes from AIMer v2.1

- Prefix byte of H_0: 0x00 $\rightarrow$ 0x00,...,0x50
  - If message digest is same, a signer may generate same views throughout different parameter sets

Changes from AIMer v2.1

- Prefix byte of H_0: 0x00 $\rightarrow$ 0x00,...,0x50
  - If message digest is same, a signer may generate same views throughout different parameter sets
- Divide into internal/external functions
  - Internal function: deterministic, for debugging
  - External function: (possibly) probabilistic, for use

Changes from AIMer v2.1

- Prefix byte of H_0: 0x00 $\rightarrow$ 0x00,...,0x50
  - If message digest is same, a signer may generate same views throughout different parameter sets
- Divide into internal/external functions
  - Internal function: deterministic, for debugging
  - External function: (possibly) probabilistic, for use
- Remove ExpandIV function

Changes from AIMer v2.1

- Prefix byte of H_0: 0x00 $\rightarrow$ 0x00,...,0x50
  - If message digest is same, a signer may generate same views throughout different parameter sets
- Divide into internal/external functions
  - Internal function: deterministic, for debugging
  - External function: (possibly) probabilistic, for use
- Remove ExpandIV function
- Now index of arrays starts at 0 rather than 1

Changes from AIMer v2.1

- Prefix byte of H_0: 0x00 $\rightarrow$ 0x00,...,0x50
  - If message digest is same, a signer may generate same views throughout different parameter sets
- Divide into internal/external functions
  - Internal function: deterministic, for debugging
  - External function: (possibly) probabilistic, for use
- Remove ExpandIV function
- Now index of arrays starts at 0 rather than 1
- Now a context string is inputted to Sign/Verify similarly to ML-DSA

Name Changes from AIMer v2.1

- ReconstructSeedTree $\rightarrow$ ReconstructTree
- KeyGen $\rightarrow$ AIMer_keygen
- Sign $\rightarrow$ AIMer_sign
- Verify $\rightarrow$ AIMer_verify
- GenerateLinear $\rightarrow$ AIM2_GenerateLinear

Q&A