The AIMer Signature Scheme

Version 3.0

Authors Jihoon Cho (Samsung SDS)
Jincheol Ha (Soongsil University)
Seongkwang Kim (Korea University)
Jihoon Kwon (Samsung SDS)
Byeonghak Lee (Samsung SDS)
Joohee Lee (Sungshin Women's University)
Jooyoung Lee (KAIST)
Sangyub Lee (Samsung SDS)
Dukjae Moon (Samsung SDS)
Mincheol Son (KAIST)
Hyojin Yoon (Samsung SDS)

Contact team@aimer-signature.org

Homepage https://aimer-signature.org

Reference Code https://github.com/samsungsds-opensource/AIMer

Integrity Hash (SHA-256):

```
4b67f9bd 41005de6 789bc24b 86279f1c
0ecf685b 9de80397 d8cb459b cd1bee81
```

Friday 31st July, 2026

Contents

1 Introduction **5**
- 1.1 Overview of the Algorithm . . . 6
- 1.2 Notation . . . 7

2 Background **7**
- 2.1 Security Definitions . . . 7
- 2.2 MPC-in-the-Head Paradigm . . . 8
- 2.3 BN++ Proof System . . . 10
- 2.4 Fiat-Shamir Transform . . . 11
- 2.5 Gröbner Basis Attack . . . 11

3 Symmetric Primitive AIM3 **12**
- 3.1 Specification . . . 12
- 3.2 Design Rationale . . . 14

4 Specification of the AIMer Signature Scheme **16**
- 4.1 Basic Algorithms . . . 16
  - 4.1.1 Field Representation . . . 16
  - 4.1.2 Hash Functions . . . 17
  - 4.1.3 GGM Tree Evaluation . . . 18
  - 4.1.4 AIM3 Functions . . . 18
- 4.2 Signature Scheme . . . 20
  - 4.2.1 Key Generation . . . 22
  - 4.2.2 Signature Generation . . . 22
  - 4.2.3 Signature Verification . . . 25
- 4.3 Recommended Parameters . . . 25

5 Formal Security Analysis **25**
- 5.1 EUF-CMA Security of AIMer in the Random Oracle Model . . . 25

6 Security Evaluation **34**
- 6.1 Summary of Expected Security Strength . . . 34
- 6.2 Soundness Analysis . . . 36
- 6.3 Known Attacks to AIM3 . . . 37
  - 6.3.1 Algebraic Attacks . . . 37
  - 6.3.2 Brute-Force Attack . . . 41
  - 6.3.3 Quantum Attacks . . . 41
- 6.4 Attacks in the Multi-User Setting . . . 44
- 6.5 Side-Channel Attacks . . . 45

7 Key and Signature Sizes **46**

8 Advantages and Limitations **47**
- 8.1 General . . . 47
- 8.2 Compatibility with Existing Protocols . . . 47

A Quadratic-Equation Rank Computation 48

Change Log

v2.1 → v3.0

We have replaced the symmetric primitive AIM2 by AIM3. In AIM2, fixed constants are added before the inverse Mersenne S-boxes and a random affine layer is applied afterwards. In AIM3, an iv-dependent input affine layer is followed by enhanced Mersenne S-boxes

$$S[e](x) = x^{2^e-1} + x^{2^n-2},$$

and an output affine layer. Implicit quadratic relation of the S-box is valid only for nonzero inputs. Accordingly, key generation rejects $(\mathsf{iv}, \mathsf{pt})$ pairs for which any S-box input is zero.

The AIMer specification has been updated to use AIM3 throughout. We added AIM3-specific algorithms for linear-component generation, zero-input rejection, MPC simulation, and S-box output generation, and updated key generation, signing, verification, recommended exponents, signature-size calculations, and related notation.

The security analysis was revised for AIM3:

- The algebraic analysis now studies the quadratic systems induced by the enhanced Mersenne S-box and discusses the applicability of attacks previously considered for AIM2.
- The brute-force and quantum analyses were rewritten for AIM3, including Grover, GroverXL, and QuantumBooleanSolve.
- The information-theoretic proof of AIM2 in the random permutation model is no longer included; the formal security section now focuses on the EUF-CMA proof and the AIM3 one-wayness assumption.

The active document no longer includes the detailed v2.1 timing and memory benchmark sections. We added an appendix with exact quadratic-equation rank computations for the recommended AIM3 parameters and updated the corresponding references and security tables.

v2.0 → v2.1

In this version, the updates are mainly related to the implementation. We updated our implementations to be more friendly to PQClean project and run all tests of PQClean test framework. We merged **Reference** C and **Optimized** C to **Reference** C, and change the name **AVX2** implementation to **Optimized** implementation. We added **mem_opt** C implementation for memory-constrained devices, and **aarch64_shake_opt** implementation which utilizes ARM Advanced SIMD instructions on SHAKE. Now **aarch64** and **aarch64_shake_opt** implementations can be compiled for ARM-based Apple SoCs (Apple M series). In response to the recommendations of Bernstein in KpqC bulletin,[1] we have applied following patches:

[1] https://groups.google.com/g/kpqc-bulletin/c/_Gb2n7ZwlTo

- Since the variables of patch-1-reveal, patch-7-commits, and patch-8-alpha were public data, we have utilized `crypto_declassify` function.
- patch-2-poly64: we replaced `poly64_mul` by `poly64_mul_s` which contains all the recommendation, and applied it to all the arithmetic operations related to secret data.
- patch-3-htole: we replaced `htole64` and `ltohe64` with the recommended byte computations, and removed `portable_endian.h` file.
- patch-4-loadstore: we replaced `_load_` and `_store_` with `_loadu_` and `_storeu_` in the **AVX2** implementation.
- patch-5-square: we modified all implementations to use the recommended code for square arithmetic in the **Reference** and **mem_opt** implementations.
- patch-6-selfaddmask: we removed the `selfaddmask` function from all implementations.
- patch-9-initialize: we added the recommended initialization process in the **AVX2** implementation.
- Lastly, we have included TIMECOP results for all TIMECOP-supported implementations.

v1.0 → v2.0

We have modified the core specification to enhance security, efficiency, and usability. The main change is the update from AIM to AIM2, which mitigates the analyses on AIM. We reduced the salt size by half without any security degradation. For usability, we introduced a message pre-hashing mechanism, and reduced the number of parameter sets (and we renamed them).

From an implementation perspective, we newly implemented the reference source code, improving readability. This version additionally supports the aarch64 architecture, reference C, optimized C, and AVX2 builds. We have removed the OpenSSL dependency and optimized memory usage for all implementations. These changes aim to provide a more readable, efficient, and versatile implementation framework for users and developers alike.

Editorial revisions have been made to align the documentation with the aforementioned specification and implementation changes. We revised the EUF-CMA security proof, security analysis, and performance figures. The specification became more implementation-friendly as each specification has a link to its implementation.

v0.9 → v1.0

We integrated Commit and ExpandTape to a single hash function, improved the matrix generation algorithm, and reduced the entropy requirement for the gener-

ation of root seeds for better performance. To enhance concrete security, we added the domain separation mechanism to the hash functions.

1 Introduction

AIMer is a signature scheme which is obtained from a zero-knowledge proof of preimage knowledge for a certain one-way function. AIMer consists of two parts: a non-interactive zero-knowledge proof of knowledge (NIZKPoK) system, and a one-way function. The security of both parts solely depends on the security of the underlying symmetric primitives.

The NIZKPoK system in AIMer can be viewed as a customized version of the BN++ proof system [KZ22]. BN++ is a NIZKPoK system based on the MPC-in-the-Head (MPCitH) paradigm [IKOS07], which efficiently proves large-field arithmetic. The difference between our system and BN++ is given as follows.

- Our system integrates Commit and ExpandTape to a single hash function. It reduces a significant amount of signing and verification time without loss of security in the random oracle model.
- Hash functions and extendable-output functions used in our system are domain-separated for stronger concrete security.
- The size of salt is halved.
- The hash value of the message is precomputed to efficiently handle a long message.
- Our system requires a smaller amount of randomness to generate the master seeds (seed_k) for each repetition.

The one-way function of AIMer in version 1.0 was AIM [KHS+23], which is a tweakable one-way function dedicated to the BN++ system. AIM was designed to have strong security against algebraic attacks producing short signatures when combined with BN++.

After AIM was introduced, several studies have revealed algebraic vulnerabilities [LMOM23, ZWY+23]: the most notable is the fast exhaustive search of Liu et al., which exploits the fact that AIM admits a low-degree system of equations in a moderate number of Boolean variables and degrades its security by up to 12 bits. AIM2 [KHSL24] was proposed to mitigate these attacks and was mounted as the symmetric primitive of AIMer version 2.0, however, the *magic-pot attack* proposed by Biryukov et al. [BGFU26] degrades the claimed security of AIM2.

To restore the security level of the one-way function used in AIMer, we introduce a new symmetric primitive AIM3 with the following structural changes. First, it prepends an iv-generated random invertible linear layer before the first-round S-boxes instead of adding fixed constants, so that the secret pt enters the S-boxes at high algebraic degree. The addition of this linear layer was also part of the very early AIM design (prior to v1) and is in line with the approach proposed by Sun et al. [SCL+25] to enhance the security of AIM.

Second, it replaces the invertible Mersenne S-box with a non-invertible S-box $S[e](x) = x^{2^e-1} + x^{2^n-2}$. Overall, AIM3 provides stronger security against recent attacks on AIM2. In AIMer version 3.0, we mount AIM3 as its symmetric primitive.

1.1 Overview of the Algorithm

The AIMer signature algorithm consists of key generation, signing, and verification algorithms. To provide an intuitive understanding of the AIMer signature scheme, we will briefly describe the three algorithms below. The detailed specification is given in Section 4.

KEY GENERATION. Key generation is basically an AIM3 computation for random pt and iv; however, since cases where the AIM3 S-box receives 0 as input are not suitable for the BN++ proof system, we select $(\mathsf{pt}, \mathsf{iv})$ for which this does not occur. The key generation algorithm proceeds as follows.

1. A plaintext pt and a tweak iv are sampled uniformly at random.
2. $\mathsf{ct} = \mathsf{AIM3}(\mathsf{pt}, \mathsf{iv})$ is computed.
3. If there exists an AIM3 S-box which has input with 0, go back to Step 1.
4. The secret key is set to $sk = (\mathsf{pt}, \mathsf{iv}, \mathsf{ct})$, and the corresponding public key is defined as $pk = (\mathsf{iv}, \mathsf{ct})$.

SIGNING ALGORITHM. The signing algorithm is a virtual MPC simulation of AIM3. The multiple parties involved in the MPC evaluation are not real participants, but a simulation by the signer (MPCitH). As both signing and verification algorithms are non-interactive, random challenges are computed by hash functions (via the Fiat-Shamir transform). The signing algorithm proceeds as follows.

1. The signer prepares the MPC simulation; it generates seeds for each party, and shares of the input and intermediate values appearing in the computation of AIM3 from each seed. The signer commits each seed.
2. The signer computes a multiplication-checking protocol from a challenge.
3. The signer opens all the views except one determined by another challenge.

VERIFICATION ALGORITHM. The verification algorithm is a recomputation of the signing algorithm to check whether the MPC simulation has been faithfully executed or not. The verification algorithm mainly checks two steps: preparation of the MPC simulation, and the multiplication-checking protocol. The verification algorithm proceeds as follows.

1. The verifier recomputes shares of all the parties except the unopened one, and computes the first challenge.
2. The verifier recomputes the multiplication-checking protocol, and computes the second challenge.
3. The verifier checks whether the opened views of the MPC simulation are consistent or not.

1.2 Notation

Unless stated otherwise, all logarithms are to the base 2. For a positive integer n, we denote the set of all bitstrings of bitlength n by $\{0,1\}^n$. We also denote the set of all finite-length bitstrings by $\{0,1\}^*$. For two vectors a and b over a finite field or two bitstrings a and b, their concatenation is denoted by $a \parallel b$. For a nonnegative integer n, we write $[n] = \{0, \cdots, n-1\}$. For a nonnegative integer n, $(a_i)_{i\in[n]}$ stands for a list $(a_0, a_1, \ldots, a_{n-1})$. For a nonnegative integer n, iterator in a "For" loop iterating $[n]$ starts from 0 and increases to $n-1$ by 1. We will write $a \leftarrow b$ to denote the assignment of b to a. For a set S, $a \rightarrow S$ denotes that a is added to S as an element, and $a \leftarrow_{\$} S$ denotes that a is chosen uniformly at random from S.

In this document, additions are usually operated on a binary field, in which case additions are exclusive-OR (XOR). Nevertheless, when we want to emphasize that an addition is actually XOR, we denote the addition by $\oplus$. In the multi-party computation setting, $x^{(i)}$ denotes the i-th party's additive share of x, which implies that $\sum_i x^{(i)} = x$. We summarize some notations of parameters and non-conventional notations in Table 1.

In this paper, index of every vector or list starts from 0. When a vector is multiplied to a matrix, the vector is interpreted as a column vector even if there is no explicit transpose notation ($^\top$). For a vector `vec`, the notation $\texttt{vec}[n]$ is used to denote the n-th element of `vec`. For a vector `vec`, $\texttt{vec}[a:b]$ denotes the sub-vector of $b-a$ elements from $\texttt{vec}[a]$ (inclusive) to $\texttt{vec}[b]$ (exclusive). For a bitstring `str`, similar to vectors, we use $\texttt{str}[n]$ and $\texttt{str}[a:b]$ to denote n-th bit of `str` and sub-string from bit-position a (inclusive) to b (exclusive), respectively. Bitlength of a bitstring a is denoted by $|a|$. For a bitstring a, we denote $a \ll i$ (resp. $a \gg i$) logical left (resp. right) shift operation by i, where the bitlength of the output bitstring is $|a|$ and shifted positions are filled with 0s. We write bitstrings in hexadecimal format, with big-endian order. For example, we write

$$\texttt{0x0603}[0:8] = 0000\ 0110\ 0000\ 0011[0:8] = \texttt{0x06}.$$

λ	Security parameter
n	Input/output bit-length of S-boxes in AIM3 (which is always same as λ)
ℓ	Number of S-boxes in front of the linear layer in AIM3
τ	Number of the parallel repetitions in NIZKPoK
N	Number of the parties in NIZKPoK (which is always a power-of-two)

Table 1: The notation used in the document.

2 Background

2.1 Security Definitions

PRF SECURITY. Let $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$ be a keyed function from $\mathcal{X}$ to $\mathcal{Y}$ with key space $\mathcal{K}$. A (probabilistic) adversary $\mathcal{A}$ against the PRF security of F makes a certain

number of queries $F(k, x)$ where $k \in \mathcal{K}$ is chosen uniformly at random from the key space and kept secret, and tries to distinguish F from a truly random function. More formally, the advantage of $\mathcal{A}$ against the PRF security of F is defined as

$$\mathbf{Adv}_F^{\mathsf{prf}}(\mathcal{A}) \coloneqq \left|\Pr[\mathcal{A}^{F(k,\cdot)} = 1] - \Pr[\mathcal{A}^{g(\cdot)} = 1]\right|,$$

where g denotes a truly random function that has been chosen uniformly at random from the set of all possible functions from $\mathcal{X}$ to $\mathcal{Y}$.

ONE-WAYNESS. Given a function $F : \{0,1\}^n \to \{0,1\}^m$ and $y \in \{0,1\}^m$, the goal of a (probabilistic) preimage-finding adversary $\mathcal{A}$ is to find $x \in \{0,1\}^n$ such that $y = F(x)$. Formally, the advantage of $\mathcal{A}$ against the one-wayness of F is defined as

$$\mathbf{Adv}_F^{\mathsf{owf}}(\mathcal{A}) \coloneqq \Pr\left[x \leftarrow \mathcal{A}(y) \wedge F(x) = y\right] \tag{1}$$

where $y = F(z)$ for a random $z \in \{0,1\}^n$. This notion of one-wayness will be used in the security proof of the AIMer signature scheme.

EUF-KO SECURITY. The existential unforgeability of a signature scheme Π under key-only attacks (EUF-KO) ensures that no probabilistic adversary $\mathcal{A}$ is able to compute a valid signature on any message m without having access to a signing oracle. In this model, the forging advantage of $\mathcal{A}$ against $\Pi = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ is defined as

$$\mathbf{Adv}_\Pi^{\mathsf{euf\text{-}ko}}(\mathcal{A}) \coloneqq \Pr\left[\mathsf{Verify}(pk, m, \sigma) = 1 \;\middle|\; \begin{array}{l}(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)\\ (m, \sigma) \leftarrow \mathcal{A}(pk)\end{array}\right],$$

where λ is the security parameter.

EUF-CMA SECURITY. The existential unforgeability of a signature scheme Π under chosen message attacks (EUF-CMA) ensures that no probabilistic adversary $\mathcal{A}$ is able to compute a valid signature on any message that has not been signed during the attack, despite having observed the signatures on a certain number of chosen messages. More formally, the forging advantage of $\mathcal{A}$ against $\Pi = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ is defined as

$$\mathbf{Adv}_\Pi^{\mathsf{euf\text{-}cma}}(\mathcal{A}) \coloneqq \Pr\left[\begin{array}{l}\mathsf{Verify}(pk, m, \sigma) = 1\\ \wedge\, m \text{ is not signed before.}\end{array} \;\middle|\; \begin{array}{l}(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)\\ (m, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot)}(pk)\end{array}\right],$$

where λ is the security parameter, and $\mathcal{A}^{\mathsf{Sign}(sk,\cdot)}$ implies that $\mathcal{A}$ has access to the signing oracle with private key sk.

2.2 MPC-in-the-Head Paradigm

The MPC-in-the-Head (MPCitH) paradigm, proposed by Ishai et al. [IKOS07], allows one to construct a zero-knowledge proof (ZKP) system from a multi-party computation (MPC) protocol. Consider an MPC protocol where N parties collaborate to securely evaluate a function f on an input x with perfect correctness. Suppose that the views of k parties leak no information on x. Then, one can build a ZKP from the MPC protocol as follows.

1. The prover generates random secret shares $x^{(0)}, \ldots, x^{(N-1)}$ such that $x^{(0)} + \cdots + x^{(N-1)} = x$, and assigns them to N parties, say $\mathcal{P}_0, \ldots, \mathcal{P}_{N-1}$.

2. The prover simulates the MPC protocol "in her head" by simulating each $\mathcal{P}_i$, $i = 0, \ldots, N-1$.

3. The prover commits to each party's view which includes its random tape, the secret input share, and the communicated messages from and to the party. She sends the commitments to the verifier.

4. The prover possibly gets random challenges for MPC simulation from the verifier when needed, and conducts local computations on each party. She may repeat this step several times.

5. The prover completes the MPC simulation and hands over requested output shares of the MPC protocol to the verifier.

Note that the verifier interactively joins the above procedure to provide random challenges to the prover. After that, the verifier selects k parties and asks the prover to open their views. Once the views are received, the verifier checks

1. if the opened views are consistent, i.e., the messages sent from and to a party match and the commitments are correctly evaluated from the resulting views, and

2. if the output recovered from the output share is y.

Since only k views are opened, no information on x is leaked from the revealed views. Also, since the verifier opens the random views, any cheating adversary's winning probability is upper bounded by $(N-k)/N$. We fix $k = N-1$ throughout this proposal.

The practicality of MPCitH is demonstrated by the ZKBoo scheme, the first efficient MPCitH-based proof scheme proposed by Giacomelli et al. [GMO16]. One of the main applications of the MPCitH paradigm is to construct a post-quantum signature. Picnic [CDG+17] is the first and the most famous signature scheme based on the MPCitH paradigm; it combines an MPC-friendly block cipher LowMC [ARS+15] and an MPCitH proof system called ZKB++, which is an optimized variant of ZKBoo. Katz et al. [KKW18] proposed a new proof system KKW by further improving the efficiency of ZKB++ with pre-processing, and updated Picnic accordingly. The updated version of Picnic was the only MPCitH-based scheme that advanced to the third round of the NIST PQC competition. BBQ [dSGMOS19] and Banquet [BSGK+21] are AES-based signature schemes, where BBQ employs the KKW proof system and Banquet improves BBQ by injecting shares for intermediate states.

To fully exploit efficient multiplication over a large field in the Banquet proof system, Dobraunig et al. [DKR+22] proposed MPCitH-friendly ciphers LS-AES and Rain. They are substitution-permutation ciphers based on the inverse S-box over a large field. This design strategy increases the efficiency of the resulting MPCitH-based signature scheme, while the number of rounds should be carefully determined by comprehensive analysis on any possible algebraic attack due to their

simple algebraic structures. Kales and Zaverucha [KZ22] proposed several optimization techniques to further improve the efficiency of the Baum and Nof's proof system [BN20], and their variant is called BN++.

