# The AIMer Signature Scheme


**Authors**

Jihoon Cho (Samsung SDS)
Jincheol Ha (CryptoLab)
Seongkwang Kim (Samsung SDS)
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):

`d7bbdb28 e70dad02 1da2057b c0b48025`

`85770cbc 2d7449ac fcd770d8 bc624521`


Friday 30th January, 2026

# Contents

# 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 ($\mathsf{seed}_k$) 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++. The AIM function fully exploits the optimization techniques of BN++ using *repeated multipliers* for checking multiplication triples and *locally computed output shares* to reduce the overall signature size.

However, recent studies have identified certain algebraic vulnerabilities in AIM [LMOM23, ZWY+23]. The most powerful attack among them is a fast exhaustive search attack by Liu et al, which exploits the property that AIM allows a low-degree system of equations in a moderate number of Boolean variables. They demonstrated potential security degradation of up to 12 bits compared to the existing analysis on the complexity of exhaustive search on AIM [KHSL24].

To mitigate such attacks, Kim et al. proposed a new symmetric primitive AIM2 [KHSL24]. AIM2 has a similar structure with AIM except with minor changes: it employs the inverse Mersenne S-boxes, which are the inverse functions of Mersenne S-boxes. The inverse Mersenne S-boxes with higher exponents make it harder to establish a low-degree system of equations in a moderate number of Boolean variables. Second, a distinct constant is added to the input to each S-box, which makes it hard to establish a system of equations using a common variable fed to all the S-boxes. Overall, AIM2 provides stronger security against recent attacks on AIM, at the cost of small performance overhead. In AIMer version 2.0, we mount AIM2 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. The key generation is simply a computation of AIM2, which proceeds as follows.

1. A plaintext pt and a tweak iv are sampled uniformly at random.
2. $\mathsf{ct} = \mathsf{AIM2}(\mathsf{pt}, \mathsf{iv})$ is computed.
3. 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 AIM2. 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 AIM2 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 increase 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 substring from bit-position $a$ (inclusive) to $b$ (exclusive), respectively. Bitlength of a bitstring $a$ is deonted 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

$$\texttt{0x0603}[0:7] = 1100\ 0000\ 0110\ 0000[0:7] = \texttt{0x03}.$$

| | |
|---|---|
| $\lambda$ | Security parameter |
| $n$ | Input/output bit-length of S-boxes in AIM2 (which is always same as $\lambda$) |
| $\ell$ | Number of S-boxes in front of the linear layer in AIM2 |
| $\tau$ | Number of the parallel repetitions in NIZKPoK |
| $N$ | Number of the parties in NIZKPoK (which is always a power-of-two in ver. 2.0) |

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

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

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

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

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

For the proof of the one-wayness of $\mathsf{AIM}2$, we will use the information-theoretic notion of *everywhere preimage resistance* given in [RS04] by assuming that $\mathsf{AIM}2$ is based on public random permutations. We refer to Section 5.2 for the formal definition of everywhere preimage resistance.

EUF-KO SECURITY. The existential unforgeability of a signature scheme $\Pi$ 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

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

where $\lambda$ is the security parameter.

EUF-CMA SECURITY. The existential unforgeability of a signature scheme $\Pi$ 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

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

where $\lambda$ 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 assign 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 AIM2 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 $\lambda$ is the security parameter, $\tau$ 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].

OPTIMIZATION TECHNIQUES. If multiplication triples use an identical multiplier in common, for example, given $(x_0, y, z_0)$ and $(x_1, y, z_1)$, then the corresponding $\alpha$ values can be batched to reduce the signature size. Instead of computing $\alpha_0 = \epsilon_0 \cdot x_0 + a_0$ and $\alpha_1 = \epsilon_1 \cdot x_1 + a_1$, $\alpha = \epsilon_0 \cdot x_0 + \epsilon_1 \cdot x_1 + a$ is computed, and $v$ is defined as

$$v = \epsilon_0 \cdot z_0 + \epsilon_1 \cdot z_1 - \alpha \cdot y + c,$$

where $c = a \cdot y$. This technique is called *repeated multiplier* technique. Our symmetric primitive design allows us to take full advantage of this technique to reduce the number of $\alpha$ values in each repetition of the protocol.

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 $\Sigma$-protocol, a three-round interactive proof of knowledge. Let $R$ be a relation such that, for a given $x$, it is difficult to find an $w$ 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 $\Sigma$-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 $\Sigma$-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 $(x, \text{com})$. Denoting the random oracle as $\mathcal{RO}$, the challenge step of the above procedure is replaced by ch $\leftarrow \mathcal{RO}(x, \mathrm{com})$. 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 $\{f_i(x_0, \dots, x_{n-1}) = 0\}_{i=0}^{m-1}$ in $n$ Boolean variables. Let $d_i$ denote the degree of $f_i$ for $i = 0, 1, \dots, m-1$. 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 $d_{\mathrm{reg}}$, the complexity is

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

ignoring the constant factor, where $\omega$ 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

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

where $d_{\text{reg}}(n,k)$ 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 AIM2

## 3.1 Specification

AIM2 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 $(e_0, \ldots, e_{\ell-1}, e_*) \in \mathbb{Z}^{\ell+1}$, AIM2 : $\mathbb{F}_{2^n} \times \{0,1\}^n \to \mathbb{F}_{2^n}$ is defined by

$$\mathsf{AIM2}(\mathsf{pt}, \mathsf{iv}) = \mathsf{Mer}[e_*] \circ \mathsf{Lin}[\mathsf{iv}] \circ \mathsf{Mer}[e_0, \ldots, e_{\ell-1}]^{-1} \circ \mathsf{AddConst}(\mathsf{pt}) \oplus \mathsf{pt}$$

where each function will be described below. See Figure 1 for the pictorial description of AIM2 with $\ell = 3$.

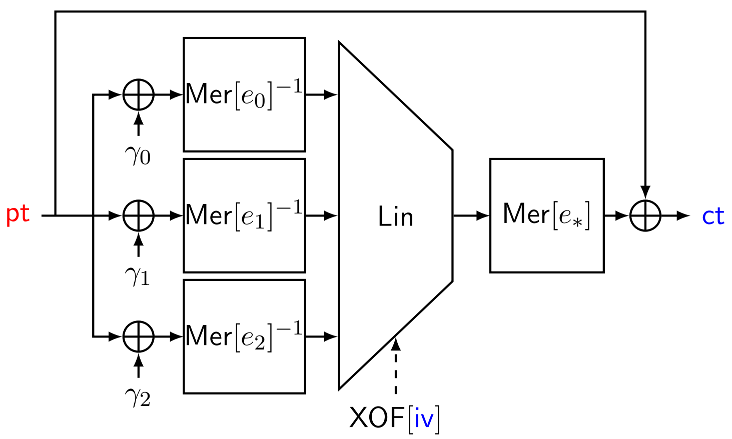


Figure 1: The AIM2-V one-way function with $\ell = 3$. The input pt (in red) is the secret key of the signature scheme, and (iv, ct) (in blue) is the corresponding public key.

NON-LINEAR COMPONENTS. AIM2 uses two types of S-boxes: Mersenne S-box $\mathsf{Mer}[e]$, and its inverse $\mathsf{Mer}[e]^{-1}$. These two S-boxes are defined by exponentiation over a large field as follows. For $x \in \mathbb{F}_{2^n}$,

$$\mathsf{Mer}[e](x) = x^{2^e-1},$$
$$\mathsf{Mer}[e]^{-1}(x) = x^{\bar{e}} \quad \text{where } \bar{e} = (2^e-1)^{-1} \pmod{2^n-1}$$

for some $e$. The exponents $e$ in AIM2 are selected for $\mathsf{Mer}[e]^{-1}$ to have $3n$ quadratic equations. We remark that the exponents $e$ are chosen such that $\gcd(e, n) = 1$, and hence the inverse exponent $\bar{e}$ is well-defined. As an extension, $\mathsf{Mer}[e_0, \ldots, e_{\ell-1}]^{-1} : \mathbb{F}_{2^n}^{\ell} \to \mathbb{F}_{2^n}^{\ell}$ is defined by

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

LINEAR COMPONENTS. AIM2 includes three types of linear components: constant addition, an affine layer, and feed-forward. For fixed constants $\gamma_0, \ldots, \gamma_{\ell-1}$, $\mathsf{AddConst} : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}^{\ell}$ is defined by

$$\mathsf{AddConst}(x) = (x + \gamma_0) \| \ldots \| (x + \gamma_{\ell-1})$$

where the constants are defined in Table 2.

| | | |
|---|---|---|
| AIM2-I | $\gamma_0$ | 0x243f6a88 85a308d3 13198a2e 03707344 |
| | $\gamma_1$ | 0xa4093822 299f31d0 082efa98 ec4e6c89 |
| AIM2-III | $\gamma_0$ | 0x452821e6 38d01377 be5466cf 34e90c6c c0ac29b7 c97c50dd |
| | $\gamma_1$ | 0x3f84d5b5 b5470917 9216d5d9 8979fb1b d1310ba6 98dfb5ac |
| AIM2-V | $\gamma_0$ | 0x2ffd72db d01adfb7 b8e1afed 6a267e96 ba7c9045 f12c7f99 24a19947 b3916cf7 |
| | $\gamma_1$ | 0x0801f2e2 858efc16 636920d8 71574e69 a458fea3 f4933d7e 0d95748f 728eb658 |
| | $\gamma_2$ | 0x718bcd58 82154aee 7b54a41d c25a59b5 9c30d539 2af26013 c5d1b023 286085f0 |

Table 2: Constants $\gamma_0, \ldots, \gamma_{\ell-1}$ in AddConst are written in hexadecimal. These constants are taken from the numbers below the decimal point of the $\pi$ ratio.

The affine layer in AIM2 consists of multiplication by an $n \times \ell n$ random binary matrix $A_{\mathsf{iv}}$ and addition by a random constant $b_{\mathsf{iv}} \in \mathbb{F}_2^n$. The matrix

$$A_{\mathsf{iv}} = \left[A_{\mathsf{iv},0} \middle| \ldots \middle| A_{\mathsf{iv},\ell-1}\right] \in (\mathbb{F}_2^{n \times n})^{\ell}$$

is composed of $\ell$ random invertible matrices $A_{\mathsf{iv},i}$. The matrix $A_{\mathsf{iv}}$ and the vector $b_{\mathsf{iv}}$ are generated by an extendable-output function (XOF) with the initial vector iv. Each matrix $A_{\mathsf{iv},i}$ can be equivalently represented by a linearized polynomial $L_{\mathsf{iv},i}$ on $\mathbb{F}_{2^n}$. For $x = (x_0, \ldots, x_{\ell-1}) \in (\mathbb{F}_{2^n})^{\ell}$,

$$\mathsf{Lin}[\mathsf{iv}](x) = \sum_{0 \le i \le \ell-1} L_{\mathsf{iv},i}(x_i) \oplus b_{\mathsf{iv}}.$$

By abuse of notation, we will write $Ax$ to denote $\sum_{0 \le i \le \ell-1} L_{\mathsf{iv},i}(x_i)$. Feed-forward operation, which is addition by the input itself, makes the entire function non-invertible.

RECOMMENDED PARAMETERS. Table 3 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.

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

| Scheme | $\lambda$ | $n$ | $\ell$ | $e_0$ | $e_1$ | $e_2$ | $e_*$ |
|---|---|---|---|---|---|---|---|
| AIM2-I | 128 | 128 | 2 | 49 | 91 | - | 3 |
| AIM2-III | 192 | 192 | 2 | 17 | 47 | - | 5 |
| AIM2-V | 256 | 256 | 3 | 11 | 141 | 7 | 3 |

Table 3: Recommended sets of parameters of AIM2.

## 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 6.3.2.

With the above observation, we tried to find an invertible S-box of high-degree which is moderately resistant to differential/linear cryptanalysis as well as implicitly quadratic, and *producing only a small number of quadratic equations*. Since our attack model does not allow multiple queries to a single instance of AIM2, we allow a relaxed condition on the DC/LC resistance, not being necessarily maximal. As a family of S-boxes that beautifully fit all the conditions, we choose a family of Mersenne S-boxes; they are exponentiation by Mersenne numbers $2^e - 1$ such that $\gcd(n, e) = 1$, are invertible, are of high-degree, need only one multiplication for its proof, produce only $3n$ Boolean quadratic equations with its input and output, and provide moderate DC/LC resistance. Furthermore, when the implicit equation $xy = x^{2^e}$ of a Mersenne S-box is computed in the BN++ proof system, it is not required to broadcast the output share since the output of multiplication $x^{2^e}$ can be locally computed from the share of $x$. AIM2 uses Mersenne S-boxes in the forward and backward directions. The inverse Mersenne S-boxes enjoy the same algebraic properties as Mersenne S-boxes, while they result in a harder polynomial system as a whole.

REPETITIVE STRUCTURE. The efficiency of the BN++ proof system partially comes from the optimization technique using *repeated multipliers*. When a multiplier is repeated in multiple equations to prove, the proof can be done in a batched way, reducing the overall signature size. In order to maximize the advantage of repeated multipliers, we put S-boxes at the first round in parallel and an additional S-box at the second round with feed-forward to its output to make the implicit equations from the S-boxes share the same multiplier pt (with constant differences).

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 AIM2. 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 of 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 SHAKE128$(x,d)$ and SHAKE256$(x,d)$ means 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 $H_i$. For example, $H_0$ uses $0$ 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,

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

where

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

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 $1$.
- $H_2$: $\{0,1\}^{2\lambda} \times \{0,1\}^\lambda \times ((\mathbb{F}_{2^n})^N \times (\mathbb{F}_{2^n})^N)^\tau \to \{0,1\}^{2\lambda}$, 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 $4$.
- $H_5$: $\{0,1\}^\lambda \times [\tau] \times [N] \times \{0,1\}^\lambda \to \{0,1\}^{2\lambda} \times \mathbb{F}_{2^n} \times (\mathbb{F}_{2^n})^\ell \times \mathbb{F}_{2^n} \times \mathbb{F}_{2^n}$, 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 use 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. There are three following algorithms related with evaluation of 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 AIM2 Functions

AIM2 is the symmetric primitive which is zero-knowledge proved in AIMer. AIM2 is computed in a plain manner in the AIM2 algorithm for key generation, and computed in a secret-shared manner in the AIM2_MPC for signing and verification. Generally, matrix multiplication can be performed more efficiently if the matrix is provided in its transposed form. Consequently, the AIM2_GenerateLinear algorithm in AIMer is deliberately designed to directly produce matrices in their transposed form.

- AIM2_GenerateLinear: $\{0,1\}^\lambda \to (\mathbb{F}_2^{n\times n})^\ell \times \mathbb{F}_2^{n\times 1}$, generate the linear components in AIM2.
- AIM2: $\{0,1\}^n \times \{0,1\}^\lambda \to \{0,1\}^n$, the AIM2 one-way function.
- AIM2_MPC: $(\mathbb{F}_2^{n\times n})^\ell \times \mathbb{F}_2^n \times \mathbb{F}_{2^n} \times (\mathbb{F}_{2^n})^\ell \to (\mathbb{F}_{2^n} \times \mathbb{F}_{2^n})^{\ell+1}$, a function to generate multiplication inputs/outputs for MPC simulation of AIM2.
- AIM2_SboxOutputs: $\mathbb{F}_2^n \to (\mathbb{F}_{2^n})^\ell$, a function to generate outputs for first-round S-boxes.

The detailed specifications are in Figure 3.

## 4.2 Signature Scheme

The AIMer signature scheme $\Pi =$ (AIMer_keygen, AIMer_sign, AIMer_verify) consists of key generation, signing, and verification algorithms. Each algorithm calls the corresponding internal algorithm (AIMer_keygen_internal, AIMer_sign_internal, 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.

ExpandTree(salt, $k$, seed)

Initialize tree: nodes $\leftarrow (0^\lambda)^{2N-1}$.
Set the root seed: nodes$[0] \leftarrow$ seed.
**for** $i = 0, \ldots, N-2$ **do**
  nodes$[2i+1]$, nodes$[2i+2] \leftarrow H_4($salt, $k, i,$ nodes$[i])$
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 node: path$[d] \leftarrow$ nodes$[(j \oplus 1) - 1]$
  Move to parent node: $j \leftarrow \lfloor (j-1)/2 \rfloor$
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**
  sib $\leftarrow (j \oplus 1) - 1$
  nodes[sib] $\leftarrow$ path$[d]$
  `// Expand parital tree`
  **for** $u = 0, \ldots, d-1$ **do**
    **for** $v = 0, \ldots, 2^u - 1$ **do**
      $w \leftarrow 2^u \cdot$ sib $+ v$
      nodes$[2w+1]$, nodes$[2w+2] \leftarrow H_4($salt, $k, w,$ nodes$[w])$
  Move to parent: $j \leftarrow \lfloor (j-1)/2 \rfloor$
Output nodes$[N-1 : 2N-1]$.