2.3 BN++ Proof System

In this section, we briefly review the BN++ proof system [KZ22], one of the state-of-the-art MPCitH zero-knowledge protocols. The BN++ protocol will be combined with our symmetric primitive AIM3 to construct the AIMer signature scheme. At a high level, BN++ is a variant of the BN protocol [BN20] with several optimization techniques applied to reduce the signature size.

PROTOCOL OVERVIEW. The BN++ protocol follows the MPCitH paradigm [IKOS07]. In order to check C multiplication triples $(x_j, y_j, z_j = x_j \cdot y_j)_{j=0}^{C-1}$ over a finite field $\mathbb{F}$ in the multiparty computation setting with N parties, *helping triples* $((a_j, b_j)_{j=0}^{C-1}, c)$ are required, where $a_j \in \mathbb{F}, b_j = y_j$, and $c = \sum_{j=0}^{C-1} a_j \cdot b_j$. Each party holds secret shares of the multiplication triples $(x_j, y_j, z_j)_{j=0}^{C-1}$ and the helping triples $((a_j, b_j)_{j=0}^{C-1}, c)$. Then the protocol proceeds as follows.

- A prover is given random challenges $\epsilon_0, \cdots, \epsilon_{C-1} \in \mathbb{F}$.
- For $i \in [N]$, the i-th party locally sets $\alpha_0^{(i)}, \cdots, \alpha_{C-1}^{(i)}$ where $\alpha_j^{(i)} = \epsilon_j \cdot x_j^{(i)} + a_j^{(i)}$.
- The parties open $\alpha_0, \cdots, \alpha_{C-1}$ by broadcasting their shares.
- For $i \in [N]$, the i-th party locally sets

$$v^{(i)} = \sum_{j=0}^{C-1} \epsilon_j \cdot z_j^{(i)} - \sum_{j=0}^{C-1} \alpha_j \cdot b_j^{(i)} + c^{(i)}.$$

- The parties open v by broadcasting their shares and output `Accept` if $v = 0$.

The probability that there exist incorrect triples and the parties output `Accept` in a single run of the above steps is upper bounded by $1/|\mathbb{F}|$.

SIGNATURE SIZE. By applying the Fiat-Shamir transform [DFM20], one can obtain a signature scheme from the BN++ proof system. In this signature scheme, the signature size is given as

$$6\lambda + \tau \cdot (3\lambda + \lambda \cdot \lceil \log_2(N) \rceil + \mathcal{M}(C)),$$

where λ is the security parameter, τ is the number of parallel repetitions of the multiplication checking protocol for reducing the soundness error, C is the number of multiplication gates in the underlying symmetric primitive, and $\mathcal{M}(C) = (2C + 1) \cdot \log_2(|\mathbb{F}|)$. In particular, $\mathcal{M}(C)$ has been defined so from the observation that sharing the secret share offsets for $(z_j)_{j=0}^{C-1}$ and c, and opening shares for $(\alpha_j)_{j=0}^{C-1}$ occurs for each repetition, using C, 1, and C elements of $\mathbb{F}$, respectively. For more details, we refer to [KZ22].

If the output of the multiplication z_i can be locally generated from each share, then the secret share offset is not necessarily included in the signature.

2.4 Fiat-Shamir Transform

The Fiat-Shamir transform [FS87] is a technique for taking an interactive proof of knowledge and creating a non-interactive counterpart, or a digital signature based on it. The core of the technique is to replace challenges from the verifier by random oracle access which is realized by hashing of the transcript obtained so far.

The Fiat-Shamir transform was originally targeted at a Σ-protocol, a three-round interactive proof of knowledge. Let R be a relation such that, for a given x, it is difficult to find a witness w such that $R(x, w) = 1$. Given public R and x, the value w such that $R(x, w) = 1$ becomes the secret information that a prover P wants to prove the knowledge of to the verifier V. Then, a Σ-protocol proceeds as follows.

1. **Commitment**: a random number r is generated, committed to by the prover, and sent to the verifier.

   $\mathsf{P} \xrightarrow{\text{com}} \mathsf{V}, \quad \text{where com} = \mathsf{Commit}(r).$

2. **Challenge**: on receiving the commitment, the verifier sends a random challenge ch to the prover.

   $\mathsf{P} \xleftarrow{\text{ch}} \mathsf{V}.$

3. **Response**: the prover creates an appropriate response corresponding to the challenge.

   $\mathsf{P} \xrightarrow{\text{res}} \mathsf{V}, \quad \text{where res} = \mathsf{Response}(w, r, \text{ch}).$

Then, the verifier checks the validity of the response together with com and ch. This Σ-protocol is transformed into a non-interactive version, by replacing the challenge sent by the verifier by a random oracle access, using the previous transcript (x, com). Denoting the random oracle as $\mathcal{RO}$, the challenge step of the above procedure is replaced by $\text{ch} \leftarrow \mathcal{RO}(x, \text{com})$. This approach can be extended to multi-round proofs. The security loss is known to be linear in the number of attacker's queries to the random oracle [AFK22].

2.5 Gröbner Basis Attack

The Gröbner basis attack aims to solve systems of equations by determining their Gröbner basis through a structured approach. The process unfolds in several stages:

1. Calculation of a Gröbner basis using the graded reverse lexicographic (*grevlex*) order.

2. Conversion of the basis into lexicographic (*lex*) order by reordering terms.

3. Identification and finding a solution of a univariate polynomial equation within the basis.

4. Substitution of the solution back into the basis, with iterative applications of the previous step for further solutions.

A system's Gröbner basis in lex order always contains a univariate polynomial when the system has a finite number of solutions within its algebraic closure. When a single variable of the polynomial is replaced by a concrete solution, the Gröbner basis still remains a Gröbner basis of the "reduced" system, allowing one to obtain a univariate polynomial again for the next variable. For a comprehensive understanding of Gröbner basis calculation, the reader is directed to [SS21].

The resilience of a cryptographic system against the Gröbner basis attack primarily depends on the complexity of the first step, which is computing the Gröbner basis in grevlex order, typically using the F4/F5 algorithm or its variants [Fau99, Fau02]. This complexity can be estimated through the system's degree of regularity [BFS04]. Consider a system of m homogeneous equations $\{f_i(x_0, \ldots, x_{n-1}) = 0\}_{i=0}^{m-1}$ in n Boolean variables. Let d_i denote the degree of f_i for $i = 0, 1, \ldots, m-1$. Assuming that almost all polynomial sequences are semi-regular [Frö85], then the degree of regularity can be estimated for overdetermined systems ($m > n$) by the smallest degree of the terms with non-positive coefficients appearing in the Hilbert series as follows.

$$\frac{(1+z)^n}{\prod_{i=0}^{m-1}(1+z^{d_i})}.$$

For nonhomogeneous equations, the degree of regularity comes from the following Hilbert series obtained by homogenization [BFSS13].

$$\frac{(1+z)^n}{(1-z)\prod_{i=0}^{m-1}(1+z^{d_i})}. \tag{2}$$

Given the degree of regularity d_{reg}, the complexity is

$$\binom{n}{d_{\mathrm{reg}}}^{\omega}$$

ignoring the constant factor, where ω is the linear algebra constant ($2 \leq \omega \leq 3$). Combined with the hybrid approach of guessing some variables, the time complexity of the hybrid Gröbner basis attack is given by

$$\min_k 2^k \cdot \binom{n-k}{d_{\mathrm{reg}}(n,k)}^{\omega} \tag{3}$$

where $d_{\mathrm{reg}}(n,k)$ denotes the minimal degree from the Hilbert series after adjusting for guessed variables. This formula with $\omega = 2$ provides a conservative estimate of the complexity, and we use this formula to estimate the complexity in this paper.

3 Symmetric Primitive AIM3

3.1 Specification

AIM3 is designed to be a "tweakable" one-way function so that it offers multi-target one-wayness. Given input/output size n and an $(\ell+1)$-tuple of exponents $(e_0, \ldots, e_\ell) \in \mathbb{Z}^{\ell+1}$, for $\mathsf{pt} \in \mathbb{F}_{2^n}$ and $\mathsf{iv} \in \{0,1\}^n$, $\mathsf{AIM3}(\mathsf{pt}, \mathsf{iv})$ is defined by

$$\mathsf{AIM3}(\mathsf{pt}, \mathsf{iv}) = S[e_\ell] \circ \mathsf{Lin}[\mathsf{iv}] \circ S[e_0, \ldots, e_{\ell-1}] \circ \mathsf{Lin}_{\mathsf{in}}[\mathsf{iv}](\mathsf{pt}) \oplus \mathsf{pt}$$

where each function will be described below. See Figure 1 for the pictorial description of AIM3 with $\ell = 2$.

NON-LINEAR COMPONENTS. AIM3 uses the enhanced Mersenne S-box

$$S[e](x) = x^{2^e-1} + x^{2^n-2}.$$

defined by exponentiation over a large field as follows. We remark that the exponents e are chosen such that $\gcd(e, n) = 1$. As an extension, $S[e_0, \ldots, e_{\ell-1}] : \mathbb{F}_{2^n}^{\ell} \to \mathbb{F}_{2^n}^{\ell}$ is defined by

$$S[e_0, \ldots, e_{\ell-1}] = S[e_0](x_0) \| \ldots \| S[e_{\ell-1}](x_{\ell-1}).$$

LINEAR COMPONENTS. AIM3 includes three types of linear components: an input affine layer $\mathsf{Lin}_{\mathsf{in}}[\mathsf{iv}]$, an output affine layer $\mathsf{Lin}[\mathsf{iv}]$, and feed-forward.

Linear components used in affine layers are generated by

$$\big((A_i)_{i\in[2\ell]}, (b_i)_{i\in[\ell+1]}\big) \leftarrow \mathsf{GenerateLinear}(\mathsf{iv})$$

where each A_i is an invertible linearized polynomial over $\mathbb{F}_{2^n}$ for $i \in [2\ell]$, and $b_i \in \mathbb{F}_{2^n}$ for $i \in [\ell+1]$. GenerateLinear is a function that generates linear components and internally uses $\mathsf{XOF}(\mathsf{iv})$. As an invertible linearized polynomial over $\mathbb{F}_{2^n}$ can be equivalently represented by an invertible n-by-n matrix over $\mathbb{F}_2$, GenerateLinear actually generates matrices in $\mathsf{GL}_n(\mathbb{F}_2)$ instead of invertible linearized polynomials. The full definition of AIM3 including GenerateLinear will be described later in Section 4.1.4.

Then the input affine layer $\mathsf{Lin}_{\mathsf{in}}[\mathsf{iv}]$ is defined by

$$\mathsf{Lin}_{\mathsf{in}}[\mathsf{iv}](x) = (A_0(x) + b_0) \| \ldots \| (A_{\ell-1}(x) + b_{\ell-1})$$

and the output affine layer $\mathsf{Lin}_{\mathsf{out}}[\mathsf{iv}]$ is defined by

$$\mathsf{Lin}[\mathsf{iv}]((x_0, \ldots, x_{\ell-1})) = b_\ell + \sum_{i\in[\ell]} A_{\ell+i}(x_i).$$

Feed-forward operation, which is addition by the input itself, makes the entire function non-invertible.

RECOMMENDED PARAMETERS. Table 2 describes the recommended sets of parameters for $\lambda \in \{128, 192, 256\}$. The irreducible polynomials for extension fields $\mathbb{F}_{2^{128}}$, $\mathbb{F}_{2^{192}}$, and $\mathbb{F}_{2^{256}}$ are as follows.

- $\mathbb{F}_{2^{128}} : f(X) = X^{128} + X^7 + X^2 + X + 1$,
- $\mathbb{F}_{2^{192}} : f(X) = X^{192} + X^7 + X^2 + X + 1$,
- $\mathbb{F}_{2^{256}} : f(X) = X^{256} + X^{10} + X^5 + X^2 + 1$.

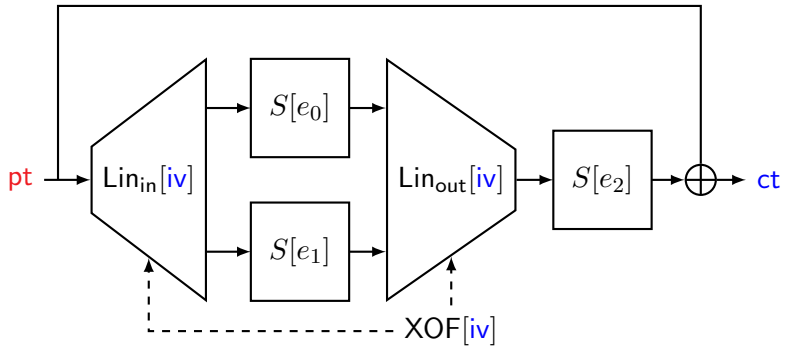


Figure 1: The AIM3-I one-way function with $\ell = 2$. The input pt (in red) is the secret key of the signature scheme, and $(\mathsf{iv}, \mathsf{ct})$ (in blue) is the corresponding public key.

Scheme	λ	n	ℓ	e_0	e_1	e_2	e_3
AIM3-I	128	128	2	3	7	11	-
AIM3-III	192	192	2	11	23	47	-
AIM3-V	256	256	3	3	7	11	19

Table 2: Recommended sets of parameters of AIM3.

3.2 Design Rationale

CHOICE OF FIELD. When a symmetric primitive is built upon field operations, the field is typically binary since bitwise operations are cheap in most of modern architectures. However, when the multiplicative complexity of the primitive becomes a more important metric for efficiency, it is hard to generally specify which type of field has merits with respect to security and efficiency.

Focusing on a primitive for MPCitH-style zero-knowledge protocols, a primitive over a large field generally requires a small number of multiplications, leading to shorter signatures. However, any primitive operating on a large field of a large prime characteristic might permit algebraic attacks since the number of variables and the algebraic degree will be significantly limited for efficiency reasons. On the other hand, binary extension fields enjoy both advantages from small and large fields. In particular, matrix multiplication is represented by a polynomial of high algebraic degree without increasing the proof size.

ALGEBRAICALLY SOUND S-BOXES. In an MPCitH-style zero-knowledge protocol, the proof size of a circuit is usually proportional to the number of nonlinear operations in the circuit. In order to minimize the number of multiplications, one might introduce intermediate variables for some wires of the circuit. For example, the inverse S-box ($S(x) = x^{-1}$) has high (bitwise) algebraic degree $n-1$, while it can be simply represented by a quadratic equation $xy = 1$ by letting the output from the S-box be a new variable y. When an S-box is represented by a quadratic equation of its input and output, we will say it is *implicitly quadratic*. In particular, we consider implicitly quadratic S-boxes which are represented by a single multiplication over

$\mathbb{F}_{2^n}$. This feature makes the proof size short and mitigates algebraic attacks at the same time.

The inverse S-box is one of the well-studied implicitly quadratic S-boxes. The inverse S-box has been widely adopted to symmetric ciphers due to its nice cryptographic properties [DR02, AIK+01, SSA+07]. It is invertible, is of high-degree, and has good enough differential uniformity and nonlinearity. Recently, it has been used in symmetric primitives for advanced cryptographic protocols such as multi-party computation and zero-knowledge proof [GKR+21, GLR+20, DKR+22].

Meanwhile, the inverse S-box has one minor weakness; a single evaluation of the n-bit inverse S-box as a form of $xy = 1$ produces $5n - 1$ linearly independent quadratic equations over $\mathbb{F}_2$ [CDG06]. The complexity of an algebraic attack is typically bounded (with heuristics) by the degree and the number of equations, and the number of variables. In particular, an algebraic attack is more efficient with a larger number of equations, while this aspect has not been fully considered in the design of recent symmetric ciphers based on algebraic S-boxes. When the number of rounds is small, this issue might be critical to the overall security of the cipher. For more details, see Section 6.3.1.

With the above observation, we seek an S-box that admits a one-multiplication input-output representation, induces only a small number of Boolean quadratic equations, and has high algebraic degree. Since our attack model does not allow multiple queries to a single instance of AIM3, we allow a relaxed condition on the resistance against differential and linear cryptanalysis, not necessarily requiring the optimal bounds.

The Mersenne S-box $M[e](x) = x^{2^e-1}$ satisfies several of these requirements. It has a one-multiplication representation: if $y = M[e](x)$, then

$$xy = x^{2^e}.$$

Hence, its input-output relation can be verified with one multiplication in the BN++ proof system. Moreover, it induces only $3n$ Boolean quadratic equations between its input and output, and provides moderate resistance to differential and linear cryptanalysis. However, the algebraic degree of $M[e]$ is only e, which is substantially smaller than the degree $n - 1$ of the inverse map.

To overcome this limitation while preserving the one-multiplication representation, AIM3 uses the enhanced Mersenne S-box

$$S[e](x) = x^{2^e-1} + x^{2^n-2}.$$

The second term is the formal inverse power over $\mathbb{F}_{2^n}$; in particular, $0^{2^n-2} = 0$, while $x^{2^n-2} = x^{-1}$ for $x \neq 0$. Since AIM3 rejects instances in which an S-box input is zero, the S-box relation is used only on nonzero inputs in valid executions. For $x \neq 0$ and $y = S[e](x)$, we have

$$xy = x^{2^e} + 1.$$

Conversely, if $xy = x^{2^e} + 1$, then $x \neq 0$ and $y = x^{2^e-1} + x^{-1} = S[e](x)$. Therefore, one multiplication still suffices to prove the S-box relation.

Unlike the Mersenne and inverse Mersenne S-boxes used in AIM2, the enhanced Mersenne S-box is not a permutation over the full field $\mathbb{F}_{2^n}$. Nevertheless, it is at most two-to-one: for any fixed output y, every nonzero preimage x satisfies

$$x^{2^e} + yx + 1 = 0,$$

which is an affine equation whose homogeneous part $x^{2^e} + yx$ has kernel of size at most two, since $\gcd(n, e) = 1$. Hence every output value has at most two preimages. This mild non-injectivity is acceptable because AIM3 is used as a one-way function, and the proof system only needs to verify the forward computation.

AFFINE LAYER GENERATION. The main advantage of using binary affine layers in large S-box-based constructions is to increase the algebraic degree of equations over the large field. Multiplication by a random $n \times n$ binary matrix can be represented as

$$\sum_{i=0}^{n-1} a_i x^{2^i} = a_0 x + a_1 x^{2^1} + a_2 x^{2^2} + \cdots + a_{n-1} x^{2^{n-1}}$$

where $a_0, a_1, \ldots, a_{n-1} \in \mathbb{F}_{2^n}$. Similarly, our design uses a random affine map from $\mathbb{F}_2^{\ell n}$ to $\mathbb{F}_2^n$. In order to mitigate multi-target attacks (in the multi-user setting), the affine map is uniquely generated for each user; each user's iv is fed to an XOF, generating the corresponding linear layer.

4 Specification of the AIMer Signature Scheme

4.1 Basic Algorithms

Before providing the detailed specifications of the AIMer signature scheme, we introduce the foundational algorithms that underpin the signature scheme. In the forthcoming sections, we provide detailed algorithmic descriptions of the conversion processes for inputs and outputs, the exact functionalities of hash functions, and various auxiliary functions that play crucial roles in our signature scheme.

4.1.1 Field Representation

Many variables in AIMer are considered to be elements of $\mathbb{F}_{2^n}$, thus they need to be converted to bitstrings to be used as inputs/outputs of hash functions, and to be used as inputs/outputs of the linear layer of AIM3. The finite field is defined as $\mathbb{F}_{2^n} = \mathbb{F}_2[X]/f(X)$ where $f(X)$ is the irreducible polynomial defined in Section 3.1. The conversions from an element in $\mathbb{F}_{2^n}$ to a vector or a bitstring are defined as follows.

$$\begin{array}{ccccc} \{0,1\}^n & \longleftrightarrow & \mathbb{F}_{2^n} & \longleftrightarrow & \mathbb{F}_2^n \\ a_0 \| \ldots \| a_{n-1} & \leftrightarrow & \sum_{i\in[n]} a_i \cdot X^i & \leftrightarrow & (a_0, \ldots, a_{n-1})^\top \end{array}$$

For example, a bitstring $\texttt{0xA0}\underbrace{\texttt{0}\ldots\texttt{0}}_{28}\texttt{01}$ represented in hexadecimal form can be converted into $X^{127} + X^{125} + 1$ in $\mathbb{F}_{2^{128}}$. In our specification, we sometimes refer to elements of $\mathbb{F}_{2^n}$ as elements of $\mathbb{F}_2^n$ or $\{0,1\}^n$ depending on the context.