Figure 2: Algorithms for GGM tree evaluation.

AIM2_GenerateLinear(iv)

Initialize linear components:
- for $j \in [\ell]$, $L_j \leftarrow 0^{n \times n}$.
- for $j \in [\ell]$, $U_j \leftarrow 0^{n \times n}$.
- $b \leftarrow 0^n$

$\mathsf{tape} \leftarrow \mathsf{SHAKE}_\lambda(\mathsf{iv}, \ell n^2 + n)$.

**for** $j \in [\ell]$ **do**
- **for** $r \in [n]$ **do**
  - **for** $c \in [n]$ **do**
    - **if** $c < r$ **then**
      - Set $L_j[c][r] \leftarrow \mathsf{tape}[jn^2 + rn + c]$.
    - **else if** $c = r$ **then**
      - Set $L_j[c][r] \leftarrow 1$.
      - Set $U_j[c][r] \leftarrow 1$.
    - **else**
      - Set $U_j[c][r] \leftarrow \mathsf{tape}[jn^2 + rn + c]$.
- Set $A_j \leftarrow L_j \cdot U_j$

for $r \in [n]$, $b[r] \leftarrow \mathsf{tape}[\ell n^2 + r]$.

Output $((A_j)_{j \in [\ell]}, b)$.

AIM2(pt, iv)

Sample linear components: $((A_{\mathsf{iv},j})_{j \in [\ell]}, b_{\mathsf{iv}}) \leftarrow$ GenerateLinear(iv).

$t_* \leftarrow b_{\mathsf{iv}}$

**for** $j \in [\ell]$ **do**
- $t_j \leftarrow \mathsf{Mer}[e_j]^{-1}(\mathsf{pt} + \gamma_j)$.
- $t_* \leftarrow t_* + A_{\mathsf{iv},j} \cdot t_j$.

$\mathsf{ct} \leftarrow \mathsf{Mer}[e_*](t_*) + \mathsf{pt}$

Output ct.

AIM2_MPC($(A_j)_{j \in [\ell]}, b, \mathsf{ct}, (t_j^{(\cdot)})_{j \in [\ell]}$) - Run the MPC simulation

**for** $j \in [\ell]$ **do**
- Set $x_j^{(\cdot)} \leftarrow t_j^{(\cdot)}$;
- Set $z_j^{(\cdot)} \leftarrow (x_j^{(\cdot)})^{2^{e_j}} + \gamma_j \cdot x_j^{(\cdot)}$.

Set $x_\ell^{(\cdot)} \leftarrow \sum_{j \in [\ell]} A_j \cdot x_j^{(\cdot)} + b$.

Set $z_\ell^{(\cdot)} \leftarrow (x_\ell^{(\cdot)})^{2^{e_*}} + \mathsf{ct} \cdot x_\ell^{(\cdot)}$.

Output $(x_j^{(\cdot)}, z_j^{(\cdot)})_{j \in [\ell+1]}$.

AIM2_SboxOutputs(pt)

**for** $j \in [\ell]$ **do**
- $t_j \leftarrow \mathsf{Mer}[e_j]^{-1}(\mathsf{pt} + \gamma_j)$.

Output $(t_j)_{j \in [\ell]}$.

Figure 3: Algorithms used for AIM2 evaluation.

- AIMer_keygen$(1^\lambda) \rightarrow (sk, pk)$ : Sample uniform random pt $\leftarrow_\$ \mathbb{F}_{2^n}$, and iv $\leftarrow_\$ \{0,1\}^n$. Compute ct $\leftarrow$ AIM2(pt, iv) as described in Section 3, and set the public key $pk \leftarrow (\text{iv}, \text{ct}) \in \{0,1\}^n \times \mathbb{F}_{2^n}$ and the private key $sk \leftarrow (\text{pt}, \text{iv}, \text{ct}) \in \mathbb{F}_{2^n} \times \{0,1\}^n \times \mathbb{F}_{2^n}$.

- AIMer_sign$(sk, M, \text{ctx}) \rightarrow \sigma$ : Take as input a private key $sk = (\text{pt}, \text{iv}, \text{ct})$, a message $m \in \{0,1\}^*$, and a context string ctx, and compute the zero-knowledge proof $\pi$ for the AIM2 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 $|\sigma| = (5 + (\log_2 N + \ell + 5)\tau)\lambda$

- AIMer_verify$(pk, M, \sigma, \text{ctx}) \rightarrow$ Accept or Reject : Take as input a public key $pk = (\text{iv}, \text{ct})$, a message $m$, a signature $\sigma$, and a context string ctx, and conduct the verification of NIZKPoK for the AIM2 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 not 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$(1^\lambda)$ initiated by generating two random $\lambda$-bit sequences, pt and iv. The secret key pt is encrypted under the AIM2 function using iv as the initialization vector, resulting in the ciphertext ct. 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$(1^\lambda)$ - AIMer signature scheme, key generation algorithm

```
Sample pt ←$ {0,1}^λ.
Sample iv ←$ {0,1}^λ.
if pt = null or iv = null then
    Abort.
return AIMer_keygen_internal(pt, iv)
```

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

```
Set ct ← AIM2(pt, iv).
Set sk ← (pt, iv, ct), pk ← (iv, ct).
Output (sk, pk).
```

### 4.2.2 Signature Generation

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

PHASE 1: COMMITTING TO THE SEEDS AND THE EXECUTION VIEWS OF THE PARTIES. It first pre-hash the message, and computes an instance of AIM2 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 compute 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$(sk, M, \mathsf{ctx})$ - AIMer signature scheme, signing algorithm.

---

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

---

### 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.

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

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

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

Compute the first $\ell$ S-boxes' outputs: $(t_j)_{j\in[\ell]} \leftarrow \textsf{AIM2\_SboxOutputs}(\textsf{pt})$

Derive the AIM2 linear components $(A_{\textsf{iv},j})_{j\in[\ell]} \in (\mathbb{F}_2^{n\times n})^\ell$ and $b_{\textsf{iv}} \in \mathbb{F}_2^n$: $((A_{\textsf{iv},j})_{j\in[\ell]}, b_{\textsf{iv}}) \leftarrow \textsf{AIM2\_GenerateLinear}(\textsf{iv})$

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

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

- Compute parties' seeds:
  - $\textsf{nodes}_k \leftarrow \textsf{ExpandTree}(\textsf{salt}, k, \textsf{seed}_k)$;
  - $\textsf{seed}_k^{(0)}, \ldots, \textsf{seed}_k^{(N-1)} \leftarrow \textsf{nodes}_k[N-1 : 2N-1]$.
- **for** *each party* $i \in [N]$ **do**
  - Commit to the seed and expand random tape: $(\textsf{com}_k^{(i)}, \textsf{pt}_k^{(i)}, (t_{k,j}^{(i)})_{j\in[\ell]}, a_k^{(i)}, c_k^{(i)}) \leftarrow H_5(\textsf{salt}, k, i, \textsf{seed}_k^{(i)})$.
- Compute offsets and adjust last shares:
  - $\Delta\textsf{pt}_k \leftarrow \textsf{pt} - \sum_i \textsf{pt}_k^{(i)}$, $\textsf{pt}_k^{(N-1)} \leftarrow \textsf{pt}_k^{(N-1)} + \Delta\textsf{pt}_k$;
  - for $j \in [\ell]$, $\Delta t_{k,j} \leftarrow t_j - \sum_i t_{k,j}^{(i)}$, $t_{k,j}^{(N-1)} \leftarrow t_{k,j}^{(N-1)} + \Delta t_{k,j}$;
  - $\Delta c_k \leftarrow \sum_i a_k^{(i)} \cdot \textsf{pt} - \sum_i c_k^{(i)}$, $c_k^{(N-1)} \leftarrow c_k^{(N-1)} + \Delta c_k$.
- **for** *each party* $i \in [N]$ **do**
  - Set $b \leftarrow b_{\textsf{iv}}$ if $i = N-1$ or set $b \leftarrow 0^n$ otherwise.
  - Run the MPC simulation and prepare the multiplication check inputs: $(x_{k,j}^{(i)}, z_{k,j}^{(i)})_{j\in[\ell+1]} \leftarrow \textsf{AIM2\_MPC}((A_{\textsf{iv},j})_{j\in[\ell]}, b, \textsf{ct}, (t_{k,j}^{(i)})_{j\in[\ell]})$

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

`// Phase 2: Challenging the multiplication checking protocol.`

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

Expand hash: $((\epsilon_{k,j})_{j\in[\ell+1]})_{k\in[\tau]} \leftarrow \textsf{ExpandH1}(h_1)$ 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 $i \in [N]$, $\alpha_k^{(i)} \leftarrow a_k^{(i)} + \sum_{j\in[\ell+1]} x_{k,j}^{(i)} \cdot \epsilon_{k,j}$.
  - Set $\alpha_k = \sum_{i\in[N]} \alpha_k^{(i)}$.
  - for $i \in [N]$, $v_k^{(i)} \leftarrow c_k^{(i)} + \sum_{j\in[\ell+1]} z_{k,j}^{(i)} \cdot \epsilon_{k,j} - \alpha_k \cdot \textsf{pt}_k^{(i)}$.