4.1.2 Hash Functions

All hash functions are instantiated using SHAKE128 if $\lambda = 128$, and SHAKE256 if $\lambda \in \{192, 256\}$ [NIS15]. For a bitstring x and positive integer d, we denote

$$\mathsf{SHAKE}_\lambda(x, d) = \begin{cases} \mathsf{SHAKE128}(x, d) & \text{if } \lambda = 128 \\ \mathsf{SHAKE256}(x, d) & \text{if } \lambda = 192, 256 \end{cases}$$

where $\mathsf{SHAKE128}(x, d)$ and $\mathsf{SHAKE256}(x, d)$ mean the digest of x using SHAKE128 and SHAKE256 with d-bit length, respectively. To distinguish between domains, we hash the input with a single-byte prefix, which is the same as the number i in the subscript of H_i except for $i = 0$. For example, H_2 uses `0x02` as the domain separation prefix. Since there are field elements, integers, and tuples in the inputs and outputs of hash functions, we apply the following rules to them.

- For the field elements in inputs and outputs of hash functions, we use conversion between field elements and n-bit strings as we described in 4.1.1.
- For integers in the inputs of hash functions, we use the standard byte representation as they always fit in a byte (i.e. from 0 to 255).
- For tuples used as input/output of hash functions, we convert each element to/from a string and concatenate/split them in ascending order.

For example, H_4: $\{0,1\}^\lambda \times [\tau] \times [N] \times \{0,1\}^\lambda \to (\{0,1\}^\lambda)^2$ is a domain-separated hash function with prefix 4. Then,

$$H_4\,(\mathsf{salt}, 1, 255, \mathsf{node}) = (\mathsf{tape}[0:\lambda], \mathsf{tape}[\lambda : 2\lambda])$$

where

$$\mathsf{tape} = \mathsf{SHAKE}_\lambda(\texttt{0x04} \parallel \mathsf{salt} \parallel \texttt{0x01} \parallel \texttt{0xff} \parallel \mathsf{node}, 2\lambda).$$

The list of specific functions used in AIMer is as follows.

- H_0: $\{0,1\}^n \times \mathbb{F}_{2^n} \times \{0,1\}^* \to \{0,1\}^{2\lambda}$, hash function for message pre-hashing, domain separation prefix is different among the parameter sets. The prefix is defined as follows.
  - AIMer-128f: `0x00`
  - AIMer-128s: `0x10`
  - AIMer-192f: `0x20`
  - AIMer-192s: `0x30`
  - AIMer-256f: `0x40`
  - AIMer-256s: `0x50`
- H_1: $\{0,1\}^{2\lambda} \times \{0,1\}^\lambda \times ((\{0,1\}^{2\lambda})^N \times \mathbb{F}_{2^n} \times (\mathbb{F}_{2^n})^\ell \times \mathbb{F}_{2^n})^\tau \to \{0,1\}^{2\lambda}$, hash function for generating challenge hash h_1, domain separation prefix is 1.
- H_2: $\{0,1\}^{2\lambda} \times \{0,1\}^\lambda \times ((\mathbb{F}_{2^n})^{(\ell+1)N} \times (\mathbb{F}_{2^n})^N)^\tau \to \{0,1\}^{2\lambda}$, hash function for generating challenge hash h_2, domain separation prefix is 2.
- H_3: $\{0,1\}^{2\lambda} \times \mathbb{F}_{2^n} \times \{0,1\}^\lambda \to \{0,1\}^\lambda \times (\{0,1\}^\lambda)^\tau$, hash function for generating salt and root seeds, domain separation prefix is 3.

- H_4: $\{0,1\}^\lambda \times [\tau] \times [N] \times \{0,1\}^\lambda \to (\{0,1\}^\lambda)^2$, hash function for expanding seed trees, domain separation prefix is 4.
- H_5: $\{0,1\}^\lambda \times [\tau] \times [N] \times \{0,1\}^\lambda \to \{0,1\}^{2\lambda} \times \mathbb{F}_{2^n} \times (\mathbb{F}_{2^n})^\ell \times \mathbb{F}_{2^n}^{\ell+1} \times \mathbb{F}_{2^n}$, hash function for committing and expanding seeds, domain separation prefix is 5.
- ExpandH1: $\{0,1\}^{2\lambda} \to ((\mathbb{F}_{2^n})^{\ell+1})^\tau$, hash function for expanding challenge hash h_1, no domain separation prefix.
- ExpandH2: $\{0,1\}^{2\lambda} \to [N]^\tau$, hash function for expanding challenge hash h_2, no domain separation prefix. Note that this function is the only one that uses integers as outputs. Following is the detailed definition.

  $$\mathsf{ExpandH2}(h_2) = (\bar{i}_0, \ldots, \bar{i}_{\tau-1})$$

  where

  $$\bar{i}_k = \mathsf{SHAKE}_\lambda(h_2, 8\tau)[8k : 8k+8] \bmod N$$

  for $k \in [\tau]$.

Note that ExpandH1 and ExpandH2 are not required to be domain-separated since they just expand the output of other hash functions.

4.1.3 GGM Tree Evaluation

In AIMer, GGM Tree [GGM86] is used to generate and publicize a set of seeds from a master seed while a punctured seed is unknown to the verifier. The GGM tree uses H_4 as an inner pseudorandom generator. The following three algorithms are used to evaluate the GGM tree.

- ExpandTree: $\{0,1\}^\lambda \times [\tau] \times \{0,1\}^\lambda \to (\{0,1\}^\lambda)^{2N-1}$, tree expanding algorithm with the salt, repetition index, and root seed.
- RevealAllBut: $(\{0,1\}^\lambda)^{2N-1} \times [N] \to (\{0,1\}^\lambda)^{\log N}$, algorithm for reveal all but one seeds.
- ReconstructTree: $\{0,1\}^\lambda \times (\{0,1\}^\lambda)^{\log N} \times [\tau] \times [N] \to (\{0,1\}^\lambda)^N$, recompute all but one seeds. The seed of challenged party is filled with dummy bits.

The detailed specifications are in Figure 2.

4.1.4 AIM3 Functions

AIM3 is the symmetric primitive which is zero-knowledge proved in AIMer. For ZKP to function efficiently, the input to all S-boxes must not be 0. For this reason, AIMer uses a modified version that outputs a $\perp$ (failure) instead of the original output ct if any S-box has an input of 0. Modified AIM3 is computed in AIM3_NoZeroInpSB algorithm for key generation, and computed in a secret-shared manner in the AIM3_MPC for signing and verification.

Generally, matrix multiplication can be performed more efficiently if the matrix is provided in its transposed form. Consequently, the AIM3_GenerateLinear algorithm in AIMer is deliberately designed to directly produce matrices in their transposed form.

ExpandTree(salt, k, seed)

Initialize tree: nodes $\leftarrow (0^\lambda)^{2N-1}$.
Set the root seed: nodes$[0] \leftarrow$ seed.
for $i = 1, \ldots, N-1$ **do**
    nodes$[2i-1]$, nodes$[2i] \leftarrow H_4(\text{salt}, k, i, \text{nodes}[i-1])$
Output nodes.

RevealAllBut(nodes, $\bar{i}$)

Initialize path: path $\leftarrow (0^\lambda)^{\log N}$.
$j \leftarrow N + \bar{i}$.
for $d = 0, \ldots, \log N - 1$ **do**
    Copy the sibling: path$[d] \leftarrow$ nodes$[(j \oplus 1) - 1]$
    Move to parent: $j \leftarrow \lfloor j/2 \rfloor$
Output path.

ReconstructTree(salt, path, k, $\bar{i}$)

Initialize tree: nodes $\leftarrow (0^\lambda)^{2N-1}$.
$j \leftarrow N + \bar{i}$.
for $d = 0, \ldots, \log N - 1$ **do**
    nodes$[(j \oplus 1) - 1] \leftarrow$ path$[d]$
    $j \leftarrow \lfloor j/2 \rfloor$
`// Expand the nodes, excluding the hidden ones`
for $h = 1, \ldots, \log N - 1$ **do**
    $j \leftarrow \lfloor (N + \bar{i})/2^{\log N - h} \rfloor$
    for $i = 2^h, \ldots, 2^{h+1} - 1$ **do**
        if $i \neq j$ **then**
            nodes$[2i-1]$, nodes$[2i] \leftarrow H_4(\text{salt}, k, i, \text{nodes}[i-1])$
Output nodes$[N-1 : 2N-1]$.

Figure 2: Algorithms for GGM tree evaluation.

- AIM3_GenerateLinear: $\{0,1\}^\lambda \to (\mathbb{F}_2^{n\times n})^{2\ell} \times (\mathbb{F}_2^n)^{\ell+1}$, generate the linear components in AIM3.
- AIM3_NoZeroInpSB: $\{0,1\}^n \times \{0,1\}^\lambda \to \{0,1\}^n \cup \perp$, the modified AIM3 one-way function that outputs $\perp$ if any S-box has an input of 0.
- AIM3_MPC: $(\mathbb{F}_2^{n\times n})^{2\ell} \times (\mathbb{F}_2^n)^{\ell+1} \times \mathbb{F}_{2^n} \times \mathbb{F}_{2^n} \times (\mathbb{F}_{2^n})^\ell \to (\mathbb{F}_{2^n} \times \mathbb{F}_{2^n} \times \mathbb{F}_{2^n})^{\ell+1}$, a function to generate multiplication check inputs from AIM3 MPC simulation.
- AIM3_SboxOutputs: $(\mathbb{F}_2^{n\times n})^{2\ell} \times (\mathbb{F}_2^n)^{\ell+1} \times \mathbb{F}_2^n \times \mathbb{F}_2^n \to (\mathbb{F}_{2^n})^{\ell+1}$, a function to generate outputs for first-round S-boxes.

Algorithm 4: AIM3_GenerateLinear(iv)

Initialize linear components:
  for $j \in [2\ell]$: $L_j \leftarrow 0^{n\times n}$, $U_j \leftarrow 0^{n\times n}$
  for $j \in [\ell+1]$: $b_j \leftarrow 0^n$
tape $\leftarrow$ SHAKE$_\lambda$(iv, $2\ell n^2 + (\ell+1)n$)
for $j \in [2\ell]$ **do**
  for $(r,c) \in [n]^2$ **do**
    if $c < r$ **then**
      Set $L_j[c][r] \leftarrow$ tape$[j \cdot n^2 + r \cdot n + c]$
    else if $c = r$ **then**
      Set $L_j[c][r] \leftarrow 1$
      Set $U_j[c][r] \leftarrow 1$
    else
      Set $U_j[c][r] \leftarrow$ tape$[j \cdot n^2 + r \cdot n + c]$
  Set $A_j \leftarrow L_j \cdot U_j$
for $j \in [\ell+1]$ **do**
  for $r \in [n]$ **do**
    Set $b_j[r] \leftarrow$ tape$[2\ell n^2 + j \cdot n + r]$.
Output $((A_j)_{j\in[2\ell]}, (b_j)_{j\in[\ell+1]})$

4.2 Signature Scheme

The AIMer signature scheme $\Pi = (\text{AIMer_keygen}, \text{AIMer_sign}, \text{AIMer_verify})$ consists of key generation, signing, and verification algorithms. Each algorithm calls the corresponding internal algorithms, AIMer_keygen_internal, AIMer_sign_internal, and AIMer_verify_internal. The internal algorithms are deterministic, and all the probabilistic steps are handled by external algorithms. The internal algorithms have been isolated for testing purposes. For the normal use of signature scheme, the internal functions should not be used without external functions.

- AIMer_keygen$(1^\lambda) \to (sk, pk)$: Continue sampling uniform random pt $\leftarrow_\$ \mathbb{F}_{2^n}$ and iv $\leftarrow_\$ \{0,1\}^n$ until ct $\leftarrow$ AIM3_NoZeroInpSB(pt, iv) makes ct $\neq \perp$. Set the

Algorithm 5: AIM3_NoZeroInpSB$(\mathsf{pt}, \mathsf{iv})$

Sample linear components: $((A_{\mathsf{iv},j})_{j\in[2\ell]}, (b_{\mathsf{iv},j})_{j\in[\ell+1]}) \leftarrow \mathsf{GenerateLinear}(\mathsf{iv})$
for $j \in [\ell]$ **do**
  Set $x_j \leftarrow A_{\mathsf{iv},j} \cdot \mathsf{pt} + b_{\mathsf{iv},j}$
  Set $y_j \leftarrow (x_j)^{2^{e_j}-1} + (x_j)^{2^n-2}$
Set $x_\ell \leftarrow b_{\mathsf{iv},\ell} + \sum_{j\in[\ell]} A_{\mathsf{iv},j+\ell} \cdot y_j$
Set $y_\ell \leftarrow (x_\ell)^{2^{e_\ell}-1} + (x_\ell)^{2^n-2}$
Set $\mathsf{ct} \leftarrow \mathsf{pt} + y_\ell$
for $j \in [\ell+1]$ **do**
  If $x_j = 0$ **then** Output $\perp$
Output ct

Algorithm 6: AIM3_MPC$\left(L = ((A_{\mathsf{iv},j})_{j\in[2\ell]}, (b_{\mathsf{iv},j})_{j\in[\ell+1]}), \mathsf{ct}, \mathsf{pt}^{(\cdot)}, (y_j^{(\cdot)})_{j\in[\ell]}, i\right)$

for $j \in [\ell]$ **do**
  Set $x_j^{(\cdot)} \leftarrow A_{\mathsf{iv},j} \cdot \mathsf{pt}^{(\cdot)}$
Set $x_\ell^{(\cdot)} \leftarrow \sum_{j\in[\ell]} A_{\mathsf{iv},j+\ell} \cdot y_j^{(\cdot)}$
Set $y_\ell^{(\cdot)} \leftarrow \mathsf{pt}^{(\cdot)}$
if $i = N-1$ **then**
  for $j \in [\ell+1]$ **do**
    $x_j^{(\cdot)} \leftarrow x_j^{(\cdot)} + b_{\mathsf{iv},j}$
  $y_\ell^{(\cdot)} \leftarrow y_\ell^{(\cdot)} + \mathsf{ct}$
for $j \in [\ell+1]$ **do**
  Set $z_j^{(\cdot)} \leftarrow (x_j^{(\cdot)})^{2^{e_j}}$
  if $i = N-1$ **then**
    $z_j^{(\cdot)} \leftarrow z_j^{(\cdot)} + 1$
Output $(x_j^{(\cdot)}, y_j^{(\cdot)}, z_j^{(\cdot)})_{j\in[\ell+1]}$

Algorithm 7: AIM3_SboxOutputs$(L = ((A_{\mathsf{iv},j})_{j\in[2\ell]}, (b_{\mathsf{iv},j})_{j\in[\ell+1]}), \mathsf{pt}, \mathsf{ct})$

for $j \in [\ell]$ **do**
  Set $x_j \leftarrow A_{\mathsf{iv},j} \cdot \mathsf{pt} + b_{\mathsf{iv},j}$
  Set $y_j \leftarrow (x_j)^{2^{e_j}-1} + (x_j)^{2^n-2}$
Set $y_\ell \leftarrow \mathsf{pt} + \mathsf{ct}$
Output $(y_j)_{j\in[\ell+1]}$

public key $pk \leftarrow (\mathsf{iv}, \mathsf{ct}) \in \{0,1\}^n \times \mathbb{F}_{2^n}$ and the private key $sk \leftarrow (\mathsf{pt}, \mathsf{iv}, \mathsf{ct}) \in \mathbb{F}_{2^n} \times \{0,1\}^n \times \mathbb{F}_{2^n}$.

- AIMer_sign$(sk, M, \mathsf{ctx}) \rightarrow \sigma$: Take as input a private key $sk = (\mathsf{pt}, \mathsf{iv}, \mathsf{ct})$, a message $m \in \{0,1\}^*$, and a context string ctx, and compute the zero-knowledge proof π for the AIM3 one-way function circuit using m as a part of the input to the challenge hash as described in Algorithm 10. Output the corresponding signature $\sigma \leftarrow \pi$ where $|\sigma| = (5 + (\log_2 N + 2\ell + 5)\tau)\lambda$
- AIMer_verify$(pk, M, \sigma, \mathsf{ctx}) \rightarrow$ Accept or Reject : Take as input a public key $pk = (\mathsf{iv}, \mathsf{ct})$, a message m, a signature σ, and a context string ctx, and conduct the verification of NIZKPoK for the AIM3 one-way function circuit as described in Algorithm 12. Output either Accept or Reject according to the verification result of the ZKP.

The context string should be no longer than 255 bytes, and bit-length of it should be divisible by 8. If nothing is inputted in the context string, then it is the empty string by default. Each algorithm will be described in detail in the following sections.

4.2.1 Key Generation

The key generation algorithm AIMer_keygen(1^λ) is initiated by generating two random λ-bit sequences, pt and iv. The secret key pt is encrypted under the AIM3 function using iv as the initialization vector, resulting in the ciphertext ct. If any S-box has an input of 0, do above steps again. Consequently, the algorithm sets the secret key sk as a tuple comprising pt, iv, and ct, and constructs the public key pk as a tuple comprising iv and ct. The final output of the algorithm is the key pair (sk, pk) of the AIMer signature scheme.

Algorithm 8: AIMer_keygen(1^λ) - AIMer signature scheme, key generation algorithm

```
Set ct ← ⊥
while ct = ⊥ do
    Sample pt ←$ {0,1}^n
    Sample iv ←$ {0,1}^λ
    if pt = null or iv = null then
        Abort
    Set ct ← AIMer_keygen_internal(pt, iv)
Set sk ← (pt, iv, ct), pk ← (iv, ct)
return (sk, pk)
```

4.2.2 Signature Generation

The signing algorithm consists of five phases as commented in Algorithm 10.

Algorithm 9: AIMer_keygen_internal(pt, iv) - AIMer signature scheme, internal key generation algorithm

Set ct $\leftarrow$ AIM3_NoZeroInpSB(pt, iv)
Output ct

PHASE 1: COMMITTING TO THE SEEDS AND THE EXECUTION VIEWS OF THE PARTIES. It first pre-hashes the message, and computes an instance of AIM3 using the initial vector. Next, together with sampling per-signature randomness, it generates the salt and the root seeds. After that, for each parallel execution, it does the following.

1. It computes the parties' seeds as leaves of a binary tree from the root seed of each repetition.
2. It commits to each party's seed and expands random tape.
3. It prepares for the multi-party computation among the N parties using the parties' seeds, by generating secret shares of the multiplication triples for each S-box.

PHASE 2: CHALLENGING THE CHECKING PROTOCOL. It then computes the first challenge hash and expands it to the first challenge for the multiplication checking protocol in BN++.

PHASE 3: COMMITTING TO THE SIMULATION OF THE CHECKING PROTOCOL. It computes and outputs the broadcast values for the multiplication checking protocol of BN++.

PHASE 4: CHALLENGING THE VIEWS OF THE MPC PROTOCOL. It computes the second challenge hash and expands it to the second challenge for choosing unopened views.

PHASE 5: OPENING THE VIEWS OF THE MPC AND CHECKING PROTOCOLS. It collects the seeds to open the views of $N - 1$ parties for each repetition, and outputs a signature.

Algorithm 10: AIMer_sign(sk, M, ctx) - AIMer signature scheme, signing algorithm.

if |ctx| > 2040 *or* |ctx| $\bmod 8 \neq 0$ **then**
    Abort
Sample randomness: $\rho \leftarrow_{\$} \{0,1\}^{\lambda}$ ($\rho \leftarrow 0^{\lambda}$ for deterministic signature)
if $\rho =$ `null` **then**
    Abort
`// Byte-length of ctx is represented in a byte`
$M' \leftarrow (|\text{ctx}|/8)\|\text{ctx}\|M$
return AIMer_sign_internal(sk, M', ρ)

Algorithm 11: AIMer_sign_internal$(sk = (\mathsf{pt}, \mathsf{iv}, \mathsf{ct}), M', \rho)$ - AIMer signature scheme, internal signing algorithm.

// Phase 1: Committing to the seeds and the execution views.

Compute the hash of the message: $\mu \leftarrow H_0(\mathsf{iv}, \mathsf{ct}, M')$

Derive the linear components: $L \leftarrow \mathsf{AIM3_GenerateLinear}(\mathsf{iv})$

Compute the S-boxes' outputs: $(y_j)_{j\in[\ell+1]} \leftarrow \mathsf{AIM3_SboxOutputs}(L, \mathsf{pt}, \mathsf{ct})$

Compute salt and root seeds: $(\mathsf{salt}, (\mathsf{seed}_k)_{k\in[\tau]}) \leftarrow H_3(\mu, \mathsf{pt}, \rho)$

for *each repetition* $k \in [\tau]$ **do**

  Compute parties' tape and commit to it:

    $\mathsf{nodes}_k \leftarrow \mathsf{ExpandTree}(\mathsf{salt}, k, \mathsf{seed}_k)$

    for *each party* $i \in [N]$ **do**

      $\mathsf{seed}_k^{(i)} \leftarrow \mathsf{nodes}_k[N-1+i]$

      $(\mathsf{com}_k^{(i)}, \mathsf{pt}_k^{(i)}, (y_{k,j}^{(i)})_{j\in[\ell]}, (a_{k,j}^{(i)})_{j\in[\ell+1]}, c_k^{(i)}) \leftarrow H_5(\mathsf{salt}, k, i, \mathsf{seed}_k^{(i)})$

  Compute offsets and adjust last shares for witness shares:

    $\Delta\mathsf{pt}_k \leftarrow \mathsf{pt} - \sum_i \mathsf{pt}_k^{(i)}$, $\mathsf{pt}_k^{(N-1)} \leftarrow \mathsf{pt}_k^{(N-1)} + \Delta\mathsf{pt}_k$

    for $j \in [\ell]$, $\Delta y_{k,j} \leftarrow y_j - \sum_i y_{k,j}^{(i)}$, $y_{k,j}^{(N-1)} \leftarrow y_{k,j}^{(N-1)} + \Delta y_{k,j}$

    $\Delta c_k \leftarrow \sum_{j\in[\ell+1]}(\sum_i a_{k,j}^{(i)} \cdot y_j) - \sum_i c_k^{(i)}$, $c_k^{(N-1)} \leftarrow c_k^{(N-1)} + \Delta c_k$.

  for *each party* $i \in [N]$ **do**

    Prepare the multiplication check inputs by MPC simulation:

      $(x_{k,j}^{(i)}, y_{k,j}^{(i)}, z_{k,j}^{(i)})_{j\in[\ell+1]} \leftarrow \mathsf{AIM3_MPC}(L, \mathsf{ct}, \mathsf{pt}_k^{(i)}, (y_{k,j}^{(i)})_{j\in[\ell]}, i)$

Set $\sigma_1 \leftarrow (\mathsf{salt}, ((\mathsf{com}_k^{(i)})_{i\in[N]}, \Delta\mathsf{pt}_k, (\Delta y_{k,j})_{j\in[\ell]}, \Delta c_k)_{k\in[\tau]})$

// Phase 2: Challenging the multiplication checking protocol.

Compute challenge hash: $h_1 \leftarrow H_1(\mu, \sigma_1)$

Expand hash: $((\epsilon_{k,j})_{j\in[\ell+1]})_{k\in[\tau]} \leftarrow \mathsf{ExpandH1}(h_1)$ where $\epsilon_{k,j} \in \mathbb{F}_{2^n}$

// Phase 3: Committing to the multiplication check results.

for *each repetition* $k \in [\tau]$ **do**

  Simulate the multiplication checking protocol as in Section 2.3:

    for $(i,j) \in [N] \times [\ell+1]$, $\alpha_{k,j}^{(i)} \leftarrow a_{k,j}^{(i)} + x_{k,j}^{(i)} \cdot \epsilon_{k,j}$

    for $j \in [\ell+1]$, Set $\alpha_{k,j} = \sum_{i\in[N]} \alpha_{k,j}^{(i)}$

    for $i \in [N]$, $v_k^{(i)} \leftarrow c_k^{(i)} + \sum_{j\in[\ell+1]}(z_{k,j}^{(i)} \cdot \epsilon_{k,j} - \alpha_{k,j} \cdot y_{k,j}^{(i)})$

Set $\sigma_2 \leftarrow (\mathsf{salt}, ((\alpha_{k,0}^{(i)}, \ldots, \alpha_{k,\ell}^{(i)})_{i\in[N]}, (v_k^{(i)})_{i\in[N]})_{k\in[\tau]})$

// Phase 4: Challenging the views of the MPC protocol.

Compute challenge hash: $h_2 \leftarrow H_2(h_1, \sigma_2)$

Expand hash: $(\bar{i}_k)_{k\in[\tau]} \leftarrow \mathsf{ExpandH2}(h_2)$ where $\bar{i}_k \in [N]$

// Phase 5: Opening the execution views.

for *each repetition* $k \in [\tau]$ **do**

  $\mathsf{path}_k \leftarrow \mathsf{RevealAllBut}(\mathsf{nodes}_k, \bar{i}_k)$

Output

$\sigma \leftarrow \left(\mathsf{salt}, h_1, h_2, (\mathsf{path}_k, \mathsf{com}_k^{(\bar{i}_k)}, \Delta\mathsf{pt}_k, (\Delta y_{k,j})_{j\in[\ell]}, \Delta c_k, (\alpha_{k,j}^{(\bar{i}_k)})_{j\in[\ell+1]})_{k\in[\tau]}\right)$

4.2.3 Signature Verification

The verification algorithm takes as input $(pk = (\mathsf{iv}, \mathsf{ct}), m, \sigma)$, and outputs Accept or Reject. We refer to Algorithm 12 for the detailed description.

First, given a public key, it computes the hash value of the message and an instance of AIM3. From the signature, it expands hash values to obtain the challenges in Phase 2 and 4 of the signing algorithm.

RECOMPUTATION OF PHASE 1 AND 2. It does the following for each parallel repetition:

- Recomputes random seeds for disclosed parties, and regenerate commitments and tapes.
- From the commitments and tapes, recomputes σ_1 and the first challenge hash.

RECOMPUTATION OF PHASE 3 AND 4. For each parallel repetition, it simulates the multiplication checking protocol for each disclosed party. It recomputes the broadcast values for each disclosed party. Also, it computes the remaining share of the broadcast value $v_k^{(\bar{i}_k)}$. Finally, it recomputes σ_2 and the challenge hash.

COMPARISON OF THE HASH VALUES. It compares the hash values in the input signature and those obtained from the recomputation. It outputs Accept only if both hash values agree, and outputs Reject otherwise.

Algorithm 12: AIMer_verify$(pk = (\mathsf{iv}, \mathsf{ct}), M, \sigma, \mathsf{ctx})$ - AIMer signature scheme, verification algorithm.

if $|\mathsf{ctx}| > 2040$ *or* $|\mathsf{ctx}| \bmod 8 \neq 0$ **then**
  Abort.
$M' \leftarrow (|\mathsf{ctx}|/8) \| \mathsf{ctx} \| M$
`// Byte-length of ctx is represented in a byte.`
return AIMer_verify_internal(pk, M', σ)

4.3 Recommended Parameters

For security levels L1, L3, and L5, recommended sets of parameters are given in Table 3. For each value of security parameter λ, the corresponding sets of parameters are expected to provide λ-bit security against all classical attacks, and $\lambda/2$-bit security against quantum attacks.

5 Formal Security Analysis

5.1 EUF-CMA Security of AIMer in the Random Oracle Model

In this section, we prove the EUF-CMA (existential unforgeability under adaptive chosen-message attacks [GMR88]) security of AIMer. To prove the EUF-CMA secu-

Algorithm 13: AIMer_verify_internal$(pk = (\mathsf{iv}, \mathsf{ct}), M', \sigma)$ - AIMer signature scheme, internal verification algorithm.

Parse σ as

$$\left(\mathsf{salt}, h_1, h_2, \left(\mathsf{path}_k, \mathsf{com}_k^{(\bar{i}_k)}, \Delta\mathsf{pt}_k, (\Delta y_{k,j})_{j\in[\ell]}, \Delta c_k, (\alpha_{k,j}^{(\bar{i}_k)})_{j\in[\ell+1]}\right)_{k\in[\tau]}\right)$$

Compute the hash value of the message: $\mu \leftarrow H_0(\mathsf{iv}, \mathsf{ct}, M')$

Derive the linear components: $L \leftarrow$ AIM3_GenerateLinear(iv)

Expand hashes:

$((\epsilon_{k,j})_{j\in[\ell+1]})_{k\in[\tau]} \leftarrow$ ExpandH1(h_1) and $(\bar{i}_k)_{k\in[\tau]} \leftarrow$ ExpandH2(h_2)

for *each repetition* $k \in [\tau]$ **do**

  Compute seeds except challenged one:

  $(\mathsf{seed}_k^{(0)}, \ldots, \mathsf{seed}_k^{(N-1)}) \leftarrow$ ReconstructTree$(\mathsf{salt}, \mathsf{path_k}, k, \bar{i}_k)$

  for *each party* $i \in [N] \setminus \{\bar{i}_k\}$ **do**

    Recompute

    $(\mathsf{com}_k^{(i)}, \mathsf{pt}_k^{(i)}, (y_{k,j}^{(i)})_{j\in[\ell]}, (a_{k,j}^{(i)})_{j\in[\ell+1]}, c_k^{(i)}) \leftarrow H_5(\mathsf{salt}, k, i, \mathsf{seed}_k^{(i)})$

    if $i = N-1$ **then**

      Adjust last share:

      $\mathsf{pt}_k^{(i)} \leftarrow \mathsf{pt}_k^{(i)} + \Delta\mathsf{pt}_k$

      for $j \in [\ell]$, $y_{k,j}^{(i)} \leftarrow y_{k,j}^{(i)} + \Delta y_{k,j}$

      $c_k^{(i)} \leftarrow c_k^{(i)} + \Delta c_k$

    Prepare the multiplication check inputs by MPC simulation:

    $(x_{k,j}^{(i)}, y_{k,j}^{(i)}, z_{k,j}^{(i)})_{j\in[\ell+1]} \leftarrow$ AIM3_MPC$(L, \mathsf{ct}, \mathsf{pt}_k^{(i)}, (y_{k,j}^{(i)})_{j\in[\ell]}, i)$

    Simulate the multiplication checking protocol as in Section 2.3:

    for $(i,j) \in ([N] \setminus \{\bar{i}_k\}) \times [\ell+1]$, $\alpha_{k,j}^{(i)} \leftarrow a_{k,j}^{(i)} + x_{k,j}^{(i)} \cdot \epsilon_{k,j}$

    for $j \in [\ell+1]$, Set $\alpha_{k,j} = \sum_{i\in[N]} \alpha_{k,j}^{(i)}$

    for $i \in [N] \setminus \{\bar{i}_k\}$, $v_k^{(i)} \leftarrow c_k^{(i)} + \sum_{j\in[\ell+1]} (z_{k,j}^{(i)} \cdot \epsilon_{k,j} - \alpha_{k,j} \cdot y_{k,j}^{(i)})$

    Set $v_k^{(\bar{i}_k)} = 0 - \sum_{i\in[N]\setminus\{\bar{i}_k\}} v_k^{(i)}$

Set $\sigma_1 \leftarrow \left(\mathsf{salt}, \left((\mathsf{com}_k^{(i)})_{i\in[N]}, \Delta\mathsf{pt}_k, (\Delta y_{k,j})_{j\in[\ell]}, \Delta c_k\right)_{k\in[\tau]}\right)$.

Set $h_1' \leftarrow H_1(\mu, \sigma_1)$.

Set $\sigma_2 \leftarrow \left(\mathsf{salt}, ((\alpha_{k,0}^{(i)}, \ldots, \alpha_{k,\ell}^{(i)})_{i\in[N]}, (v_k^{(i)})_{i\in[N]})_{k\in[\tau]}\right)$

Set $h_2' = H_2(h_1', \sigma_2)$.

Output Accept if $h_1 = h_1'$ and $h_2 = h_2'$.

Otherwise, output Reject.

Security	Parameters	λ	n	ℓ	e_0	e_1	e_2	e_3	Hash	N	τ
L1	AIMer-128f	128	128	2	3	7	11	-	SHAKE128	16	33
	AIMer-128s	128	128	2	3	7	11	-	SHAKE128	256	17
L3	AIMer-192f	192	192	2	11	23	47	-	SHAKE256	16	49
	AIMer-192s	192	192	2	11	23	47	-	SHAKE256	256	25
L5	AIMer-256f	256	256	3	3	7	11	19	SHAKE256	16	65
	AIMer-256s	256	256	3	3	7	11	19	SHAKE256	256	33

Table 3: The recommended parameters for AIMer.

rity, we first show that AIMer is secure against key-only attack (EUF-KO) in Theorem 1, where an adversary is given the public key and no access to the signing oracle. Then, we show that AIMer is EUF-CMA secure by proving that the signing can be simulated without using the secret key in Theorem 2. In our security proof, we followed the same arguments as the security proof of BN++ in [KZ22].

Theorem 1 (EUF-KO Security of AIMer)**.** *Assume that H_0, H_1, H_2, H_4, H_5,* ExpandH1*, and* ExpandH2 *be modeled as random oracles, and let (N, τ, λ) be parameters of the* AIMer *signature scheme. Let $\mathcal{A}$ be a probabilistic polynomial-time (PPT) adversary against the EUF-KO security of* AIMer *that makes a total of Q random oracle queries. There exists a PPT adversary $\mathcal{B}$ such that*

$$\mathbf{Adv}^{\mathsf{euf\text{-}ko}}_{\mathsf{AIMer}}(\mathcal{A}) \leq \frac{(\tau N+1)Q^2}{2^{2\lambda}} + \Pr[X+Y=\tau] + \mathbf{Adv}^{\mathsf{owf}}_{\mathsf{AIM3}}(\mathcal{B}),$$

where $\Pr[X+Y=\tau]$ is as described in the proof.

Proof. We build an algorithm $\mathcal{B}$ that retrieves a pre-image for the one-way function AIM3 using the EUF-KO adversary $\mathcal{A}$ as a subroutine. Suppose that all the queries to H_1, H_2 and H_5 are listed in $\mathcal{Q}_1$, $\mathcal{Q}_2$ and $\mathcal{Q}_5$, respectively.

Algorithm $\mathcal{B}$ takes the AIM3 one-way function value $(\mathsf{iv}, \mathsf{ct})$ as an input, and forwards it to $\mathcal{A}$ as an AIMer public key for the EUF-KO game. $\mathcal{B}$ manages a set Bad to keep track of all the answers from the three random oracles and two tables $\mathcal{T}_{\mathsf{sh}}$ and $\mathcal{T}_{\mathsf{in}}$ to record the values derived from $\mathcal{A}$'s RO queries as follows:

- $\mathcal{T}_{\mathsf{sh}}$ to store secret shares of the parties, and
- $\mathcal{T}_{\mathsf{in}}$ to store inputs to the MPC protocol.

We also program the random oracles for $\mathcal{A}$ as follows.

- H_1 : When $\mathcal{A}$ commits to seeds and sends the offsets for the preimage pt which is the secret key and the multiplication triples, $\mathcal{B}$ checks the query list $\mathcal{Q}_5$ to see if the commitments were output by its simulation of H_5. If $\mathcal{B}$ finds matching results for all i's in some repetition k, then it can recover pt. See Algorithm 14.
- H_2 : See Algorithm 15.

- H_5 : When $\mathcal{A}$ queries random oracle for H_5, $\mathcal{B}$ records the query to match the commitments and expanded random tape with its corresponding seeds. See Algorithm 16.

- H_0, H_4, ExpandH1 and ExpandH2 are not programmed.

After $\mathcal{A}$ terminates, $\mathcal{B}$ checks whether there is $\mathsf{pt}_k \in \mathcal{T}_{\mathsf{in}}$ satisfying $\mathsf{AIM3}(\mathsf{pt}_k, \mathsf{iv}) = \mathsf{ct}$. If $\mathcal{B}$ finds a match pt_k, $\mathcal{B}$ outputs it as a pre-image of AIM3, otherwise $\mathcal{B}$ outputs $\perp$.

Given the algorithm of $\mathcal{B}$ as above, the probability that $\mathcal{A}$ wins is bounded as below.

$$\begin{aligned}\Pr[\mathcal{A}\text{ wins}] &= \Pr[\mathcal{A}\text{ wins} \wedge \mathcal{B}\text{ aborts}] + \Pr[\mathcal{A}\text{ wins} \wedge \mathcal{B}\text{ outputs }\perp] \\ &\quad + \Pr[\mathcal{A}\text{ wins} \wedge \mathcal{B}\text{ outputs pt}] \\ &\leq \Pr[\mathcal{B}\text{ aborts}] + \Pr[\mathcal{A}\text{ wins} \mid \mathcal{B}\text{ outputs }\perp] + \Pr[\mathcal{B}\text{ outputs pt}] \qquad (4)\end{aligned}$$

We define Q_1, Q_2 and Q_5 as the number of queries made by $\mathcal{A}$ to random oracles H_1, H_2 and H_5, respectively. Then we can bound the probability that $\mathcal{B}$ aborts (The first term on the RHS of (4)) as follows.

$$\begin{aligned}\Pr[\mathcal{B}\text{ aborts}] &= (\#\text{times an } r \text{ is sampled}) \cdot \Pr[\mathcal{B}\text{ aborts at that sample}] \\ &\leq (Q_1 + Q_2 + Q_5) \cdot \frac{\max|\mathsf{Bad}|}{2^{2\lambda}} \\ &= (Q_1 + Q_2 + Q_5) \cdot \frac{(\tau N + 1)Q_1 + 2Q_2 + Q_5}{2^{2\lambda}} \\ &\leq \frac{(\tau N + 1)(Q_1 + Q_2 + Q_5)^2}{2^{2\lambda}} \leq \frac{(\tau N + 1)Q^2}{2^{2\lambda}}. \qquad (5)\end{aligned}$$

We now analyze $\Pr[\mathcal{A}\text{ wins} \mid \mathcal{B}\text{ outputs }\perp]$ (the second term in the RHS of (4)), which means that pt corresponding to $(\mathsf{iv}, \mathsf{ct})$ is not found. We parse it into two cases, which correspond to cheating in the first and second rounds, respectively.

CHEATING IN THE FIRST ROUND. Let $q_1 \in \mathcal{Q}_1$ be a query to H_1, and $h_1 = ((\epsilon_{k,j})_{j\in[\ell+1]})_{k\in[\tau]}$ be its corresponding answer. We collect the set of indices $k \in [\tau]$ representing "good executions" such that $\mathcal{T}_{\mathsf{in}}[q_1, k]$ is not empty and $v_k = 0$, say $G_1(q_1, h_1)$. For $k \in G_1(q_1, h_1)$, the challenges $(\epsilon_{k,j})_{j\in[\ell+1]}$ were sampled so that the multiplication check protocol presented in Section 2.3 is passed in this repetition. According to Lemma 1, if the secret shared inputs contain an incorrect multiplication triple, since h_1 is sampled uniformly at random, this happens with probability at most $1/2^\lambda$.