Set $\sigma_2 \leftarrow (\textsf{salt}, ((\alpha_k^{(i)})_{i\in[N]}, (v_k^{(i)})_{i\in[N]})_{k\in[\tau]})$.

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

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

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

`// Phase 5: Opening the views of the MPC and checking protocols.`

**for** *each repetition* $k$ **do**

- $\textsf{path}_k \leftarrow \textsf{RevealAllBut}(\textsf{nodes}_k, \bar{i}_k)$.

Output $\sigma \leftarrow \Big(\textsf{salt}, h_1, h_2, (\textsf{path}_k, \textsf{com}_k^{(\bar{i}_k)}, \Delta\textsf{pt}_k, (\Delta t_{k,j})_{j\in[\ell]}, \Delta c_k, \alpha_k^{(\bar{i}_k)})_{k\in[\tau]}\Big)$.

First, given a public key, it computes the hash value of the message and an instance of AIM2. 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 re-generate commitments and tapes.
- From the commitments and tapes, recomputes $\sigma_1$ 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 $\sigma_2$ 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** $|\mathsf{ctx}| > 2040$ **then**
  **Abort.**
$M' \leftarrow (|\mathsf{ctx}|/8)\|\mathsf{ctx}\|M$
`// Byte-length of ctx is represented in a byte.`
**return** AIMer_verify_internal$(pk, M', \sigma)$

---

### 4.3 Recommended Parameters

For security levels L1, L3, and L5, recommended sets of parameters are given in Table 4. For each value of security parameter $\lambda$, the corresponding sets of parameters are expected to provide $\lambda$-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 security, 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

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

Parse $\sigma$ as $\left(\mathsf{salt}, h_1, h_2, \left(\mathsf{path}_k, \mathsf{com}_k^{(\bar{i}_k)}, \Delta\mathsf{pt}_k, (\Delta t_{k,j})_{j\in[\ell]}, \Delta c_k, \alpha_k^{(\bar{i}_k)}\right)_{k\in[\tau]}\right)$.

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

Derive the AIM2 linear components $(A_{\mathsf{iv},j})_{j\in[\ell]} \in (\mathbb{F}_2^{n\times n})^\ell$ and $b_{\mathsf{iv}} \in \mathbb{F}_2^n$: $((A_{\mathsf{iv},j})_{j\in[\ell]}, b_{\mathsf{iv}}) \leftarrow$ AIM2_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 $(\mathsf{com}_k^{(i)}, \mathsf{pt}_k^{(i)}, (t_{k,j}^{(i)})_{j\in[\ell]}, a_k^{(i)}, c_k^{(i)}) \leftarrow H_5(\mathsf{salt}, k, i, \mathsf{seed}_k^{(i)})$.

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

      Adjust last share:

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

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

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

    Set $b \leftarrow b_{\mathsf{iv}}$ if $i = N - 1$ or set $b \leftarrow 0^n$ otherwise.

    Run the MPC simulation and prepare the multiplication check inputs: $(x_{k,j}^{(i)}, z_{k,j}^{(i)})_{j\in[\ell+1]} \leftarrow$ AIM2_MPC$((A_{\mathsf{iv},j})_{j\in[\ell]}, b, \mathsf{ct}, (t_{k,j}^{(i)})_{j\in[\ell]})$

    Simulate the multiplication checking protocol as in Section 2.3:

      for $i \in [N] \setminus \{\bar{i}_k\}$, $\alpha_k^{(i)} \leftarrow a_k^{(i)} + \sum_{j\in[\ell+1]} x_{k,j}^{(i)} \cdot \epsilon_{k,j}$.

      Set $\alpha_k = \sum_{i\in[N]} \alpha_k^{(i)}$.

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

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

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

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

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

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 | $\lambda$ | $n$ | $\ell$ | $e_1$ | $e_2$ | $e_3$ | $e_*$ | Hash | $N$ | $\tau$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| L1 | AIMer-128f | 128 | 128 | 2 | 49 | 91 | - | 3 | SHAKE128 | 16 | 33 |
| | AIMer-128s | 128 | 128 | 2 | 49 | 91 | - | 3 | SHAKE128 | 256 | 17 |
| L3 | AIMer-192f | 192 | 192 | 2 | 17 | 47 | - | 5 | SHAKE256 | 16 | 49 |
| | AIMer-192s | 192 | 192 | 2 | 17 | 47 | - | 5 | SHAKE256 | 256 | 25 |
| L5 | AIMer-256f | 256 | 256 | 3 | 11 | 141 | 7 | 3 | SHAKE256 | 16 | 65 |
| | AIMer-256s | 256 | 256 | 3 | 11 | 141 | 7 | 3 | SHAKE256 | 256 | 33 |

Table 4: The recommended parameters for AIMer.

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 $(N, \tau, \lambda)$ 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*

$$\mathbf{Adv}^{\mathsf{euf\text{-}ko}}_{\mathsf{AIMer}}(\mathcal{A}) \leq \frac{(\tau N + 1)Q^2}{2^{2\lambda}} + \Pr[X + Y = \tau] + \mathbf{Adv}^{\mathsf{owf}}_{\mathsf{AIM2}}(\mathcal{B}),$$

*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 AIM2 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 AIM2 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 $\mathsf{pt}$ which is the secret key and the multiplication triples, $\mathcal{B}$ check 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 $\mathsf{pt}$. 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.

- $H_0$, $H_4$, $\mathsf{ExpandH1}$ and $\mathsf{ExpandH2}$ are not programmed.

After $\mathcal{A}$ terminates, $\mathcal{B}$ checks whether there is $\mathsf{pt}_k \in \mathcal{T}_{\mathsf{in}}$ satisfying $\mathsf{AIM2}(\mathsf{iv}, \mathsf{pt}_k) = \mathsf{ct}$. If $\mathcal{B}$ finds a match $\mathsf{pt}_k$, $\mathcal{B}$ outputs it as a pre-image for the AIM2, otherwise $\mathcal{B}$ outputs $\perp$.

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

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

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

**Algorithm 14:** $H_1(q_1 = \sigma_1)$:

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

**for** $k \in [\tau], i \in [N]$ **do**

  $\mathsf{com}_k^{(i)} \to \mathsf{Bad}$.

// 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 $\sigma_1$.

**for** $k \in [\tau], i \in [N]$ **do**

  **if** $\exists\mathsf{seed}_k^{(i)} : ((\mathsf{salt}, k, i, \mathsf{seed}_k^{(i)}), \mathsf{com}_k^{(i)}, \mathsf{pt}_k^{(i)}, a_k^{(i)}, c_k^{(i)}, (t_{k,j}^{(i)})_{j\in[\ell]}) \in \mathcal{Q}_5$ **then**

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

      $\mathsf{pt}_k^{(i)} \leftarrow \mathsf{pt}_k^{(i)} + \Delta\mathsf{pt}_k, c_k^{(i)} \leftarrow c_k^{(i)} + \Delta c_k$ and $(t_{k,j}^{(i)} \leftarrow t_{k,j}^{(i)} + \Delta t_{k,j})_{j\in[\ell]}$

    $(\mathsf{pt}_k^{(i)}, c_k^{(i)}, (t_{k,j}^{(i)})_{j\in[\ell]}) \to \mathcal{T}_{\mathsf{sh}}[q_1, k, i]$

// 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**

  $\mathsf{pt}_k \leftarrow \sum_i \mathsf{pt}_k^{(i)}, c_k \leftarrow \sum_i c_k^{(i)}, a_k \leftarrow \sum_i a_k^{(i)}, (t_{k,j} \leftarrow \sum_i t_{k,j}^{(i)})_{j\in[\ell]}$.

  **for** $j \in [\ell]$ **do**

    Set $x_{k,j} \leftarrow t_{k,j}$ and $z_{k,j} \leftarrow (x_{k,j})^{2^{e_j}} + \gamma_j \cdot x_{k,j}$.