Lemma 1. *If the secret-shared input $(x_j, y_j, z_j)_{j\in[C]}$ contains an incorrect multiplication triple, or if the shares of $((a_j, y_j)_{j\in[C]}, c)$ form an incorrect dot product, then the parties output* `Accept` *in the subprotocol with probability at most $1/2^\lambda$.*

Algorithm 14: $H_1(q_1 = (\mu, \sigma_1))$:

Parse σ_1 as $\left(\mathsf{salt}, ((\mathsf{com}_k^{(i)})_{i\in[N]}, \Delta\mathsf{pt}_k, (\Delta y_{k,j})_{j\in[\ell]}, \Delta c_k)_{k\in[\tau]}\right)$.
for $k \in [\tau], i \in [N]$ **do**
  $\mathsf{com}_k^{(i)} \rightarrow \mathsf{Bad}$.
// If the committed seed is known for some k and i, then $\mathcal{B}$ records the shares of the secret key and the views of the parties, derived from that seed and the offsets in σ_1.
for $k \in [\tau], i \in [N]$ **do**
  if $\exists \mathsf{seed}_k^{(i)} : ((\mathsf{salt}, k, i, \mathsf{seed}_k^{(i)}), \mathsf{com}_k^{(i)}, \mathsf{pt}_k^{(i)}, (y_{k,j}^{(i)})_{j\in[\ell]}, (a_{k,j}^{(i)})_{j\in[\ell+1]}, c_k^{(i)}) \in \mathcal{Q}_5$ **then**
    if $i = N - 1$ **then**
      $\mathsf{pt}_k^{(i)} \leftarrow \mathsf{pt}_k^{(i)} + \Delta\mathsf{pt}_k, (y_{k,j}^{(i)} \leftarrow y_{k,j}^{(i)} + \Delta y_{k,j})_{j\in[\ell]}$ and $c_k^{(i)} \leftarrow c_k^{(i)} + \Delta c_k$
    $(\mathsf{pt}_k^{(i)}, (y_{k,j}^{(i)})_{j\in[\ell]}, (a_{k,j})_{j\in[\ell+1]}^{(i)}, c_k^{(i)}) \rightarrow \mathcal{T}_{\mathsf{sh}}[q_1, k, i]$
// If the shares of the various elements are known for every party in that repetition, $\mathcal{B}$ records the resulting secret key, multiplication inputs and S-box outputs.
for *each* $k : \forall i, \mathcal{T}_{\mathsf{sh}}[q_1, k, i] \neq \emptyset$ **do**
  $\mathsf{pt}_k \leftarrow \sum_i \mathsf{pt}_k^{(i)}, (y_{k,j} \leftarrow \sum_i y_{k,j}^{(i)})_{j\in[\ell]}, (a_{k,j} \leftarrow \sum_i a_{k,j}^{(i)})_{j\in[\ell+1]}, c_k \leftarrow \sum_i c_k^{(i)}$.
  $y_{k,\ell} \leftarrow \mathsf{pt}_k + \mathsf{ct}$
  for $j \in [\ell]$ **do**
    Set $x_{k,j} \leftarrow A_{\mathsf{iv},j} \cdot \mathsf{pt}_k + b_{\mathsf{iv},j}$ and $z_{k,j} \leftarrow (x_{k,j})^{2^{e_j}} + 1$.
  Set $x_{k,\ell} \leftarrow b_{\mathsf{iv},\ell} + \sum_{j\in[\ell]} A_{\mathsf{iv},j+\ell} \cdot y_{k,j}$ and $z_{k,\ell} \leftarrow (x_{k,\ell})^{2^{e_\ell}} + 1$.
  $\mathsf{pt}_k \rightarrow \mathcal{T}_{\mathsf{in}}[q_1, k]$.
$r \leftarrow_{\$} \{0,1\}^{2\lambda}$.
if $r \in \mathsf{Bad}$ **then**
  abort.
$r \rightarrow \mathsf{Bad}$.
$(q_1, r) \rightarrow \mathcal{Q}_1$.
// Compute the multiplication check protocol values.
$((\epsilon_{k,j})_{j\in[\ell+1]})_{k\in[\tau]} \leftarrow \mathsf{ExpandH1}(r)$.
for *each* $k : \mathcal{T}_{\mathsf{in}}[q_1, k] \neq \emptyset$ **do**
  for $j \in [\ell + 1]$, $\alpha_{k,j} = a_{k,j} + \epsilon_{k,j} \cdot x_{k,j}$.
  $v_k = c_k + \sum_{j\in[\ell+1]}(\epsilon_{k,j} \cdot z_{k,j} - \alpha_{k,j} \cdot y_{k,j})$.
Return r.

Algorithm 15: $H_2(q_2 = (h_1, \sigma_2))$:

$h_1 \to \mathsf{Bad}$.
$r \leftarrow_\$ \{0,1\}^{2\lambda}$.
if $r \in \mathsf{Bad}$ **then**
  abort.
$r \to \mathsf{Bad}$.
$(q_2, r) \to \mathcal{Q}_2$.
Return r.

Algorithm 16: $H_5(q_5 = (\mathsf{salt}, k, i, \mathsf{seed}))$:

$r \leftarrow_\$ \{0,1\}^{2\lambda}$.
if $r \in \mathsf{Bad}$ **then**
  abort.
$r \to \mathsf{Bad}$.
$\left(\mathsf{pt}_k^{(i)}, (y_{k,j}^{(i)})_{j\in[\ell]}, (a_{k,j}^{(i)})_{j\in[\ell+1]}, c_k^{(i)}\right) \leftarrow_\$ \mathbb{F}_{2^n} \times (\mathbb{F}_{2^n})^\ell \times (\mathbb{F}_{2^n})^{\ell+1} \times \mathbb{F}_{2^n}$
$\left(q_5, r, \mathsf{pt}_k^{(i)}, (y_{k,j}^{(i)})_{j\in[\ell]}, (a_{k,j}^{(i)})_{j\in[\ell+1]}, c_k^{(i)}\right) \to \mathcal{Q}_5$.
Return $\left(r, \mathsf{pt}_k^{(i)}, (y_{k,j}^{(i)})_{j\in[\ell]}, (a_{k,j}^{(i)})_{j\in[\ell+1]}, c_k^{(i)}\right)$.

Proof. Let $\Delta z_j = z_j - x_j \cdot y_j$ and $\Delta c = -\sum_{j\in[C]} a_j \cdot y_j + c$. Then,

$$\begin{aligned} v &= \sum_{j\in[C]} (\epsilon_j \cdot z_j - \alpha_j \cdot y_j) + c \\ &= \sum_{j\in[C]} \epsilon_j \cdot z_j - \sum_{j\in[C]} \epsilon_j \cdot x_j \cdot y_j - \sum_{j\in[C]} a_j \cdot y_j + c \\ &= \sum_{j\in[C]} \epsilon_j \cdot (z_j - x_j \cdot y_j) - \sum_{j\in[C]} a_j \cdot y_j + c \\ &= \sum_{j\in[C]} \epsilon_j \cdot \Delta z_j + \Delta c. \end{aligned}$$

Define a multivariate polynomial

$$Q(X_0, \ldots, X_{C-1}) = X_0 \cdot \Delta z_0 + \cdots + X_{C-1} \cdot \Delta z_{C-1} + \Delta c$$

over $\mathbb{F}_{2^n}$ and note that $v = 0$ if and only if $Q(\epsilon_0, \ldots, \epsilon_{C-1}) = 0$. In the case of a cheating prover, Q is nonzero, and by the multivariate version of the Schwartz-Zippel lemma, the probability that $Q(\epsilon_0, \ldots, \epsilon_{C-1}) = 0$ is at most $1/2^\lambda$, since Q has total degree 1 and $(\epsilon_0, \ldots, \epsilon_{C-1})$ is chosen uniformly at random. □

Given $\mathcal{B}$ outputs $\perp$, the number of elements $\#G_1(q_1, h_1)|_\perp \sim X_{q_1}$ where $X_{q_1} = \mathcal{B}(\tau, p_1)$, where $\mathcal{B}(\tau, p_1)$ is the binomial distribution with τ events, each with success probability $p_1 = 1/2^\lambda$. We select the query-response pair $(q_{\mathsf{best}_1}, h_{\mathsf{best}_1})$ such

that $\#G_1(q_1, h_1)$ is the maximum. Then, the following holds.

$$\#G_1(q_{\mathsf{best}_1}, h_{\mathsf{best}_1})|_\perp \sim X = \max_{q_1 \in \mathcal{Q}_1}\{X_{q_1}\}.$$

CHEATING IN THE SECOND ROUND. Let $q_2 = (h_1, \sigma_2)$ be a query to H_2. Note that q_2 can only be used in the winning EUF-KO game when the corresponding $(q_1, h_1) \in \mathcal{Q}_1$ exists. For the bad repetition $k \in [\tau] \backslash G_1(q_1, h_1)$, either $\mathcal{T}_{\mathsf{in}}[q_1, k]$ is empty (which means verification fails so that $\mathcal{A}$ loses) or $v_k \neq 0$ but the verification passes. Hence, it should be the case that one of the N parties cheated. Since $h_2 = (\bar{i}_k)_{k\in[\tau]} \in [N]^\tau$ is distributed uniformly at random, the probability that one of the N parties has cheated for all bad executions k is

$$\left(\frac{1}{N}\right)^{\tau - \#G_1(q_1,h_1)} \leq \left(\frac{1}{N}\right)^{\tau - \#G_1(q_{\mathsf{best}_1}, h_{\mathsf{best}_1})}.$$

To sum up, we can analyze the probability that $\mathcal{A}$ wins conditioning on $\mathcal{B}$ outputting $\perp$ is

$$\Pr[\mathcal{A} \text{ wins} \mid \mathcal{B} \text{ outputs } \perp] \leq \Pr[X + Y = \tau], \tag{6}$$

where X is as before, and $Y = \max_{q_2 \in \mathcal{Q}_2}\{Y_{q_2}\}$ where Y_{q_2} variables are independently and identically distributed as $\mathcal{B}(\tau - X, 1/N)$.

Finally, combining (4), (5) and (6) all together, we obtain the following.

$$\Pr[\mathcal{A} \text{ wins}] \leq \frac{(\tau N + 1) \cdot Q^2}{2^{2\lambda}} + \Pr[X + Y = \tau] + \Pr[\mathcal{B} \text{ outputs } \mathsf{pt}],$$

where X and Y are defined as above. Setting AIM3 as a secure OWF, we achieve (1) as desired.

□

Theorem 2 (EUF-CMA Security of AIMer)**.** *Assume that H_0, H_1, H_2, H_4, H_5,* ExpandH1*, and* ExpandH2 *are modeled as random oracles and that the (N, τ, λ) parameters of* AIMer *are appropriately chosen. For a PPT adversary $\mathcal{A}$ against the EUF-CMA security of* AIMer *with total Q_{sig} signing oracle queries and Q random oracle queries, there exist a PPT adversary $\mathcal{B}$ against the EUF-KO security of* AIMer *(with same number of queries to random oracles) and a PPT adversary $\mathcal{C}$ against the PRF security of H_3*[2] *such that*

$$\begin{aligned}\mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{AIMer}}(\mathcal{A}) \leq\ & Q_{\mathrm{sig}} \cdot \mathbf{Adv}^{\mathsf{prf}}_{H_3}(\mathcal{C}) + 2(\tau + 1)\log N \cdot \frac{(Q_{\mathrm{sig}} + Q)^2}{2^{2\lambda}} \\ & + \mathbf{Adv}^{\mathsf{euf\text{-}ko}}_{\mathsf{AIMer}}(\mathcal{B}).\end{aligned}$$

Proof. Let $\mathcal{A}$ be an EUF-CMA adversary against AIMer for given $(\mathsf{iv}, \mathsf{ct})$. Let G_0 be the original EUF-CMA game. Let $\mathcal{O}_{\mathrm{sig}}$ be the signing oracle, and Q_{sig} be the number of different signing queries during the game by $\mathcal{A}$, Q_i for $i = 0, 1, 2, 4, 5$ be the

[2] H_3 itself is not a PRF, but it is used as a PRF with key prepending. We use this notation for convenience.

number of queries made to H_i by $\mathcal{A}$ where Q_5 includes queries to H_5 made during signing queries.

We begin to prove the security of the deterministic version of AIMer ($\rho \leftarrow 0^\lambda$), and prove that of the probabilistic version later. Without loss of generality, we assume that all messages in signing queries are distinct.

G_1: This game acts same as G_0 except that it aborts if there exists two different queries on H_0 with same outputs. As output length of H_0 is 2λ, we have

$$\Pr[\mathsf{G}_1 \text{ aborts}] \leq \frac{(Q_{\text{sig}} + Q_0)^2}{2^{2\lambda}}.$$

G_2: $\mathcal{O}_{\text{sig}}$ replaces $\mathsf{salt} \in \{0,1\}^\lambda$ and root seeds $(\mathsf{seed}_k)_{k\in[\tau]} \in (\{0,1\}^\lambda)^\tau$ by randomly sampled values, instead of computing $H_3(\mu, \mathsf{pt}, \rho)$. As μ is always distinct for each query, the difference between this game and the previous one reduces to the PRF security of H_3 with secret key pt. Therefore, there exists a PPT adversary $\mathcal{C}$ against the PRF security of H_3 such that

$$|\Pr[\mathcal{A} \text{ wins } \mathsf{G}_1] - \Pr[\mathcal{A} \text{ wins } \mathsf{G}_2]| \leq Q_{\text{sig}} \cdot \mathbf{Adv}^{\mathsf{prf}}_{H_3}(\mathcal{C}).$$

G_3: $\mathcal{O}_{\text{sig}}$ samples $(\mathsf{nodes}[2^{j+1}-1], \mathsf{nodes}[2^{j+1}])$ in ExpandTree uniformly at random instead of computing $H_4(\mathsf{salt}, 0, 2^j, \mathsf{nodes}[2^j - 1])$ and programs the random oracle H_4 to output the sampled value for the corresponding query, for $j \in [\log_2 N]$ in step by step. The simulation is aborted if the queries to H_4 have been made previously, for any j. As salt and $\mathsf{nodes}[2^j - 1]$ are random, this game is indistinguishable with the previous game unless the simulation is aborted, and the probability of abort is

$$\Pr[\mathsf{G}_3 \text{ aborts}] \leq \frac{\log_2 N \cdot Q_{\text{sig}}(Q_{\text{sig}} + Q_4)}{2^{2\lambda}}.$$

G_4: $\mathcal{O}_{\text{sig}}$ samples $\left(\mathsf{com}_0^{(0)}, \mathsf{pt}_0^{(0)}, (y_{0,j}^{(0)})_{j\in[\ell]}, (a_{0,j}^{(0)})_{j\in[\ell+1]}, c_0^{(0)}\right)$ at random instead of computing $H_5(\mathsf{salt}, 0, 0, \mathsf{seed}_0^{(0)})$, and programs the random oracle H_5 to output the same value for the respective query. The simulation is aborted if the queries to H_5 have been made previously. As $\mathsf{salt} \in \{0,1\}^\lambda$ and $\mathsf{seed}_0^{(0)} \in \{0,1\}^\lambda$ are random, this game is indistinguishable with the previous game unless the simulation is aborted, and the probability of abort is

$$\Pr[\mathsf{G}_4 \text{ aborts}] \leq \frac{Q_{\text{sig}}(Q_{\text{sig}} + Q_5)}{2^{2\lambda}}.$$

G_5: $\mathcal{O}_{\text{sig}}$ samples $h_1 \in \{0,1\}^{2\lambda}$ at random instead of computing

$$H_1(\mu, \mathsf{salt}, ((\mathsf{com}_k^{(i)})_{i\in[N]}, \Delta\mathsf{pt}_k, \Delta c_k, (\Delta y_{k,j})_{j\in[\ell]})_{k\in[\tau]})$$

and program the random oracle H_1 to output h_1 for the respective query. The first challenge $(\epsilon_{k,j})_{k\in[\tau],j\in[\ell+1]}$ is derived by expanding h_1. The simulation is

aborted if the queries to H_1 have been made previously. As $\mathsf{com}_0^{(0)} \in \{0,1\}^{2\lambda}$ is random, this game is indistinguishable with the previous game unless the simulation is aborted, and the probability of abort is

$$\Pr[\mathsf{G}_5 \text{ aborts}] \leq \frac{Q_{\mathrm{sig}}(Q_{\mathrm{sig}} + Q_1)}{2^{2\lambda}}.$$

G_6: $\mathcal{O}_{\mathrm{sig}}$ samples $h_2 \in \{0,1\}^{2\lambda}$ at random instead of computing

$$H_2(h_1, \mathsf{salt}, ((\alpha_{k,0}^{(i)}, \ldots, \alpha_{k,\ell}^{(i)})_{i\in[N]}, (v_k^{(i)})_{i\in[N]})_{k\in[\tau]})$$

and program the random oracle H_2 to output h_2 for the respective query. The unopened parties $(\bar{i}_k)_{k\in[\tau]}$ are derived by expanding h_2. The simulation is aborted if the queries to H_2 have been made previously. As $h_1 \in \{0,1\}^{2\lambda}$ is random, this game is indistinguishable with the previous game unless the simulation is aborted, and the probability of abort is

$$\Pr[\mathsf{G}_6 \text{ aborts}] \leq \frac{Q_{\mathrm{sig}}(Q_{\mathrm{sig}} + Q_2)}{2^{2\lambda}}.$$

G_7: $\mathcal{O}_{\mathrm{sig}}$ replaces the seed of the unopened parties $\mathsf{seed}_k^{(\bar{i}_k)}$ in the binary tree by a random element for each $k \in [\tau]$. If $\bar{i}_0 = 0$, it does not need to replace $\mathsf{seed}_0^{(0)}$ with a random element again. Similarly to G_3, it is indistinguishable from the previous game with the advantage bounded by

$$|\Pr[\mathcal{A} \text{ wins } \mathsf{G}_6] - \Pr[\mathcal{A} \text{ wins } \mathsf{G}_7]| \leq \frac{\tau \log_2 N \cdot Q_{\mathrm{sig}}(Q_{\mathrm{sig}} + Q_4)}{2^{2\lambda}}.$$

G_8: $\mathcal{O}_{\mathrm{sig}}$ replaces the outputs of $H_5(\mathsf{salt}, k, \bar{i}_k, \mathsf{seed}_k^{(\bar{i}_k)})$ by randomly sampled elements for each k, and programs the random oracle H_5 to output the same values for the respective queries. Also, $\mathcal{O}_{\mathrm{sig}}$ sets $v_k^{(\bar{i}_k)} \leftarrow 0 - \sum_{i\neq\bar{i}_k} v_k^{(i)}$ for each $k \in [\tau]$. $\mathcal{O}_{\mathrm{sig}}$ aborts if the replaced commitment value collides with that in $H_5(x)$ where x is queried by $\mathcal{A}$. Since $(\mathsf{salt}, \mathsf{seed}_k^{(\bar{i}_k)})$ is a random string of 2λ bits, this game is indistinguishable with the previous game unless the simulation is aborted, and the probability of abort is

$$\Pr[\mathsf{G}_8 \text{ aborts}] \leq \frac{\tau Q_{\mathrm{sig}}(Q_{\mathrm{sig}} + Q_5)}{2^{2\lambda}}.$$

Note that for $k \in [\tau]$ such that $\bar{i}_k \neq N-1$, $\alpha_k^{(\bar{i}_k)}$ is also random and independent to pt.

G_9: $\mathcal{O}_{\mathrm{sig}}$ replaces

$$(\Delta\mathsf{pt}_k, \Delta c_k, (\Delta y_{k,j})_{j\in[\ell]})_{k\in[\tau]}$$

by random elements instead of computing them using pt and S-box outputs. As $\left(\mathsf{com}_k^{(\bar{i}_k)}, \mathsf{pt}_k^{(\bar{i}_k)}, (y_{k,j}^{(\bar{i}_k)})_{j\in[\ell]}, c_k^{(\bar{i}_k)}\right)_{k\in[\tau]}$ is random, the distribution of these variables does not change.

Note that now for all $k, j \in [\tau] \times [\ell + 1]$, $\alpha_{k,j}^{(\bar{i}_k)}$ is random and independent of pt. If the multiplication triple is wrong, then $v_k^{(\bar{i}_k)} \leftarrow -\sum_{i \neq \bar{i}_k} v_k^{(i)}$ is different from an honest value derived from legitimate calculation. However $(\bar{i}_k)$ is unopened and the multiplication check is still passed. Since the signature oracle in G_8 does not depend on the secret key pt, which implies that G_8 can be reduced to the EUF-KO security. Therefore, there exists a PPT adversary $\mathcal{B}$ on EUF-KO security against AIMer such that

$$\Pr[\mathcal{A} \text{ wins } \mathsf{G}_9] \leq \mathbf{Adv}^{\mathsf{euf\text{-}ko}}_{\mathsf{AIMer}}(\mathcal{B}).$$

All in all, we have

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{AIMer}}(\mathcal{A}) &\leq \frac{(Q_{\mathrm{sig}} + Q_0)^2}{2^{2\lambda}} + Q_{\mathrm{sig}} \cdot \mathbf{Adv}^{\mathsf{prf}}_{H_3}(\mathcal{A}) \\
&\quad + (\tau + 1) \log N \cdot \frac{Q_{\mathrm{sig}}(Q_{\mathrm{sig}} + Q_4)}{2^{2\lambda}} + \frac{(\tau + 1) Q_{\mathrm{sig}}(Q_{\mathrm{sig}} + Q_5)}{2^{2\lambda}} \\
&\quad + \frac{Q_{\mathrm{sig}}(Q_{\mathrm{sig}} + Q_1)}{2^{2\lambda}} + \frac{Q_{\mathrm{sig}}(Q_{\mathrm{sig}} + Q_2)}{2^{2\lambda}} + \mathbf{Adv}^{\mathsf{euf\text{-}ko}}_{\mathsf{AIMer}}(\mathcal{A}) \\
&\leq Q_{\mathrm{sig}} \cdot \mathbf{Adv}^{\mathsf{prf}}_{H_3}(\mathcal{C}) + 2(\tau + 1) \log N \cdot \frac{(Q_{\mathrm{sig}} + Q)^2}{2^{2\lambda}} \\
&\quad + \mathbf{Adv}^{\mathsf{euf\text{-}ko}}_{\mathsf{AIMer}}(\mathcal{B})
\end{aligned}$$

provided that $\log N \geq 4$ and $Q_0 + Q_1 + Q_2 + Q_4 + Q_5 \leq Q$. The EUF-CMA advantage is negligible in λ assuming that AIM3 is a secure one-way function and that parameters (N, τ, λ) are appropriately chosen.