  **for** $j = \ell$ **do**

    Set $x_{k,j} \leftarrow \sum_{j\in[\ell]} A_{\mathsf{iv},j} \cdot x_{k,j} + b_{\mathsf{iv}}$ and $z_{k,j} \leftarrow (x_{k,j})^{2^{e_*}} + \mathsf{ct} \cdot x_{k,j}$.

  $\mathsf{pt}_k \to \mathcal{T}_{\mathsf{in}}[q_1, k]$.

$r \leftarrow_\$ \{0,1\}^{2\lambda}$.

**if** $r \in \mathsf{Bad}$ **then**

  abort.

$r \to \mathsf{Bad}$.

$(q_1, r) \to \mathcal{Q}_1$.

// Compute the multiplication check protocol values.

$(\epsilon_{k,j})_{j\in[\ell+1]} \leftarrow \mathsf{ExpandH1}(r)$.

**for** *each* $k : \mathcal{T}_{\mathsf{in}}[q_1, k] \neq \emptyset$ **do**

  $\alpha_k = a_k + \sum_{j\in[\ell+1]} \epsilon_j \cdot x_j + a_k$.

  $v_k = c_k + \sum_{j\in[\ell+1]} \epsilon_j \cdot z_{k,j} - \alpha_k \cdot \mathsf{pt}$.

Return $r$.

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

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

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

$r \leftarrow_\$ \{0,1\}^{2\lambda}$.<br>
**if** $r \in \mathsf{Bad}$ **then**<br>
  abort.<br>
$r \to \mathsf{Bad}$.<br>
$\left(\mathsf{pt}_k^{(i)}, a_k^{(i)}, c_k^{(i)}, (t_{k,j}^{(i)})_{j\in[\ell]}\right) \leftarrow_\$ \mathbb{F}_{2^n} \times \mathbb{F}_{2^n} \times \mathbb{F}_{2^n} \times (\mathbb{F}_{2^n})^\ell$<br>
$\left(q_\mathsf{c}, r, \mathsf{pt}_k^{(i)}, a_k^{(i)}, c_k^{(i)}, (t_{k,j}^{(i)})_{j\in[\ell]}\right) \to \mathcal{Q}_\mathsf{c}$.<br>
Return $\left(r, \mathsf{pt}_k^{(i)}, a_k^{(i)}, c_k^{(i)}, (t_{k,j}^{(i)})_{j\in[\ell]}\right)$.

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

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

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 $\tau$ 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.

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

where $X$ and $Y$ are defined as above. Setting AIM2 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 $(N, \tau, \lambda)$ parameters of* AIMer *are appropriately chosen. For a PPT adversary $\mathcal{A}$ against the EUF-CMA security of* AIMer *with total $Q_{\text{sig}}$ signing oracle queries and $Q$ random oracle queries, there exist a PPT adversary $\mathcal{B}$ against the EUF-KO security of* AIMer *(with same amount of queries to random oracles) and a PPT adversary $\mathcal{C}$ against the PRF security of $H_3$*[1] *such that*

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

*Proof.* Let $\mathcal{A}$ be an EUF-CMA adversary against AIMer for given $(\mathsf{iv}, \mathsf{ct})$. Let $\mathsf{G}_0$ be the original EUF-CMA game. Let $\mathcal{O}_{\text{sig}}$ be the signing oracle, and $Q_{\text{sig}}$ be the number of different signing queries during the game by $\mathcal{A}$, $Q_i$ for $i = 0, 1, 2, 4, 5$ be the

[1] $H_3$ 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 ($\rho \leftarrow 0^n$), and prove that of the probabilistic version later. Without loss of generality, we assume that all messages in signing queries are distinct.

$\mathsf{G}_1$: This game acts same as $\mathsf{G}_0$ except that it aborts if there exists two different queries on $H_0$ with same outputs. As output length of $H_0$ is $2\lambda$, we have

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

$\mathsf{G}_2$: $\mathcal{O}_{\text{sig}}$ 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 $H_3(\mathsf{pt}, \mu, \rho)$. As $\mu$ are 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 $\mathsf{pt}$. Therefore, there exists a PPT adversary $\mathcal{C}$ against the PRF security of $H_3$ such that

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

$\mathsf{G}_3$: $\mathcal{O}_{\text{sig}}$ samples $(\mathsf{nodes}_1[2^j+1], \mathsf{nodes}_1[2^j+2])$ in ExpandTree uniformly at random instead of computing $H_4(\mathsf{salt}, 0, 2^{j-1}, \mathsf{nodes}[2^{j-1}])$ 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 $\mathsf{salt}$ 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

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

$\mathsf{G}_4$: $\mathcal{O}_{\text{sig}}$ samples $\left(\mathsf{com}_0^{(0)}, \mathsf{pt}_0^{(0)}, (t_{0,j}^{(0)})_{j\in[\ell]}, c_0^{(0)}\right)$ 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

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

$\mathsf{G}_5$: $\mathcal{O}_{\text{sig}}$ samples $h_1 \in \{0,1\}^{2\lambda}$ at random instead of computing

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

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

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

$\mathsf{G}_6$: $\mathcal{O}_{\text{sig}}$ samples $h_2 \in \{0,1\}^{2\lambda}$ at random instead of computing

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

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

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

$\mathsf{G}_7$: $\mathcal{O}_{\text{sig}}$ 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 $\bar{i}_1 = 0$, it does not need to replace $\mathsf{seed}_0^{(0)}$ with a random element again. Similarly to $\mathsf{G}_3$, it is indistinguishable from the previous game with the advantage bounded by

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

$\mathsf{G}_8$: $\mathcal{O}_{\text{sig}}$ 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, $\mathcal{O}_{\text{sig}}$ sets $v_k^{(\bar{i}_k)} \leftarrow 0 - \sum_{i\neq\bar{i}_k} v_k^{(i)}$ for each $k \in [\tau]$. $\mathcal{O}_{\text{sig}}$ 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 $2\lambda$ bits, this game is indistinguishable with the previous game unless the simulation is aborted, and the probability of abort is

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

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 $\mathsf{pt}$.

$\mathsf{G}_9$: $\mathcal{O}_{\text{sig}}$ replaces

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

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

Note that now for all $k \in [\tau]$, $(\alpha_k^{(\bar{i}_k)})_{k\in[\tau]}$ is random and independent of $\mathsf{pt}$. 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 $\mathsf{G}_8$ does not depend on the secret key $\mathsf{pt}$, and it implies that $\mathsf{G}_8$ 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

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

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 $\lambda$ assuming that AIM2 is a secure one-way function and that parameters $(N, \tau, \lambda)$ 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 $\mathcal{O}_{\text{sig}}$ that involve the same messages and $\rho$ values. If $(m, \rho)$ 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 $\mathcal{A}$.

□

## 5.2 Information-Theoretic Security of AIM2 in the Random Permutation Model

In this section, we consider the one-wayness of AIM2. More precisely, we will prove the *everywhere preimage resistance* [RS04] of AIM2 when the underlying S-boxes are modeled as public random permutations and iv is (implicitly) fixed.[2] On the other hand, we do not claim that the algebraic S-boxes of AIM2 behave like random permutations. The point of the provable security of AIM2 is that one cannot break the one-wayness of AIM2 without exploiting any particular properties of the underlying S-boxes.

[2]The sum of two public random permutations is indifferentiable from a public random function up to $2^{\frac{2n}{3}}$ queries [GBJ+23], implying the preimage security of AIM2 up to the same query complexity, while we prove here its preimage security up to $2^n$ queries.

For simplicity, we will assume that $\ell = 2$. The security of AIM2 with $\ell > 2$ is similarly proved. In the public permutation model and in the single-user setting, AIM2 is defined as

$$\mathsf{AIM2}(\mathsf{pt}) = S_2(A_0 \cdot S_0(\mathsf{pt}) \oplus A_1 \cdot S_1(\mathsf{pt}) \oplus b) \oplus \mathsf{pt}$$

for $\mathsf{pt} \in \{0,1\}^n$, where $S_0$, $S_1$, $S_2$ are independent public random permutations,[3] and $A_0$ and $A_1$ are fixed $n \times n$ invertible matrices, and $b$ is a fixed $n \times 1$ vector over $\mathbb{F}_2$.

In the preimage resistance experiment, a computationally unbounded adversary $\mathcal{A}$ with oracle access to $S_i$, $i = 0, 1, 2$, selects and announces a point $\mathsf{ct} \in \{0,1\}^n$ *before making queries to the underlying permutations*. After making $q$ forward and backward queries in total,[4] $\mathcal{A}$ obtains a *query history*

$$\mathcal{Q} = \{(i_j, x_j, y_j)\}_{j=0}^{q-1}$$

such that $S_{i_j}(x_j) = y_j$ and $\mathcal{A}$'s $j$-th query is either $S_{i_j}(x_j) = y_j$ or $S_{i_j}^{-1}(y_j) = x_j$ for $j = 0, \ldots q-1$. We say that $\mathcal{A}$ *succeeds in finding a preimage of* ct if its query history $\mathcal{Q}$ contains three queries $S_0(x_0) = y_0$, $S_1(x_1) = y_1$ and $S_2(x_2) = y_2$ such that

$$\begin{aligned} x_0 &= x_1, \\ x_2 &= A_0 \cdot y_0 \oplus A_1 \cdot y_1 \oplus b, \\ \mathsf{ct} &= y_2 \oplus x_0. \end{aligned}$$

In this case, we say that $\mathcal{A}$ wins the preimage-finding game, breaking the one-wayness of AIM2. Assuming that $\mathcal{A}$ is information-theoretic, we can prove that $\mathcal{A}$'s winning probability, denoted $\mathbf{Adv}^{\mathsf{epre}}_{\mathsf{AIM2}}(\mathcal{A})$, is upper bounded as follows.

$$\mathbf{Adv}^{\mathsf{epre}}_{\mathsf{AIM2}}(\mathcal{A}) \leq \frac{2q}{2^n - q}. \tag{7}$$

PROOF OF (7). Since $\mathcal{A}$ is information-theoretic, we can assume that $\mathcal{A}$ is deterministic. Furthermore, we assume that $\mathcal{A}$ does not make any redundant query. More precisely, $\mathcal{A}$ never makes a query that will result in a triple $(i, x, y)$ which is already present in the query history.

Our security proof also uses the notion of "free" queries. Formally, these can be modeled as queries which the adversary is "forced" to query (under certain conditions), but for which the adversary is not charged: they do not count towards the maximum of $q$ queries which the adversary is allowed. However, these queries become part of the adversary's query history, just like other queries. In particular, the adversary is not allowed, later, to remake these queries "on its own" (due to the assumption that the adversary never makes a query which it already owns). Precisely, we will modify $\mathcal{A}$ so that whenever $\mathcal{A}$ makes a (forward or backward) query to $S_0$ (resp. $S_1$) obtaining $S_0(x) = y$ (resp. $S_1(x) = y$), $\mathcal{A}$ makes an additional *forward* query to $S_1$ (resp. $S_0$) with $x$ *for free*. This additional query will not degrade $\mathcal{A}$'s preimage-finding advantage since $\mathcal{A}$ is free to ignore it.

[3] We ignore constant addition to the S-box input or regard it as a part of the S-box.

[4] We assume that $\mathcal{A}$ evaluates AIM2 only by making oracle queries to the underlying permutations.

An evaluation $\mathsf{AIM2}(\mathsf{pt}) = \mathsf{ct}$ consists of three S-box queries. Among the three S-box queries, the lastly asked one is called the *preimage-finding query*. We distinguish two cases.

**Case 1.** The preimage-finding query is made to either $S_0$ or $S_1$. Since $\mathcal{A}$ consecutively obtains a pair of queries of the form $S_0(x) = y_0$ and $S_1(x) = y_1$, any preimage-finding query to either $S_0$ or $S_1$ should be forward. If it is $S_0(x)$ (without loss of generality), then there should be queries $S_1(x) = y$ for some $y$ and $S_2(z) = x \oplus \mathsf{ct}$ for some $z$ that have already been made by $\mathcal{A}$. In order for $S_0(x)$ to be the preimage-finding query, it should be the case that

$$S_2(A_0 \cdot S_0(x) \oplus A_1 \cdot S_1(x) \oplus b) = x \oplus \mathsf{ct}$$

or equivalently,

$$S_0(x) = A_0^{-1} \cdot (z \oplus b \oplus A_1 \cdot y)$$

which happens with probability at most $\frac{1}{2^n - q}$. Therefore, the probability of this case is upper bounded by $\frac{q}{2^n - q}$.

**Case 2.** The preimage-finding query is made to $S_2$. In order to address this case, we use the notion of a *wish list*, which was first introduced in [AFK+11]. Namely, whenever $\mathcal{A}$ makes a pair of queries $S_0(x) = y_0$ and $S_1(x) = y_1$, the evaluation

$$S_2 : A_0 \cdot y_0 \oplus A_1 \cdot y_1 \oplus b \mapsto x \oplus \mathsf{ct}$$

is included in the wish list $\mathcal{W}$. In order for an $S_2$-query to complete an evaluation $\mathsf{AIM2}(\mathsf{pt}) = \mathsf{ct}$ for any pt, at least one "wish" in $\mathcal{W}$ should be made come true. Each evaluation in $\mathcal{W}$ is obtained with probability at most $\frac{1}{2^n - q}$, and $|\mathcal{W}| \leq q$. Therefore, the probability of this case is upper bounded by $\frac{q}{2^n - q}$.

Overall, we can conclude that

$$\mathbf{Adv}^{\mathsf{epre}}_{\mathsf{AIM2}}(\mathcal{A}) \leq \frac{2q}{2^n - q}.$$

ONE-WAYNESS IN THE MULTI-USER SETTING. In the multi-user setting with $u$ users, $\mathcal{A}$ is given $u$ different target images, where the adversarial goal is to invert any of the target images. In this setting, the adversarial preimage finding advantage is upper bounded by

$$\frac{2uq}{2^n - q}. \tag{8}$$

The proof of (8) follows the same line of argument as the single-user security proof. The difference is that the probability that each query to either $S_0$ or $S_1$ becomes the preimage-finding one is upper bounded by $\frac{uq}{2^n - q}$ and the size of the wish list (in the second case) is upper bounded by $uq$.

We note that the above bound does not mean that AIM2 provides only the birthday-bound security in the multi-user setting. The straightforward birthday-bound attack is mitigated since AIM2 is based on a distinct linear layer for every user.

# 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 $\tau$ are fixed based on the soundness analysis given in [KZ22]; we see that an attacker should make at least $2^{\lambda}$ 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 AIM2. AIM2 is a one-way function, which does not follow the traditional design rationale of symmetric primitives. It takes random strings $\mathsf{iv}$ and $\mathsf{pt}$ as input, and outputs $\mathsf{ct} = \mathsf{AIM2}(\mathsf{iv}, \mathsf{pt})$. We expect that finding $\mathsf{pt}^*$ for a given pair $(\mathsf{iv}, \mathsf{ct})$ such that $\mathsf{ct} = \mathsf{AIM2}(\mathsf{iv}, \mathsf{pt}^*)$ is as hard as key recovery of AES with the same security level. To support our claim, we not only prove the information-theoretic security of AIM2 but also investigate its security against brute-force attacks, algebraic attacks, statistical attacks, and quantum attacks in Section 6.3.

In Section 5.2, we prove the everywhere preimage resistance [RS04] of AIM2 in the random permutation model. The one-wayness is proved assuming that the S-boxes are modeled as public random permutations. Although our choice of S-boxes is far from a random permutation, the proof itself exhibits that AIM2 is one-way unless any particular properties of the underlying S-boxes are exploited.

For the algebraic attacks, we analyze the security of AIM2 against the fast exhaustive search attacks [LMOM23, Bou22], Gröbner basis algorithm, Dinur's equation solving algorithm [Din21], and the linearization attack by Zhang et al. [ZWY+23]. We argue that AIM2 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 AIM2 require more gate-count complexity than required for AES, or require more than $2^\lambda$ memory bits. For the statistical attacks, we lower bounded the weights of differential and linear trails of AIM by near $\lambda$, although statistical attacks are not possible with a single input-output pair. 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 AIM2 is not lower than that applied to AES with the same security level. All the analysis on AIM2 is summarized in Section 6.3.

In the multi-user setting, we expect that finding one of $\mathsf{pt}_i$ given multiple pairs $\{(\mathsf{iv}_i, \mathsf{ct}_i)\}$ such that $\mathsf{ct}_i = \mathsf{AIM2}(\mathsf{iv}_i, \mathsf{pt}_i)$ for some $i$ is hard assuming that iv's are randomly chosen. If iv's can be arbitrarily chosen, a collision of $\mathsf{ct}_i$ is connected to a forgery. For example, if an IV value $\mathsf{iv}^*$ collides $q$ times in a set of public keys, an attacker may compute the function $\mathsf{AIM2}(\mathsf{iv}^*, \mathsf{pt})$ 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 AIM2 to the best of our knowledge. For algebraic attacks, any two sets of equations built for distinct pt's are not compatible. For statistical attacks, any two public-key pairs are not compatible with differential/linear cryptanalysis if corresponding pt's are distinct.