For the non-deterministic version of $\mathcal{A}$, all games are defined in a manner almost identical to the deterministic version, with the exception of handling two queries to $\mathcal{O}_{\mathrm{sig}}$ that involve the same messages and ρ values. If (m, ρ) are identical in two queries, the outputs must also be identical; thus, we avoid random sampling and use already programmed outputs for the random oracles in such cases. Consequently, the differences between the adjacent games remain unchanged from the deterministic version, leading to the same bounds on the advantage of $\mathcal{A}$.

□

6 Security Evaluation

6.1 Summary of Expected Security Strength

The AIMer signature scheme provides three levels of security: L1 (AES-128), L3 (AES-192), and L5 (AES-256). Each security level corresponds to the security of AES in the parentheses, and it implies that we expect AIMer with L1, L3, and L5 parameters to be as secure as AES-128, AES-192, AES-256 respectively, against both classical and quantum attacks. In this section, we examine the concrete security of the three components of AIMer: the non-interactive zero-knowledge proof of knowledge (NIZKPoK), the one-way function, and the hash functions.

SECURITY OF THE NIZKPOK SYSTEM. The NIZKPoK system in AIMer is basically BN++ [KZ22] with slight modifications. The security of AIMer is proved in Section 5.1 in the random oracle model.

In the quantum-accessible random oracle model (QROM), an adversary is allowed to make superposition queries to the random oracle. The NIZKPoK system in AIMer (and BN++) follows the spirit of the Fiat-Shamir transform [FS87], and there has been a significant amount of research on the QROM security of the Fiat-Shamir transform [LZ19, DFMS19, DFM20, DFMS22a, DFMS22b]. The NIZKPoK system of AIMer should be seen as a variant of the original Fiat-Shamir transform, while its security is not immediate from the above results, and we will prove it as a future work.

The parameters N and τ are fixed based on the soundness analysis given in [KZ22]; we see that an attacker should make at least 2^λ guesses in order to produce a valid forgery without any knowledge of the secret key. Since a single guess involves at least one hash or XOF call (where a single call of hash is more costly than AES), AIMer with our recommended sets of parameters would provide a sufficient level of security.

SECURITY OF AIM3. AIM3 is a one-way function, which does not follow the traditional design rationale of symmetric primitives. It takes random strings iv and pt as input, and outputs $\mathsf{ct} = \mathsf{AIM3}(\mathsf{iv}, \mathsf{pt})$. We expect that finding pt^* for a given pair $(\mathsf{iv}, \mathsf{ct})$ such that $\mathsf{ct} = \mathsf{AIM3}(\mathsf{iv}, \mathsf{pt}^*)$ is as hard as key recovery of AES with the same security level. To support our claim, we investigate its security against brute-force attacks, algebraic attacks, statistical attacks, and quantum attacks in Section 6.3.

For the algebraic attacks, we analyze the security of AIM3 against Gröbner basis algorithm, the magic-pot attack [BGFU26], the fast exhaustive search attacks [LMOM23, Bou22], the linearization attack by Zhang et al. [ZWY+23], and the resultant-based attack [SCL+25]. We argue that AIM3 is secure against these attacks under the assumption of the semi-regular system even in case such that an adversary chooses intermediate variables not only the outputs of the S-boxes. All the algebraic attacks on AIM3 require more gate-count complexity than required for AES, or require more than 2^λ memory bits. For quantum attacks, we looked into Grover's algorithm, quantum algebraic attacks, and quantum generic attacks. The most powerful attack among them turns out to be Grover's algorithm while its complexity against AIM3 is not lower than that applied to AES with the same security level. All the analysis on AIM3 is summarized in Section 6.3.

In the multi-user setting, we expect that finding one of pt_i given multiple pairs $\{(\mathsf{iv}_i, \mathsf{ct}_i)\}$ such that $\mathsf{ct}_i = \mathsf{AIM3}(\mathsf{iv}_i, \mathsf{pt}_i)$ for some i is hard assuming that iv's are randomly chosen. If iv's can be arbitrarily chosen, a collision of ct_i is connected to a forgery. For example, if an IV value iv^* collides q times in a set of public keys, an attacker may compute the function $\mathsf{AIM3}(\mathsf{iv}^*, \mathsf{pt})$ for c times with distinct pt's. Then, the probability of collision is approximately $qc/2^n$, which implies a security degradation.

Except for the risk of collision, multiple pairs $\{(\mathsf{iv}_i, \mathsf{ct}_i)\}$ do not lead to a strengthened attack on AIM3 to the best of our knowledge. For algebraic attacks, any two sets of equations built for distinct pt's are not compatible.

HASH FUNCTION SECURITY. The AIMer signature scheme requires a lot of calls to hash functions and extendable output functions (XOFs). All the hash functions and XOFs are based on NIST-standardized XOF SHAKE [NIS15]. SHAKE128 is used for the L1 parameters, and SHAKE256 is used for the L3 and L5 parameters. All the hash functions use 2λ-bit digests of the SHAKE output.

We expect the concrete security provided by SHAKE for collision and preimage resistance as claimed in [NIS15]. For $\lambda \in \{128, 192, 256\}$, the preimage resistance of SHAKE-λ with k-bit digest is claimed to be $\min(2^k, 2^{2\lambda})$ in the classical setting, and a cryptographic hash function with k-bit digest is generally believed to have $O(2^{k/2})$ preimage resistance in the quantum setting [Gro96]. In both cases, hash functions with 2λ-bit digests provide λ-bit preimage resistance. For collision resistance, while a generic quantum algorithm of finding a hash collision is of complexity $O(2^{k/3})$ when the output size is k bits [BHT98], Bernstein pointed out that the quantum hash collision algorithm has worse performance compared to classical algorithms in practice [Ber09]. Since it is claimed that k-bit digests of SHAKE-λ has collision resistance of $\min(2^{k/2}, 2^{\lambda})$ against classical attacks, the 2λ-bit digest also allows λ-bit collision resistance against classical and quantum attacks.

6.2 Soundness Analysis

In this section, we analyze the soundness error of the AIMer signature scheme to determine the set of parameters (λ, N, τ). A more formal analysis is given in Section 5.1. Let τ_1 and τ_2 denote the number of repetitions for which the attacker needs to make correct guesses on the first challenge $\epsilon_{k,j}$ in Phase 2 and the second challenge $\bar{i}_k$ in Phase 4 in Algorithm 10, respectively. Then, it should be the case that $\tau = \tau_1 + \tau_2$. For $i = 1, 2$, let P_i be the probability that the attacker makes correct guesses for τ_i challenges in the i-th challenge space.

The first challenge is sampled from the set of size 2^n, so the probability of correctly guessing τ_1 challenges in the first challenge space is given as

$$P_1 = \sum_{k=\tau_1}^{\tau} \binom{\tau}{k} p^k \cdot (1-p)^{\tau-k}$$

where $p = 2^{-\lambda}$. On the other hand, since the second challenge space is of size N, and the attacker needs to make correct guesses in the remaining repetitions, one has

$$P_2 = 1/N^{\tau_2} = 1/N^{\tau-\tau_1}.$$

Overall, the attack complexity is given as

$$\min_{0 \le \tau_1 \le \tau} (1/P_1 + 1/P_2).$$

Our parameters are set in a way such that the attack complexity is larger than or equal to 2^{λ}.

6.3 Known Attacks to AIM3

6.3.1 Algebraic Attacks

Since our attack model does not allow multiple evaluations for a single instance of AIM3, we do not consider interpolation, higher-order differential, and cube attacks. As discussed in [KHSL24], we consider the Gröbner basis attack on various systems obtained from a single evaluation of AIM3. As several attacks on AIM and AIM2 were proposed, we describe how those attacks are mitigated in AIM3.

ON THE NUMBER OF QUADRATIC EQUATIONS OF THE AIM3 S-BOX. Each S-box of AIM3 is of the form

$$S[e](x) = x^{2^e-1} + x^{2^n-2},$$

which combines a Mersenne exponentiation $x \mapsto x^{2^e-1}$ with the inverse map $x \mapsto x^{2^n-2}$, where $x^{2^n-2} = x^{-1}$ for $x \neq 0$ and $0^{2^n-2} = 0$. In contrast to the invertible Mersenne S-box of AIM2, this updated S-box is non-invertible.

THE SPACE OF QUADRATIC EQUATIONS. Fix a basis of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$ and identify a pair $(x, y) \in \mathbb{F}_{2^n}^2$ with its $2n$ coordinate bits $z = (x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1}) \in \mathbb{F}_2^{2n}$. Since key generation rejects any instance with a zero S-box input, we consider the restricted S-box graph

$$V = \{(x, S[e](x)) : x \in \mathbb{F}_{2^n}^*\} \subseteq \mathbb{F}_2^{2n}.$$

Let

$$\mathcal{B}_{2n} = \mathbb{F}_2[z_0, \ldots, z_{2n-1}]/\langle z_i^2 - z_i : 0 \leq i < 2n\rangle$$

be the Boolean polynomial ring. We define

$$\mathrm{QE}(V) = \{q \in \mathcal{B}_{2n} : \deg q \leq 2 \text{ and } q(v) = 0 \text{ for every } v \in V\}.$$

Thus, $\mathrm{QE}(V)$ is the $\mathbb{F}_2$-vector space of Boolean equations of degree at most two that vanish on the entire restricted S-box graph, and $\dim_{\mathbb{F}_2} \mathrm{QE}(V)$ is the number of linearly independent such equations.

ANALYTIC LOWER BOUND. Let $y = S[e](x)$. For $x \neq 0$, the input-output pair satisfies the following three relations over $\mathbb{F}_{2^n}$:

$$\begin{cases} P_1 : \ xy = x^{2^e} + 1, \\ P_2 : \ x^2 y = x^{2^e+1} + x, \\ P_3 : \ xy^2 = x^{2^e} y + y. \end{cases}$$

The second and third relations are obtained by multiplying the first one by x and y, respectively. Since Frobenius maps are $\mathbb{F}_2$-linear, every term in these relations has Boolean degree at most two, and each relation gives n Boolean coordinate equations. The identities $P_2 = xP_1$ and $P_3 = yP_1$ are not $\mathbb{F}_2$-linear dependencies: an $\mathbb{F}_2$-linear combination permits only fixed coefficients, not multiplication by x or y.

To prove independence for the recommended exponents $2 \le e \le n-2$, move every term of each relation to the left-hand side and let P_j also denote the resulting polynomial. Suppose that an $\mathbb{F}_2$-linear combination of all their coordinate equations is the zero Boolean polynomial. Since the trace pairing $(a,b) \mapsto \operatorname{Tr}(ab)$ is nondegenerate, every $\mathbb{F}_2$-linear functional on $\mathbb{F}_{2^n}$ is uniquely of the form $a \mapsto \operatorname{Tr}(\lambda a)$ for some $\lambda \in \mathbb{F}_{2^n}$. Consequently, there exist $\lambda_1, \lambda_2, \lambda_3 \in \mathbb{F}_{2^n}$ such that the assumed dependence is

$$\operatorname{Tr}(\lambda_1 P_1) + \operatorname{Tr}(\lambda_2 P_2) + \operatorname{Tr}(\lambda_3 P_3) = 0.$$

Consider only the mixed quadratic terms, that is, the terms containing both x and y. The uniqueness of the algebraic normal form prevents the terms involving only x, only y, or neither variable from cancelling these mixed terms. We therefore obtain

$$\operatorname{Tr}\bigl(\lambda_1 xy + \lambda_2 x^2 y + \lambda_3 x y^2 + \lambda_3 x^{2^e} y\bigr) = 0.$$

The trace is invariant under Frobenius, and hence

$$\operatorname{Tr}(\lambda_3 x y^2) = \operatorname{Tr}\Bigl((\lambda_3 x y^2)^{2^{n-1}}\Bigr) = \operatorname{Tr}\Bigl(\lambda_3^{2^{n-1}} x^{2^{n-1}} y\Bigr),$$

where we used $y^{2^n} = y$. Thus, the mixed part is $\operatorname{Tr}(yL(x))$, where

$$L(X) = \lambda_1 X + \lambda_2 X^2 + \lambda_3 X^{2^e} + \lambda_3^{2^{n-1}} X^{2^{n-1}}.$$

If $\operatorname{Tr}(yL(x)) = 0$ for every $x, y \in \mathbb{F}_{2^n}$, the nondegeneracy of the trace pairing implies that $L(x) = 0$ for every $x \in \mathbb{F}_{2^n}$. The Frobenius indices 0, 1, e, and $n-1$ are pairwise distinct because $2 \le e \le n-2$. Moreover, $L(X)$ has ordinary degree at most $2^{n-1} < 2^n$; therefore, it cannot vanish at all 2^n field elements unless it is the zero polynomial. Its coefficients must all be zero, so $\lambda_3 = 0$, $\lambda_2 = 0$, and $\lambda_1 = 0$. Hence the assumed linear dependence is trivial. Consequently,

$$\dim_{\mathbb{F}_2} \mathrm{QE}(V) \ge 3n.$$

COMPUTER-ASSISTED UPPER BOUND. Every element of $\mathcal{B}_{2n}$ of degree at most two has a unique coefficient vector over the

$$D_2 = 1 + 2n + \binom{2n}{2}$$

square-free monomials 1, z_j, and $z_j z_k$ for $j < k$. For a finite sample $T \subseteq V$, let $\mu(v) \in \mathbb{F}_2^{D_2}$ be the evaluation vector of these monomials at v, and let $M_T \in \mathbb{F}_2^{|T| \times D_2}$ be the matrix whose rows are $\mu(v)$ for $v \in T$. If c_q is the coefficient vector of q, then q vanishes on T if and only if $M_T c_q = 0$. It follows from rank–nullity that

$$\mathrm{QE}(V) \subseteq \ker M_T \quad \text{and} \quad \dim_{\mathbb{F}_2} \mathrm{QE}(V) \le D_2 - \operatorname{rank}_{\mathbb{F}_2} M_T.$$

We form T by sampling nonzero inputs x at random. The validity of this upper bound does not rely on the randomness of x: every finite $T \subseteq V$ gives a sound upper bound, while random sampling is used only to obtain a tight one. For every

exponent in the recommended parameter sets at $n \in \{128, 192, 256\}$, exact $\mathbb{F}_2$ elimination gives

$$\operatorname{rank}_{\mathbb{F}_2} M_T = D_2 - 3n.$$

Combining this upper bound with the analytic lower bound yields

$$\dim_{\mathbb{F}_2} \mathrm{QE}(V) = 3n$$

for every recommended S-box. Pseudocode for this exact rank computation is given in Appendix A.

QUADRATIC SYSTEM OF AIM3. Since the updated S-box is non-invertible, a single evaluation of AIM3 can be modeled only by one system of quadratic equations in $(\ell + 1)n$ variables; the reduced-variable systems of higher degree available for AIM2 no longer exist, since building them requires inverting the S-box to eliminate variables.

We build the system S_{quad} from a single evaluation of AIM3 using the following variables.

- x: the input of AIM3, i.e., pt
- t_i: the output of the i-th S-box $S[e_i]$, i.e., y_i, for $i = 0, \ldots, \ell$

Each S-box input is an $\mathbb{F}_2$-affine function of these variables, namely $x_i = A_i(x) + b_i$ for $i \in [\ell]$ at the first round and $x_\ell = b_\ell + \sum_{i\in[\ell]} A_{\ell+i}(t_i)$ at the second round. Moreover, the feed-forward $\mathsf{ct} = \mathsf{pt} + y_\ell$ determines the last output as $t_\ell = x + \mathsf{ct}$, so that t_ℓ is not a free variable and the system is in the $(\ell+1)n$ variables $x, t_0, \ldots, t_{\ell-1}$. As AIM3 evaluates $\ell + 1$ S-boxes, each of which induces $3n$ Boolean quadratic equations in $x, t_0, \ldots, t_{\ell-1}$ as shown above, the system S_{quad} consists of $3(\ell + 1)n$ quadratic equations in $(\ell + 1)n$ variables. Table 4 summarizes the security against Gröbner basis attack on S_{quad}.

Scheme	#Var	Variables	#Eq	Gröbner Basis		
				k	d_{reg}	Time
AIM3-I	$3n$	x, t_0, t_1	$9n$	57	19	262.6
AIM3-III	$3n$	x, t_0, t_1	$9n$	102	25	381.0
AIM3-V	$4n$	x, t_0, t_1, t_2	$12n$	150	44	651.5

Table 4: The systems of quadratic equations and their security against the Gröbner basis attack. #Var and #Eq denote the numbers of variables and quadratic equations, respectively. All the complexities are measured by (3) with $\omega = 2$. k is the number of guessed bits and d_{reg} is the degree of regularity. 'Time' is in log.

We remark that S_{quad} contains no quadratic equations in two distinct S-box outputs t_i and t_j. For AIM2, such cross relations (the additional $\binom{\ell+1}{2}n$ equations induced by $(\gamma_i + \gamma_j)t_i t_j = t_i^{2^{e_i}} t_j + t_i t_j^{2^{e_j}}$) are obtained by eliminating the common input shared by two S-boxes, which requires inverting the S-boxes to represent

each input as a low-degree polynomial in its output. Since the updated S-box is non-invertible, such a representation is unavailable and no such cross relation exists. Hence, S_{quad} consists of only $3(\ell+1)n$ quadratic equations, fewer than the $3(\ell+1)n + \binom{\ell+1}{2}n$ equations of AIM2, which is favorable for resistance against algebraic attacks.

MAGIC POT ATTACK. Biryukov et al. [BGFU26] proposed the magic-pot attack, which recovers pt from a single public key $(\mathsf{iv}, \mathsf{ct})$ by turning one evaluation of AIM2 into linear algebra over $\mathbb{F}_{2^n}[\mathsf{pt}]$. Because pt enters each first-round S-box only at degree 1 (as $\mathsf{pt} + \gamma_i$), it can be treated as a scalar parameter, and the low-degree Mersenne S-boxes then make every equation linear in the S-box-output monomials $t_i^{2^j}$ over $\mathbb{F}_{2^n}[\mathsf{pt}]$. Growing this system in the style of the XL algorithm and eliminating the monomials leaves a single univariate polynomial $g(\mathsf{pt})$ of degree $\xi = O\big((\ell+1)\,2^{\ell n/(\ell+1)}\big)$ whose roots include the key; root-finding costs $2^{103.7}/2^{147.5}/2^{213.2}$ for AIM2-I/III/V, below the claimed levels by about $24/45/43$ bits.

AIM3 mitigates by placing an iv-generated random invertible linear layer before the first-round S-boxes. This update increases the algebraic degree over $\mathbb{F}_{2^n}$, keeping $3n$ quadratic equations for the input and output of the S-boxes over $\mathbb{F}_2$, so the magic pot attack is not available on AIM3.

FAST EXHAUSTIVE SEARCH. The fast exhaustive search attacks [BCC+10, Bou22] are infeasible if the target polynomial system is of a high degree. Although the time complexity of the fast exhaustive search is claimed to be $4d\log(n)2^n$, there is a hidden preprocessing cost

$$T = \sum_{k=0}^{d-1} k\binom{n}{k}\binom{k}{\min(d-k,k)} \geq \frac{2d}{3}2^{2d/3}\binom{n}{\lfloor 2d/3 \rfloor}$$

in binary operations where $\binom{n}{\downarrow k} = \sum_{i=0}^{k}\binom{n}{i}$. One can see that $T \gg d2^n$ if $d \geq 0.341n$. Furthermore, if $d \geq n/2$, then the memory complexity will also be higher than 2^n bits.

LINEARIZATION ATTACKS ON AIM BY GUESSING. Zhang et al. [ZWY+23] proposed an algebraic attack on AIM that linearizes the S-boxes at the first round by guessing. This attack is not applicable to AIM3 since the affine map $\mathsf{Lin}_{\mathsf{in}}[\mathsf{iv}]$ makes the inputs to the S-boxes different. This is the simplest patch among the possible ones proposed by the authors.

ALGEBRAIC ATTACKS ON SYMMETRIC PRIMITIVES WITH LARGE S-BOX. Several symmetric primitives based on large fields have been proposed with applications to zero-knowledge proof systems such as MiMC [AGR+16], Jarvis [AD18], and Poseidon [GKR+21]. Some of them have been analyzed with algebraic attacks exploiting the property that their linear layers are represented as polynomials of low degrees over large fields [ACG+19, EGL+20]. However, AIM3 uses a randomized linear layer which is expected to have degree 2^{n-1} over $\mathbb{F}_{2^n}$. For this reason, the above attacks would not apply to AIM3.

APPLICABILITY OF ALGEBRAIC ATTACKS ON LOWMC. LowMC [ARS+15] is the first FHE/MPC-friendly block cipher, and one of its applications is to the Picnic signa-

ture scheme. LowMC has been analyzed in the context of the signature scheme, where an adversary is given only a single plaintext-ciphertext pair. In this setting, a number of algebraic attacks on LowMC have been proposed [BBDV20, BBVY21, LIM21, Din21, LMSI22, BBCV22], mainly based on two algebraic techniques: linearization by guessing, and the polynomial method [Bei93].