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 $2\lambda$-bit digests of the SHAKE output.

We expect the concrete security provided by SHAKE for collision and preimage resistance as claimed in [NIS15]. For $\lambda \in \{128, 256\}$, the preimage resistance of SHAKE-$\lambda$ 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 $2\lambda$-bit digests provide $\lambda$-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-$\lambda$ has collision resistance of $\min(2^{k/2}, 2^\lambda)$ against classical attacks, the $2\lambda$-bit digest also allows $\lambda$-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 $(\lambda, N, \tau)$. A more formal analysis is given in Section 5.1. Let $\tau_1$ and $\tau_2$ 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 $\tau_i$ 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 $\tau_1$ 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$2^\lambda$.

## 6.3 Known Attacks to AIM2

### 6.3.1 Brute-force Attack

Saarinen proposed an efficient brute-force attack for AIM using a linear feedback shift register.[5] Although the symmetric primitive in AIMer is changed to AIM2, his attack remains valid and is the fastest brute-force attack. By introducing an output of an inverse Mersenne S-box as a new variable, we can establish a simpler equation. For example, in AIM2-I or AIM2-III, one may find $y$ by iterating $y$ and $y^{-1}$ such satisfies

$$\mathsf{Mer}[e_0](t_0) = x + \gamma_0 \text{ where } \begin{cases} x := y^{2^{e_1}} \cdot y^{-1} + \gamma_1, \\ t_* := \mathsf{Mer}[e_*]^{-1}(x + \mathsf{ct}), \\ t_0 := A_{\mathsf{iv},0}^{-1}(b_{\mathsf{iv}} + A_{\mathsf{iv},1}(y) + t_*). \end{cases}$$

An attacker may change the new variable $y = \mathsf{Mer}[e_i]^{-1}(x + \gamma_i)$ for some $i \in \{0, \dots, \ell-1, *\}$ where ct is regarded as $\gamma_*$, and its corresponding system to reduce the amount of computation. Assuming that a multiplication by a fixed matrix does not require any AND gate and a squaring of a finite field element requires $n$ XOR gates, the minimum complexities are $2^{147.0}/2^{212.2}/2^{277.7}$ for AIM2-I/III/V.[6] These values are still larger than the gate-count complexity of AES ($2^{143}/2^{207}/2^{272}$). The numbers of gates required to evaluate addition chains for S-boxes are in Table 5.

For a comparison, we note that the complexities of the brute-force attack with direct computations are $2^{147.7}/2^{212.9}/2^{278.2}$ for AIM2-I/III/V. These values are slightly ($< 1$ bit) larger than the costs of the former method.

---

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

[6] This value is computed assuming that finite field multiplication costs $2n^2$ binary gates.

| Scheme | Circuit | #Operations | |
|---|---|---|---|
| | | FF Mult. | FF Square |
| AIM2-I | $\mathsf{Mer}[3]$ / $\mathsf{Mer}[3]^{-1}$ | 2 / 8 | 2 / 126 |
| | $\mathsf{Mer}[49]$ / $\mathsf{Mer}[49]^{-1}$ | 7 / 11 | 48 / 127 |
| | $\mathsf{Mer}[91]$ / $\mathsf{Mer}[91]^{-1}$ | 9 / 11 | 90 / 127 |
| AIM2-III | $\mathsf{Mer}[5]$ / $\mathsf{Mer}[5]^{-1}$ | 3 / 9 | 4 / 190 |
| | $\mathsf{Mer}[17]$ / $\mathsf{Mer}[17]^{-1}$ | 5 / 11 | 16 / 191 |
| | $\mathsf{Mer}[47]$ / $\mathsf{Mer}[47]^{-1}$ | 8 / 11 | 46 / 191 |
| AIM2-V | $\mathsf{Mer}[3]$ / $\mathsf{Mer}[3]^{-1}$ | 2 / 10 | 2 / 255 |
| | $\mathsf{Mer}[7]$ / $\mathsf{Mer}[7]^{-1}$ | 4 / 11 | 6 / 255 |
| | $\mathsf{Mer}[11]$ / $\mathsf{Mer}[11]^{-1}$ | 5 / 10 | 10 / 255 |
| | $\mathsf{Mer}[141]$ / $\mathsf{Mer}[141]^{-1}$ | 10 / 10 | 140 / 253 |

Table 5: The number of operations for each type of operation in AIM2.

#### 6.3.2 Algebraic Attacks

Since our attack model does not allow multiple evaluations for a single instance of AIM2, 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 AIM2. As several attacks on AIM were proposed, we describe how those attacks are mitigated in AIM2. We also consider algebraic attacks which have been recently studied for MPC/ZK-friendly ciphers such as LowMC [ARS+15] and large S-box-based ones.

Various Systems of AIM2. There are multiple ways of building a system of equations from an evaluation of AIM2. We can categorize them according to the number of (Boolean) variables and find the optimal choice of variables to obtain a system of the lowest degree. Since $\ell \in \{2, 3\}$ is recommended, we consider four types of systems of Boolean equations as follows.

1. Systems in $n$ variables.
2. Systems in $2n$ variables.
3. Systems in $3n$ variables.
4. Systems in $4n$ variables (only for $\ell = 3$).

With $(\ell+1)n$ variables, we can establish a system $S_{\mathsf{quad}}$ of *quadratic* equations. The variables are denoted as follows.

- $x$: the input of AIM2, i.e., $\mathsf{pt}$
- $t_i$: the output of $\mathsf{Mer}[e_i]^{-1}$ for $i = 0, \ldots, \ell - 1$
- $z$: the output of $\mathsf{Lin}$

From $\mathsf{Mer}[e_i]^{-1}(x + \gamma_i) = t_i$, we obtain $3n$ Boolean quadratic equations in $x$ and $t_i$ induced by the following relations.

$$\begin{cases} t_i(x+\gamma_i) = t_i^{2^{e_i}}, \\ t_i(x+\gamma_i)^2 = t_i^{2^{e_i}}(x+\gamma_i), \\ t_i^2(x+\gamma_i) = t_i^{2^{e_i}+1}. \end{cases}$$

When $x$ and $t_i$ are of higher degrees with respect to other variables, the first two relations result in $2n$ equations of degree $\deg x + \deg t_i$, while the last one results in $n$ equations of degree $\max(\deg x + \deg t_i, 2\deg t_i)$. There are also $n$ Boolean quadratic equations in $t_i$ and $t_j$ induced by the following.

$$(\gamma_i + \gamma_j)t_i t_j = t_i^{2^{e_i}} t_j + t_i t_j^{2^{e_j}}.$$

We note that $z$ has the same relation with $t_i$ with respect to $x$ as $z = \mathsf{Mer}[e_*]^{-1}(x + \mathsf{ct})$. Using the brute-force search of quadratic equations on toy parameters, we find that these are all the possible (linearly independent) quadratic equations on AIM2 (see [KHSL24] for details). Hence, $S_{\mathsf{quad}}$ consists of $3(\ell+1)n + \binom{\ell+1}{2}n$ quadratic equations.

With fewer variables, the resulting systems would have higher degrees. For example, $\mathsf{Mer}[e_i]^{-1}$ implicitly determines $3n$ quadratic equations in $x$ and $t_i$ as above, while $t_i$ (resp. $x$) can be explicitly represented by a polynomial in $x$ (resp. $t_i$). We can also explicitly represent $t_i$ using $t_j$ for $j \neq i$ or $z$ as follows.

$$\begin{aligned} t_i &= \mathsf{Mer}[e_i]^{-1}\left(\mathsf{Mer}[e_j](t_j) \oplus \gamma_i \oplus \gamma_j\right) \\ &= \mathsf{Mer}[e_i]^{-1}\left(\mathsf{Mer}[e_*](z) \oplus \mathsf{ct}\right). \end{aligned}$$

The degree of $t_i$ with respect to $t_j$ (resp. $z$) might be greater than the degree of $\mathsf{Mer}[e_i]^{-1} \circ \mathsf{Mer}[e_j]$ (resp. $\mathsf{Mer}[e_i]^{-1} \circ \mathsf{Mer}[e_*]$) due to the constant addition, while we estimate it as the degree of the composition (without constant addition) for simplicity.