The main idea of linearization-based algebraic attacks on LowMC, first proposed in [BBDV20], is to linearize the underlying S-boxes by guessing a single output bit for each S-box evaluation. In this way, one obtains a system of low-degree polynomial equations at the cost of guessing a small number of bits, and it can be solved efficiently. This linearization technique has been further extended [BBVY21, LIM21]. For AIM3 having large S-boxes with dense implicit equations, it seems to be infeasible to linearize the S-boxes by guessing some of the input/output bits.

The polynomial method [Bei93] has been studied in complexity theory, and later found its application to design algorithms for certain problems [Wil14], one of which is to solve a system of polynomial equations over a finite field. The resulting algorithm is known as the first algorithm that achieves exponential speedup over the exhaustive search even in the worst case [LPT+17]. Recently, Dinur [Din21] proposed a generic equation-solving algorithm based on the polynomial method with time complexity $O(n^2 \cdot 2^{(1-1/(2.7d))n})$ where n is the number of variables and d is the degree of the system. This attack does not apply to AIM3 because the system on the input n variables have the full degree, and the only low-degree system is the quadratic system in $(\ell+1)n$ variables.

RESULTANT-BASED ATTACK. Sun et al. [SCL+25] proposed two resultant-based algebraic attacks on AIM2. One is on AIM2-III, and the other is on AIM2-V. These approaches basically model AIM2 as a system of few-variable polynomials over a large field ($GF(2^\lambda)$), reduce degree of the polynomials by setting intermediate linear layer, and eliminate the variables using the Sylvester matrix. By adding an affine layer before the first-round S-boxes, AIM3 can mitigate this attack.

6.3.2 Brute-Force Attack

Saarinen proposed an efficient brute-force attack for AIM using a linear feedback shift register.[3] The core idea of the attack is establishing a simpler equation by introducing an output of the S-box as a new variable, but this approach is not available on AIM3 as its S-box is not invertible.

6.3.3 Quantum Attacks

Quantum attacks are classified into two types according to the attack model. In the Q1 model, an adversary is allowed to use quantum computation without making any quantum query, while in the Q2 model, both quantum computation and quantum queries are allowed [Zha12].

As a generic algorithm for exhaustive key search, Grover's algorithm has been known to give quadratic speedup compared to the classical brute-force attack [Gro96].

[3] https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/BI2ilXblNy0

In this section, we investigate if any specialized quantum algorithm targeting AIM3 might possibly achieve better efficiency than Grover's algorithm in the Q1 model.

COST OF GROVER'S ALGORITHM. We consider the cost metric of NIST [NIS22], which is defined as the product of the quantum circuit size and the quantum circuit depth with respect to Clifford and T gates.

Given a one-way function f taking n bits as input, the circuit size and the depth of the preimage-finding attack on f using Grover's algorithm is estimated as follows [BJ24].

$$\text{(Grover's circuit size/depth)} = \text{(size/depth of } f) \times 2 \times \left\lfloor \frac{\pi}{4}\sqrt{2^n} \right\rfloor.$$

The quantum circuit size and the depth of AIM3 can be computed in a modular manner. AIM3 is based on three types of operations: finite field multiplication, finite field squaring, and random matrix multiplication. The costs of finite field multiplications, finite field squaring, and evaluation of the linear layer are estimated using the result in [JOKS24].

The main difference from AIM2 arises from the S-box. The S-box of AIM3,

$$S[e](x) = x^{2^e-1} + x^{2^n-2} = \mathsf{Mer}[e](x) + x^{-1},$$

is not invertible. For AIM2, the circuit depth can be minimized by introducing the output of each inverse S-box as a variable, $t_i = \mathsf{Mer}[e_i]^{-1}(x + \gamma_i)$, and checking whether these variables satisfy the linear relation between them; then all the $\ell + 1$ S-boxes are evaluated in parallel and the critical path traverses only a single S-box. Such a construction is unavailable for AIM3, since its S-box cannot be inverted. Instead, for each candidate pt, one has to evaluate AIM3 in the forward direction,

$$x_i = A_i(\mathsf{pt}) + b_i \xrightarrow{S[e_i],\ i\in[\ell]} t_i \xrightarrow{\mathsf{Lin[iv]}} x_\ell \xrightarrow{S[e_\ell]} t_\ell,$$

so that the critical path traverses two S-boxes, namely the ℓ S-boxes of the first round evaluated in parallel followed by the second-round S-box. Introducing intermediate variables to parallelize the two rounds, as done for AIM2, requires inverting the S-box and is thus unavailable, while guessing them instead would only enlarge the Grover search space. Hence evaluating AIM3 in the forward direction is optimal.

Each S-box of AIM3 evaluates the Mersenne exponentiation x^{2^e-1} and the inverse $x^{-1} = x^{2^n-2} = (x^{2^{n-1}-1})^2$ in parallel and adds the results, where the inverse is computed by the Itoh–Tsujii algorithm [IT88]. Table 5 summarizes the total number of operations and the depth of operations for the forward evaluation of AIM3, computed from the addition chains of the AIM3 S-boxes. Based on these numbers and the above references, the estimated cost of Grover's algorithm on AIM3-I, III, and V is $2^{165.6}$, $2^{232.5}$, and $2^{298.0}$, respectively. These costs exceed the gate-count complexity of AES ($2^{143}/2^{207}/2^{272}$) as well as the corresponding thresholds of NIST, so AIM3-I, III, and V satisfy the security levels I, III, and V, respectively.[4]

[4] In the call for proposals by NIST [NIS22], the security levels I, III, V are defined as the strength of AES-128, AES-192, AES-256, respectively, against Grover's algorithm.

The total cost of Grover's algorithm might be further reduced than expected; a better representation of the AIM3 circuit with respect to the total cost might be proposed, or a more efficient addition chain might be discovered. However, we believe that the advance of such optimization technique will not reduce the total cost below that of AES with the same security level, since the amount attributable to finite multiplications is more than the total cost of AES.

Scheme	#Operations, Depth		Total Cost	Level of Security
	FF Mul	FF Square		
AIM3-I	47, 24	399, 254	165.6	I ($\geq$157)
AIM3-III	59, 26	651, 382	232.5	III ($\geq$221)
AIM3-V	73, 28	1056, 510	298.0	V ($\geq$285)

Table 5: The number of operations and the depth for each type of operation used in AIM3, and the total cost of Grover's algorithm on AIM3 for each level of security.

QUANTUM ALGEBRAIC ATTACK. When an algebraic root-finding algorithm works over a small field, the guess-and-determine strategy might be effectively combined with Grover's algorithm, reducing the overall time complexity.

The GroverXL algorithm [BY18] is a quantum version of the FXL algorithm [CKPS00], which solves a system of multivariate quadratic equations over a finite field. A single evaluation of AIM3 can be represented by Boolean quadratic equations using intermediate variables. Precisely, we have a system of $3(\ell+1)n$ quadratic equations in $(\ell+1)n$ variables. For this system of equations, the time complexity of GroverXL is given as $2^{(1.3484+o(1))n}$ for AIM3-I, III and $2^{(1.7978+o(1))n}$ for AIM3-V when using $\omega = 2$, which is worse than Grover's algorithm.

The `QuantumBooleanSolve` algorithm [FHK+17] is a quantum version of the `BooleanSolve` algorithm [BFSS13], which solves a system of Boolean quadratic equations. In [FHK+17], its time complexity has been analyzed only for a system of equations with the same number of variables and equations. A single evaluation of AIM3 can be represented by $3(\ell+1)n$ quadratic equations in $(\ell+1)n$ variables. In the paper, the complexities are summarized only when the number of equations are same as the number of variables. We numerically found the minimum complexities according to the number of guessed variables. The complexity of probabilistic variant of `QuantumBooleanSolve` for AIM3-I, III is minimized to $O(2^{1.162n})$ when 46.8%[5] of variables are guessed, and that for AIM3-V is minimized to $O(2^{1.549n})$ when 46.8% of variables are guessed, which is worse than Grover's algorithm.

In contrast to the algorithms discussed above, Chen and Gao [CG22] proposed a quantum algorithm to solve a system of multivariate equations using the Harrow-Hassidim-Lloyd (HHL) algorithm [HHL09] that solves a sparse system of linear equations with exponential speedup. In brief, Chen and Gao's algorithm solves a system of linear equations from the Macaulay matrix by the HHL algorithm. It has been claimed that this algorithm enjoys exponential speedup for a certain set of parameters. When applied to AIM3, the hamming weight of the secret key should

[5] In the original paper, this value is denoted by $1-\gamma$.

be smaller than $O(\log n)$ to achieve exponential speedup [DGG+21]. Otherwise, this algorithm is slower than Grover's algorithm [DGG+21].

QUANTUM GENERIC ATTACK. A generic attack does not use any particular property of the underlying components (e.g., S-boxes for AIM3). The underlying smaller primitives are typically modeled as public random permutations or functions. The Even-Mansour cipher [EM97], the FX-construction [KR01] and a Feistel cipher [LR86] have been analyzed in the classic and generic attack model. As their quantum analogues, the Even-Mansour cipher [KM12, BHNP+19], the FX-construction [LM17, HS18] and a Feistel cipher [KM10] have been analyzed in the Q1 or Q2 model. Most of these attacks can be seen as a combination of Simon's period finding algorithm [Sim97] (in the Q2 model), and Grover's/offline Simon's algorithms [BHNP+19] (in the Q1 model). Since Simon's period finding algorithm requires multiple queries to a *keyed* construction (which is not the case for AIM3), we believe that the above attacks do not apply to AIM3 in a straightforward manner.

6.4 Attacks in the Multi-User Setting

The analysis of the multi-user security of a cryptographic scheme is crucial, as most cryptographic schemes are used by multiple users in practice. In this setting, an adversary is given multiple users' instances (e.g., public keys and corresponding signatures), and it aims to attack one of them.

MULTI-USER EUF-CMA SECURITY. Since EUF-CMA security is a fundamental requirement for digital signatures, it is natural to consider Multi-User EUF-CMA (MU-EUF-CMA) security in the multi-user setting. Here, the adversary is given multiple signing oracles (corresponding to distinct public keys), and tries to generate a valid forgery under one of the given public keys through a chosen message attack. Thanks to the generic reduction from EUF-CMA security to MU-EUF-CMA security [GMLS02], AIMer provides MU-EUF-CMA security with losses that are (at most) linear in the number of users. In addition, the concrete design of AIMer takes into account multi-user attacks, or more generally, *multi-target attacks*.

MULTI-TARGET ATTACKS. In multi-target attacks, an adversary is given multiple targets, for example, the outputs of a cryptosystem computed with different secret keys. This is inherently possible in the multi-user setting, and even in a single-user setting, when multiple targets are available to the adversary.

There are many examples of successful multi-target attacks. In [DN19], Dinur and Nadler proposed an effective multi-target attack on Picnic version 1.0. The main idea is that an attacker collects multiple outputs generated from unknown seeds of the unopened party in the MPCitH protocol, compares them to the outputs from guessed ones, trying to find a collision using a certain efficient algorithm such as hash tables to recover the seed of the unopened party. Once the seed is revealed, the secret key is also recovered from its additive shares. The above attack is mitigated in the next version of the Picnic signature by using a random salt and domain separation prefixes as an additional input of underlying hash functions and XOFs.

Multi-target attacks have also been proposed on hash-based signature schemes

[BXKSN21, YAG21]. As many hash outputs are used as secret keys of the underlying one-time signature (OTS), the seed guessing technique also works in hash-based signatures, and the recovered seed reveals the corresponding secret keys. It can be mitigated by domain separation of the hash functions according to the position of the OTS instances. Another multi-target attack on SPHINCS$^+$ of the L5 parameter set exploits the small state size of SHA-256 [PKC22], but it is not applicable when SHAKE256 is used as the underlying hash function.

When it comes to AIMer, the use of iv mitigates multi-target attacks. AIM3 generates its linear layer from a random iv, so not only each user has a different secret key (i.e., the input of AIM3), but also the functions themselves are all different. Moreover, similarly to the mitigation techniques described above, all inputs to hash functions have at least 2λ-bit randomness (e.g., salt, seeds, or commits), and domain separation is applied to each hash function and the XOF used in the signature. It would prevent any type of efficient multi-target preimage search attack, such as time/memory/data trade-off attacks [BS00] and parallel quantum multi-target preimage attacks [BB18]. We refer to Section 4.1.2 for detailed specifications of the hash functions.

KEY SUBSTITUTION ATTACKS. In a key substitution attack (KSA), an adversary is given a signature σ_A under a public key pk_A. Then the adversary tries to produce a fake public key pk_E such that σ_A is also a valid signature under pk_E. This type of attacks were first considered in [BWM99], under the name *unknown key-share attacks*, and later formalized in [MS04]. Although the possibility of KSA does not violate the MU-EUF-CMA security, it may need to be considered in practical applications of digital signatures, in particular, when non-repudation property is required [KM13]. Fortunately, the security against KSAs can be achieved in the generic way using the following theorem.

Theorem 3 (Theorem 6 in [MS04])**.** *Let* $\Pi = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *be an EUF-CMA secure digital signature scheme. Then,* $\Pi' = (\mathsf{KeyGen}, \mathsf{Sign}', \mathsf{Verify})$ *is a secure digital signature scheme against KSAs with*

$$\mathsf{Sign}' = \mathsf{Sign}(sk, \mathsf{Encode}(pk, m))$$

where Encode *is an unambiguous encoding scheme of public keys and messages.*

In AIMer, a (fixed length) public key is always appended to the message before hashing, so we believe that AIMer is secure against KSAs.

6.5 Side-Channel Attacks

The signing algorithms, which manipulate the secret key of AIMer, are executed in constant time. Although AIMer's key generation uses rejection sampling, the rejection probability is approximately $2^{-\lambda+2}$, which is negligible; therefore, it can be considered to run in constant time. Therefore, we anticipate no vulnerabilities to either simple power attacks [KJJ99] or timing attacks [Koc96].

Numerous masking techniques designed to thwart side-channel attacks adopt the principle of secret sharing [ISW03, BBP$^+$17, KR19]. Since AIMer generates a

signature by simulating the secret-shared computation of a one-way function, it seems to provide inherent mitigation against certain side-channel attacks.

Despite this, AIMer is expected to be susceptible to power attacks [KJJ99], electromagnetic radiation attacks [QS01], and fault-injection attacks [BDL97] if no countermeasures are implemented. Specifically, during the signing algorithm, the secret key pt requires careful handling since it is used in field arithmetic operations. For $\Delta \mathsf{pt}_k$ in phase 1 of the signing algorithm, calculated as $\mathsf{pt} - \sum_i \mathsf{pt}_k^{(i)}$, it is crucial to perform field additions by pt only after the complete computation of $\sum_i \mathsf{pt}_k^{(i)}$ to avoid exposure through differential power attacks [KJJ99] or correlation power attacks [BCO04]. This precaution is based on the fact that an adversary knows most of $\mathsf{pt}_k^{(i)}$ values [HHL+23]. Therefore, all implementations ensure that field addition involving pt occurs only once. Furthermore, in both reference and optimized implementations, field multiplication employs a temporary table based on the first input, with the second input serving as a table reference. Thus, to prevent cache attacks [Ber05], pt is inputted as the first operand in field multiplication operations.

Recently, machine learning techniques have been integrated with various existing side-channel attacks targeting both conventional and post-quantum encryption schemes [DGD+19, WD20, DNGW23]. In response, we plan to develop effective countermeasures against these attacks in the future.

7 Key and Signature Sizes

Table 6 presents the sizes of the public key, secret key, and signature for various parameter sets using the NIST/SUPERCOP API[6] functions `crypto_sign_keypair`, `crypto_sign`, and `crypto_sign_open`.

Parameters	Public key size (bytes)	Secret key size (bytes)	Signature size (bytes)
AIMer-128f	32	48	6,944
AIMer-128s	32	48	4,704
AIMer-192f	48	72	15,408
AIMer-192s	48	72	10,320
AIMer-256f	64	96	31,360
AIMer-256s	64	96	20,224

Table 6: Key and signature sizes for various parameter sets.

[6] https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/example-files/api-notes.pdf

8 Advantages and Limitations

8.1 General

AIMer shares similar advantages with other MPCitH-based signature schemes as follows.

- The security of AIMer depends only on the security of the underlying symmetric primitives. In particular, the security of AIMer is reduced to the one-wayness of AIM3 in the random oracle model.
- Among the signature schemes whose security depends only on symmetric primitives, AIMer enjoys the smallest signature size.
- AIMer enjoys the small secret and public key size; the small key size makes it easier to apply to many PKI applications based on multi-chain certificates or frequent certificate transmission.
- Key generation is simple and fast.
- AIMer provides a trade-off between the execution time and the signature size. This feature makes it possible to adjust the performance based on the user's requirements.
- AIMer is resistant to the reuse of the public randomnesses such as iv and salt. To the best of our knowledge, multiple uses of an identical value of iv or salt linearly increase the probability of a pk-collision or a multi-target hash collision, respectively.

AIMer also has similar limitations to other MPCitH-based signature schemes as follows.

- The signature size is relatively large compared to standardized lattice-based schemes.
- Signing and verification are slower compared to standardized lattice-based schemes.

8.2 Compatibility with Existing Protocols

The signature size of AIMer is larger than NIST selected algorithms such as CRYSTALS-Dilithium [LDK+22] and Falcon [PFH+22] except SPHINCS+ [HBD+22], while the bandwidth of AIMer is sufficiently small so that it is still compatible with many existing protocols.

A Quadratic-Equation Rank Computation

Algorithm 17 summarizes the exact upper-bound experiment from Section 6.3.1. Field elements are represented in the polynomial basis of $\mathbb{F}_{2^n}$, and Gaussian elimination uses only bitwise XOR over $\mathbb{F}_2$. Nonzero inputs are sampled at random to form the finite set $T \subseteq V$. Randomness of x is used only to obtain a tight upper bound and is not required for the validity of the experiment.

Algorithm 17: Exact upper-bound certificate for $\dim_{\mathbb{F}_2} \mathrm{QE}(V)$

Input: Field degree n, S-box exponent e, irreducible polynomial f, and sample budget B

Output: An upper bound U on $\dim_{\mathbb{F}_2} \mathrm{QE}(V)$, or a certificate that $\dim_{\mathbb{F}_2} \mathrm{QE}(V) = 3n$

$D_2 \leftarrow 1 + 2n + \binom{2n}{2}$;
$\mathcal{P} \leftarrow \emptyset$; `// pivot rows over` $\mathbb{F}_2$
$r \leftarrow 0$; `// current rank`
for $i \leftarrow 0$ **to** $B - 1$ **do**
  $x \xleftarrow{\$} \mathbb{F}_{2^n}^*$;
  $y \leftarrow x^{2^e-1} + x^{2^n-2} \pmod f$;
  $z \leftarrow (x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1})$;
  $R \leftarrow (1, (z_j)_{j=0}^{2n-1}, (z_j z_k)_{0 \le j < k < 2n})$;
  Reduce R against $\mathcal{P}$ by exact $\mathbb{F}_2$ row elimination;
  if $R \neq 0$ **then**
    Insert R into $\mathcal{P}$ using its leading bit as pivot;
    $r \leftarrow r + 1$;
  if $r = D_2 - 3n$ **then**
    return $\big(U = 3n,\ \dim_{\mathbb{F}_2} \mathrm{QE}(V) = 3n\big)$;
return $U = D_2 - r$; `// valid but possibly loose upper bound`

For every finite $T \subseteq V$, the $3n$ analytically established equations belong to the kernel, and hence $\mathrm{rank}_{\mathbb{F}_2} M_T \le D_2 - 3n$. Thus, the computed nullity is a sound upper bound regardless of how x is sampled. Random sampling only helps the rank reach $D_2 - 3n$; once it does, we obtain

$$3n \le \dim_{\mathbb{F}_2} \mathrm{QE}(V) \le D_2 - \mathrm{rank}_{\mathbb{F}_2} M_T = 3n.$$

n	Exponents e	Low part of $f(X)$	B	Rank	Nullity
128	$3, 7, 11$	$X^7 + X^2 + X + 1$	48,000	32,513	$384 = 3n$
192	$11, 23, 47$	$X^7 + X^2 + X + 1$	80,000	73,345	$576 = 3n$
256	$3, 7, 11, 19$	$X^{10} + X^5 + X^2 + 1$	140,000	130,561	$768 = 3n$

Table 7: Exact ranks for every recommended AIM3 S-box exponent. Here $f(X) = X^n +$ (low part).

References

[ACG+19] Martin R Albrecht, Carlos Cid, Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, and Markus Schofnegger. Algebraic cryptanalysis of STARK-friendly designs: application to MARVELlous and MiMC. In *ASIACRYPT 2019*, pages 371–397. Springer, 2019.

[AD18] Tomer Ashur and Siemen Dhooghe. MARVELlous: a STARK-Friendly Family of Cryptographic Primitives. Cryptology ePrint Archive, Paper 2018/1098, 2018. https://eprint.iacr.org/2018/1098.

[AFK22] Thomas Attema, Serge Fehr, and Michael Klooß. Fiat-Shamir Transformation of Multi-round Interactive Proofs. In Eike Kiltz and Vinod Vaikuntanathan, editors, *Theory of Cryptography*, pages 113–142. Springer, 2022.

[AGR+16] Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity. In *ASIACRYPT 2016*, pages 191–219. Springer, 2016.

[AIK+01] Kazumaro Aoki, Tetsuya Ichikawa, Masayuki Kanda, Mitsuru Matsui, Shiho Moriai, Junko Nakajima, and Toshio Tokita. Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms — Design and Analysis. In *SAC 2001*, pages 39–56. Springer, 2001.

[ARS+15] Martin R Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In *EUROCRYPT 2015*, pages 430–454. Springer, 2015.

[BB18] Gustavo Banegas and Daniel J. Bernstein. Low-Communication Parallel Quantum Multi-Target Preimage Search. In *SAC 2017*, pages 325–335. Springer, 2018.

[BBCV22] Subhadeep Banik, Khashayar Barooti, Andrea Caforio, and Serge Vaudenay. Memory-Efficient Single Data-Complexity Attacks on LowMC Using Partial Sets. Cryptology ePrint Archive, Paper 2022/688, 2022. https://eprint.iacr.org/2022/688.

[BBDV20] Subhadeep Banik, Khashayar Barooti, F. Betül Durak, and Serge Vaudenay. Cryptanalysis of LowMC instances using single plaintext/ciphertext pair. *IACR Transactions on Symmetric Cryptology*, 2020(4):130–146, Dec. 2020.

[BBP+17] Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Private Multiplication over Finite Fields. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017*, pages 397–426. Springer, 2017.

[BBVY21] Subhadeep Banik, Khashayar Barooti, Serge Vaudenay, and Hailun Yan. New Attacks on LowMC Instances with a Single Plaintext/Ciphertext Pair. In *ASIACRYPT 2021*, pages 303–331. Springer, 2021.

[BCC+10] Charles Bouillaguet, Hsieh-Chung Chen, Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, Adi Shamir, and Bo-Yin Yang. Fast Exhaustive Search for Polynomial Systems in $\mathbb{F}_2$. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010*, pages 203–218. Springer, 2010.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems-CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings 6*, pages 16–29. Springer, 2004.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In Walter Fumy, editor, *EUROCRYPT '97*, pages 37–51. Springer, 1997.

[Bei93] R. Beigel. The polynomial method in circuit complexity. In *Proceedings of the Eigth Annual Structure in Complexity Theory Conference*, pages 82–95, 1993.

[Ber05] Daniel J Bernstein. Cache-timing attacks on AES, 2005.

[Ber09] D.J. Bernstein. Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete. In *SHARCS'09 Workshop Record (Proceedings 4th Workshop on Special-purpose Hardware for Attacking Cryptograhic Systems)*, pages 105–116, 2009.

[BFS04] Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. On the complexity of Gröbner basis computation of semi-regular overdetermined algebraic equations. In *Proceedings of the International Conference on Polynomial System Solving*, pages 71–74, 2004.

[BFSS13] Magali Bardet, Jean-Charles Faugère, Bruno Salvy, and Pierre-Jean Spaenlehauer. On the complexity of solving quadratic Boolean systems. *Journal of Complexity*, 29(1):53–75, 2013.

[BGFU26] Alex Biryukov, Pablo García Fernández, and Aleksei Udovenko. Magic Pot: Cryptanalysis of Full AIM2 in the Standard and Related-/Reused-Key Settings Using New Elimination Framework. In *Advances in Cryptology – EUROCRYPT 2026*, Lecture Notes in Computer Science. Springer, 2026.

[BHNP+19] Xavier Bonnetain, Akinori Hosoyamada, María Naya-Plasencia, Yu Sasaki, and André Schrottenloher. Quantum Attacks Without Superposition Queries: The Offline Simon's Algorithm. In *ASIACRYPT 2019*, pages 552–583. Springer, 2019.

[BHT98] Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum cryptanalysis of hash and claw-free functions. In *LATIN'98: Theoretical Informatics: Third Latin American Symposium Campinas, Brazil, April 20–24, 1998 Proceedings 3*, pages 163–169. Springer, 1998.

[BJ24] Anubhab Baksi and Kyungbae Jang. *Improved Quantum Analysis of SPECK and LOWMC*, pages 91–112. Springer Nature Singapore, Singapore, 2024.

[BN20] Carsten Baum and Ariel Nof. Concretely-Efficient Zero-Knowledge Arguments for Arithmetic Circuits and Their Application to Lattice-Based Cryptography. In *PKC 2020*, pages 495–526. Springer, 2020.

[Bou22] Charles Bouillaguet. Boolean Polynomial Evaluation for the Masses. Cryptology ePrint Archive, Paper 2022/1412, 2022. https://eprint.iacr.org/2022/1412.

[BS00] Alex Biryukov and Adi Shamir. Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In *ASIACRYPT 2000*, pages 1–13. Springer, 2000.

[BSGK+21] Carsten Baum, Cyprien Delpech de Saint Guilhem, Daniel Kales, Emmanuela Orsini, Peter Scholl, and Greg Zaverucha. Banquet: Short and fast signatures from AES. In *PKC 2021*, pages 266–297. Springer, 2021.

[BWM99] Simon Blake-Wilson and Alfred Menezes. Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol. In *PKC '99*, pages 154–170. Springer, 1999.

[BXKSN21] Roland Booth, Yanhong Xu, Sabyasachi Karati, and Reihaneh Safavi-Naini. An Intermediate Secret-Guessing Attack on Hash-Based Signatures. In Toru Nakanishi and Ryo Nojima, editors, *IWSEC 2021*, pages 195–215. Springer, 2021.

[BY18] Daniel J. Bernstein and Bo-Yin Yang. Asymptotically Faster Quantum Algorithms to Solve Multivariate Quadratic Equations. In *PQCrypto 2018*, pages 487–506. Springer, 2018.

[CDG06] Nicolas T. Courtois, Blandine Debraize, and Eric Garrido. On Exact Algebraic [Non-]Immunity of S-Boxes Based on Power Functions. In *ACISP 2006*, pages 76–86. Springer, 2006.

[CDG+17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg

Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *ACM CCS 2017*, pages 1825–1842, 2017.

[CG22] Yu-Ao Chen and Xiao-Shan Gao. Quantum Algorithm for Boolean Equation Solving and Quantum Algebraic Attack on Cryptosystems. *Journal of Systems Science and Complexity*, 35(1):373–412, Feb 2022.

[CKPS00] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *EUROCRYPT 2000*, pages 392–407. Springer, 2000.

[DFM20] Jelle Don, Serge Fehr, and Christian Majenz. The Measure-and-Reprogram Technique 2.0: Multi-Round Fiat-Shamir and More. In *CRYPTO 2020*, page 602–631. Springer, 2020.

[DFMS19] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Security of the Fiat-Shamir Transformation in the Quantum Random-Oracle Model. In *CRYPTO 2019*, volume 11693 of *Lecture Notes in Computer Science*, pages 356–383. Springer, 2019.

[DFMS22a] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Efficient NIZKs and Signatures from Commit-and-Open Protocols in the QROM. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022*, pages 729–757. Springer Nature Switzerland, 2022.

[DFMS22b] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Online-Extractability in the Quantum Random-Oracle Model. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022*, pages 677–706. Springer, 2022.

[DGD+19] Debayan Das, Anupam Golder, Josef Danial, Santosh Ghosh, Arijit Raychowdhury, and Shreyas Sen. X-DeepSCA: Cross-Device Deep Learning Side Channel Attack. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2019.

[DGG+21] Jintai Ding, Vlad Gheorghiu, András Gilyén, Sean Hallgren, and Jianqiang Li. Limitations of the Macaulay matrix approach for using the HHL algorithm to solve multivariate polynomial systems. arXiv 2111.00405, 2021. https://arxiv.org/abs/2111.00405.

[Din21] Itai Dinur. Cryptanalytic Applications of the Polynomial Method for Solving Multivariate Equation Systems over GF(2). In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021*, pages 374–403. Springer, 2021.

[DKR+22] Christoph Dobraunig, Daniel Kales, Christian Rechberger, Markus Schofnegger, and Greg Zaverucha. Shorter Signatures Based on Tailor-Made Minimalist Symmetric-Key Crypto. In *ACM CCS 2022*, pages 843–857. Association of Computing Machinery, November 2022.

[DN19] Itai Dinur and Niv Nadler. Multi-target Attacks on the Picnic Signature Scheme and Related Protocols. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019*, pages 699–727. Springer, 2019.

[DNGW23] Elena Dubrova, Kalle Ngo, Joel Gärtner, and Ruize Wang. Breaking a Fifth-Order Masked Implementation of CRYSTALS-Kyber by Copy-Paste. In *Proceedings of the 10th ACM Asia Public-Key Cryptography Workshop*, APKC '23, page 10–20. Association for Computing Machinery, 2023.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael*, volume 2. Springer, 2002.

[dSGMOS19] Cyprien Delpech de Saint Guilhem, Lauren De Meyer, Emmanuela Orsini, and Nigel P Smart. BBQ: Using AES in picnic signatures. In *SAC 2019*, pages 669–692. Springer, 2019.

[EGL+20] Maria Eichlseder, Lorenzo Grassi, Reinhard Lüftenegger, Morten Øygarden, Christian Rechberger, Markus Schofnegger, and Qingju Wang. An algebraic attack on ciphers with low-degree round functions: application to full MiMC. In *ASIACRYPT 2020*, pages 477–506. Springer, 2020.

[EM97] Shimon Even and Yishay Mansour. A construction of a cipher from a single pseudorandom permutation. *Journal of Cryptology*, 10(3):151–161, Jun 1997.

[Fau99] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139(1):61–88, 1999.

[Fau02] Jean Charles Faugère. A New Efficient Algorithm for Computing Gröbner Bases without Reduction to Zero (F5). In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, ISSAC '02, page 75–83, New York, NY, USA, 2002. Association for Computing Machinery.

[FHK+17] Jean-Charles Faugère, Kelsey Horan, Delaram Kahrobaei, Marc Kaplan, Elham Kashefi, and Ludovic Perret. Fast Quantum Algorithm for Solving Multivariate Quadratic Equations. Cryptology ePrint Archive, Paper 2017/1236, 2017. https://eprint.iacr.org/2017/1236.

[Frö85] Ralf Fröberg. An Inequality for Hilbert Series of Graded Algebras. *MATHEMATICA SCANDINAVICA*, 56, Dec. 1985.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO' 86*, pages 186–194. Springer, 1987.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM (JACM)*, 33(4):792–807, 1986.

[GKR+21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A New Hash Function for Zero-Knowledge Proof Systems. In *USENIX Security 2021*, pages 519–535. USENIX Association, 2021.

[GLR+20] Lorenzo Grassi, Reinhard Lüftenegger, Christian Rechberger, Dragos Rotaru, and Markus Schofnegger. On a Generalization of Substitution-Permutation Networks: The HADES Design Strategy. In *EUROCRYPT 2020*, pages 674–704. Springer, 2020.

[GMLS02] Steven D Galbraith, John Malone-Lee, and Nigel Paul Smart. Public key signatures in the multi-user setting. *Information Processing Letters*, 83(5):263–266, 2002.

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. ZKBoo: Faster Zero-Knowledge for Boolean Circuits. In *USENIX Security 2016*, pages 1069–1083. USENIX Association, 2016.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A Digital Signature Scheme Secure against Adaptive Chosen-Message Attacks. *SIAM J. Comput.*, 17(2):281–308, apr 1988.

[Gro96] Lov K. Grover. A Fast Quantum Mechanical Algorithm for Database Search. In *ACM STOC '96*, page 212–219. Association for Computing Machinery, 1996.

[HBD+22] Andreas Hulsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kolbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Jean-Philippe Aumasson, Bas Westerbaan, and Ward Beullens. SPHINCS+. Technical report, National Institute of Standards and Technology, 2022, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[HHL09] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum Algorithm for Linear Systems of Equations. *Phys. Rev. Lett.*, 103:150502, Oct 2009.

[HHL+23] Jaeseung Han, Jae-Won Huh, Sangyub Lee, Jihoon Kwon, and Dong-Guk Han. Side-Channel and Fault Injection Attacks on The KpqC Digital Signature Candidate AIMer. In *Conference on Information Security and Cryptography Summer 2023*. Korea Institute of Information Secuity & Cryptology, 2023.

[HS18] Akinori Hosoyamada and Yu Sasaki. Cryptanalysis Against Symmetric-Key Schemes with Online Classical Queries and Offline Quantum Computations. In *CT-RSA 2018*, pages 198–218. Springer, 2018.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from Secure Multiparty Computation. In *ACM STOC 2007*, pages 21–30, 2007.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private Circuits: Securing Hardware against Probing Attacks. In Dan Boneh, editor, *CRYPTO 2003*, pages 463–481. Springer, 2003.

[IT88] Toshiya Itoh and Shigeo Tsujii. A fast algorithm for computing multiplicative inverses in gf(2m) using normal bases. *Information and Computation*, 78(3):171–177, 1988.

[JOKS24] Kyungbae Jang, Yujin Oh, Hyunji Kim, and Hwajeong Seo. Quantum Implementation of AIM: Aiming for Low-Depth. *Applied Sciences*, 14(7), 2024.

[KHS+23] Seongkwang Kim, Jincheol Ha, Mincheol Son, Byeonghak Lee, Dukjae Moon, Joohee Lee, Sangyub Lee, Jihoon Kwon, Jihoon Cho, Hyojin Yoon, and Jooyoung Lee. AIM: Symmetric Primitive for Shorter Signatures with Stronger Security. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, page 401–415. Association for Computing Machinery, 2023.

[KHSL24] Seongkwang Kim, Jincheol Ha, Mincheol Son, and Byeonghak Lee. Efficacy and Mitigation of the Cryptanalysis on AIM. Cryptology ePrint Archive, Paper 2023/1474, 2024. https://eprint.iacr.org/2023/1474.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In *CRYPTO' 99*, pages 388–397. Springer, 1999.

[KKW18] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures. In *ACM CCS 2018*, pages 525–537. ACM, 2018.

[KM10] Hidenori Kuwakado and Masakatu Morii. Quantum distinguisher between the 3-round Feistel cipher and the random permutation.

In *2010 IEEE International Symposium on Information Theory*, pages 2682–2685, 2010.

[KM12] Hidenori Kuwakado and Masakatu Morii. Security on the quantum-type Even-Mansour cipher. In *2012 International Symposium on Information Theory and its Applications*, pages 312–316, 2012.

[KM13] Neal Koblitz and Alfred Menezes. Another look at security definitions. *Advances in Mathematics of Communications*, 7(1):1–38, 2013.

[Koc96] Paul C Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *CRYPTO'96*, pages 104–113. Springer, 1996.

[KR01] Joe Kilian and Phillip Rogaway. How to Protect DES Against Exhaustive Key Search (an Analysis of DESX). *Journal of Cryptology*, 14(1):17–35, Jan 2001.

[KR19] Yael Tauman Kalai and Leonid Reyzin. *A Survey of Leakage-Resilient Cryptography*, page 727–794. Association for Computing Machinery, New York, NY, USA, 2019.

[KZ22] Daniel Kales and Greg Zaverucha. Efficient Lifting for Shorter Zero-Knowledge Proofs and Post-Quantum Signatures. Cryptology ePrint Archive, Paper 2022/588, 2022. https://eprint.iacr.org/2022/588.

[LDK+22] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2022, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[LIM21] Fukang Liu, Takanori Isobe, and Willi Meier. Low-Memory Algebraic Attacks on Round-Reduced LowMC. Cryptology ePrint Archive, Paper 2021/255, 2021. https://eprint.iacr.org/2021/255.

[LM17] Gregor Leander and Alexander May. Grover Meets Simon – Quantumly Attacking the FX-construction. In *ASIACRYPT 2017*, pages 161–178. Springer, 2017.

[LMOM23] Fukang Liu, Mohammad Mahzoun, Morten Øygarden, and Willi Meier. Algebraic Attacks on RAIN and AIM Using Equivalent Representations. *IACR Transactions on Symmetric Cryptology*, 2023(4):166–186, Dec. 2023.

[LMSI22] Fukang Liu, Willi Meier, Santanu Sarkar, and Takanori Isobe. New Low-Memory Algebraic Attacks on LowMC in the Picnic Setting. *IACR Transactions on Symmetric Cryptology*, 2022(3):102–122, Sep. 2022.

[LPT+17] Daniel Lokshtanov, Ramamohan Paturi, Suguru Tamaki, Ryan Williams, and Huacheng Yu. Beating Brute Force for Systems of Polynomial Equations over Finite Fields. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2190–2202. SIAM, 2017.

[LR86] Michael Luby and Charles Rackoff. How to Construct Pseudo-random Permutations from Pseudo-random Functions. In *CRYPTO '85*, pages 447–447. Springer, 1986.

[LZ19] Qipeng Liu and Mark Zhandry. Revisiting Post-quantum Fiat-Shamir. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019*, pages 326–355. Springer, 2019.

[MS04] Alfred Menezes and Nigel Smart. Security of Signature Schemes in a Multi-User Setting. *Designs, Codes and Cryptography*, 33(3):261–274, Nov 2004.

[NIS15] NIST. SHA-3 standard: Permutation-based hash and extendable-output functions, 2015. FIPS PUB 202.

[NIS22] NIST. Call for Additional Digital Signature Schemes for the Post-Quantum Cryptography Standardization Process. Technical report, National Institute of Standards and Technology, 2022, 2022. available at https://csrc.nist.gov/projects/pqc-dig-sig.

[PFH+22] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2022, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[PKC22] Ray Perlner, John Kelsey, and David Cooper. Breaking Category Five SPHINCS+ with SHA-256. In Jung Hee Cheon and Thomas Johansson, editors, *PQCrypto 2022*, pages 501–522. Springer, 2022.

[QS01] Jean-Jacques Quisquater and David Samyde. ElectroMagnetic Analysis (EMA): Measures and Counter-measures for Smart Cards. In Isabelle Attali and Thomas Jensen, editors, *Smart Card Programming and Security*, pages 200–210. Springer, 2001.

[SCL+25] Yimeng Sun, Shiyao Chen, Guowei Liu, Meiqin Wang, and Chao Niu. High Exponents May Not Suffice to Patch AIM (On Attacks, Weak Parameters, and Patches for AIM2). Cryptology ePrint Archive, Paper 2025/2272, 2025.

[Sim97] Daniel R. Simon. On the Power of Quantum Computation. *SIAM Journal on Computing*, 26(5):1474–1483, 1997.

[SS21] Jan Ferdinand Sauer and Alan Szepieneic. SoK: Gröbner Basis Algorithms for Arithmetization Oriented Ciphers. Cryptology ePrint Archive, Paper 2021/870, 2021. https://eprint.iacr.org/2021/870.

[SSA+07] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-Bit Blockcipher CLEFIA (Extended Abstract). In *FSE 2007*, pages 181–195. Springer, 2007.

[WD20] Huanyu Wang and Elena Dubrova. Tandem Deep Learning Side-Channel Attack Against FPGA Implementation of AES. In *2020 IEEE International Symposium on Smart Electronic Systems (iSES) (Formerly iNiS)*, pages 147–150, 2020.

[Wil14] Richard Ryan Williams. The Polynomial Method in Circuit Complexity Applied to Algorithm Design (Invited Talk). In Venkatesh Raman and S. P. Suresh, editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS 2014)*, volume 29 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 47–60, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[YAG21] Mahmoud Yehia, Riham AlTawy, and T. Aaron Gulliver. Security Analysis of DGM and GM Group Signature Schemes Instantiated with XMSS-T. In Yu Yu and Moti Yung, editors, *Information Security and Cryptology*, pages 61–81. Springer, 2021.

[Zha12] Mark Zhandry. How to Construct Quantum Random Functions. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 679–687, 2012.

[ZWY+23] Kaiyi Zhang, Qingju Wang, Yu Yu, Chun Guo, and Hongrui Cui. Algebraic Attacks on Round-Reduced Rain and Full AIM-III. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023*, pages 285–310. Springer, 2023.