Table 6 summarizes a system of equations of the lowest degree for each type, where such systems are denoted by $S_1, S_2, \ldots, S_{\mathsf{quad}}$ respectively, according to the number of variables. The complexities are measured by (3) with $\omega = 2$. For systems of equations of type $S_1$ in $n$ variables, we did not compute precise complexities since a system of degree near $n/2$ requires the Gröbner basis algorithm to use approximately $2^n$ monomials so that the time complexity will be close to $O(2^{2n})$.

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.

| Scheme | Type | #Var | Variables | (#Eq, Deg) | Gröbner Basis | | | Dinur [Din21] | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | $k$ | $d_{reg}$ | Time | Time | Memory |
| AIM2-I | $S_1$ | $n$ | $t_0$ | $(n, 60)$ | - | - | - | 141.2 | 140.4 |
| | $S_2$ | $2n$ | $t_0$, $t_1$ | $(3n, 2)$ | 62 | 15 | 207.9 | 244.6 | 177.2 |
| | $S_{\text{quad}}$ | $3n$ | $x$, $t_0$, $t_1$ | $(12n, 2)$ | 0 | 16 | 185.3 | 330.1 | 258.9 |
| AIM2-III | $S_1$ | $n$ | $x$ | $(2n, 114)$ | - | - | - | 206.5 | 205.9 |
| | $S_2$ | $2n$ | $t_0$, $t_1$ | $(3n, 2)$ | 100 | 20 | 301.9 | 330.1 | 258.9 |
| | $S_{\text{quad}}$ | $3n$ | $x$, $t_0$, $t_1$ | $(12n, 2)$ | 0 | 22 | 262.4 | 487.7 | 381.0 |
| AIM2-V | $S_1$ | $n$ | $x$ | $(2n, 172)$ | - | - | - | 271.4 | 270.9 |
| | $S_2$ | $2n$ | $t_1$, $z$ | $(n, 2) + (2n, 38)$ | 253 | 30 | 513.5 | 525.0 | 520.0 |
| | $S_3$ | $3n$ | $t_0$, $t_1$, $t_2$ | $(6n, 2)$ | 2 | 47 | 503.7 | 644.9 | 502.7 |
| | $S_{\text{quad}}$ | $4n$ | $x$, $t_0$, $t_1$, $t_2$ | $(18n, 2)$ | 9 | 32 | 411.4 | 854.4 | 664.7 |

Table 6: Optimal systems of equations and their security against algebraic attacks. $(\#\text{Eq}, \text{Deg}) = (a, b)$ means that the system contains $a$ equations of degree $b$. 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' and 'Memory' are in log.

Introducing New Variables Other Than S-box Outputs. We considered systems whose variables are inputs/outputs of the S-boxes. One might try to build a system by introducing new variables other than S-box outputs. However, such systems have no advantage over the previous ones in terms of algebraic attacks. We refer to [KHSL24] for details.

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 AIM2 since the constant addition by AddConst 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 Starkad/Poseidon [GKI 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, AIM2 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 AIM2.

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 signature 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, LIM21b, 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, LIM21b]. For AIM2 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. One arguable issue of this algorithm is its high memory complexity of $O(n^2 \cdot 2^{(1-1/(1.35d))n})$, making it infeasible in practice. For AIM2, the memory complexity required by Dinur's algorithm exceeds the security level, i.e., more than $2^\lambda$ bits of memory are required for each level of security $\lambda$. Table 6 shows the time and the memory complexity of Dinur's algorithm for each system of AIM2. Subsequent works [LMSI22, BBCV22] are proposed to reduce the memory complexity of the algorithm at the cost of slightly increased time complexity, while these variants do not apply to AIM2 since they all follow the guess-and-linearization strategy on LowMC.

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. According to the authors' claim, these attacks costs $2^{211.51}$ binary operations for AIM2-III, and $2^{319.97}$ binary operations for AIM2-V even if linear algebra constant $\omega$ is 2. These values are larger than the brute-force complexity of AES with the same security level. Assuming that the finite field multiplication in $GF(2^n)$ costs $n \log n$ binary operations via FFT (see Table 4 in [SCL+25]), this algebraic attack is slower than the brute-force search of AIM2.

### 6.3.3 Differential and Linear Cryptanalysis

An adversary is allowed to evaluate AIM2 with an arbitrary input pair $(\mathsf{pt}, \mathsf{iv})$ in an offline manner. However, such an evaluation is independent of the actual secret key $\mathsf{pt}^*$, so the adversary is not able to collect a sufficient amount of statistical data which are related to $\mathsf{pt}^*$. Furthermore, the linear layer of AIM2 is generated independently at random for every user. For this reason, we believe that our construction is secure against any type of statistical attack including (impossible) differential, boomerang, and integral attacks.

In the multi-target scenario, an adversary has no information on which users have the same secret. Even for multiple users with the same iv, statistical attacks would not be feasible since all the inputs and their differences are unknown to the adversary. That said, to prevent any unexpected variant of differential and linear cryptanalysis, we summarize a lower bound of the weight of differential and correlation trails in this section.

DIFFERENTIAL CRYPTANALYSIS. Since AIM2 is a key-less primitive, we will estimate the security of AIM2 against differential cryptanalysis by lower bounding the weight of a differential trail (for example, as in [DVA12]).

Given a function $f : \{0,1\}^n \to \{0,1\}^m$, the *weight* of a differential $(\Delta x, \Delta y) \in \{0,1\}^n \times \{0,1\}^m$ is defined by

$$w_d(\Delta x \xrightarrow{f} \Delta y) := n - \log |\{x \in \{0,1\}^n : f(x \oplus \Delta x) \oplus f(x) = \Delta y\}| .$$

The weight is not defined if there is no $x$ such that $f(x \oplus \Delta x) \oplus f(x) = \Delta y$. Otherwise, we say that $\Delta x$ and $\Delta y$ are *compatible*.

A differential trail is the composition of compatible differentials. For AIM2, a differential trail from an input to the output (ignoring the feed-forward) can be represented as follows.

$$Q = \Delta_0 \xrightarrow{\mathsf{Mer}[e_0,\ldots,e_{\ell-1}]^{-1}} \Delta_1 \xrightarrow{\mathsf{Lin}} \Delta_2 \xrightarrow{\mathsf{Mer}[e_*]} \Delta_3$$

as AddConst does not affect differentials. Then, the weight of the differential trail $Q$ is defined as

$$w_d(Q) := \sum_{i=0}^{2} w_d(\Delta_i \to \Delta_{i+1}).$$

The weight of a Mersenne S-box is determined by the number of solutions to $\mathsf{Mer}[e](x \oplus \Delta x) \oplus \mathsf{Mer}[e](x) = \Delta y$, which is a polynomial equation of degree $2^e - 2$. Therefore, there are at most $2^e - 2$ solutions to this equation, which implies for $\Delta x \neq 0$,

$$w_d(\Delta x \xrightarrow{\mathsf{Mer}[e]} \Delta y) \geq n - \log_2(2^e - 2) \geq n - e.$$

Then we have

$$\begin{aligned} w_d(Q) &= \sum_i w_d(\Delta_i \to \Delta_{i+1}) \\ &\geq \sum_{0 \leq j \leq \ell-1} (n - e_j) = \ell n - \sum_j e_j \end{aligned}$$

as $\Delta_2$ may be zero. So, for any differential trail $Q$, $w_d(Q)$ is close to $\lambda$ with $\lambda = n$. We note that a trail $Q$ such that $w_d(Q) < \lambda$ never incurs a collision since $\Delta_3 = \Delta_0$, and the existence of such trail does not imply the feasibility of differential cryptanalysis since an adversary is not given a large enough number of plaintext-ciphertext pairs to mount the analysis.

DIFFERENCE ENUMERATION ATTACK. Recently, difference enumeration attacks to LowMC have been proposed [RST18, LIM21a, LSW+22], which require only a couple of chosen plaintext-ciphertext pairs. In such attacks, an adversary enumerates all possible input and output differences and tries to find a collision and recover the unknown key. This type of attack works for LowMC since it is based on small S-boxes. So one can easily find all possible differentials in LowMC. On the other hand, AIM2 is based on $n$-bit S-boxes, making it infeasible to enumerate all possible differences of each S-box.

LINEAR CRYPTANALYSIS. In contrast to differential cryptanalysis, security against linear cryptanalysis has been rarely evaluated for key-less primitives since its goal is to retrieve the secret key, not finding a collision or a second-preimage. That said, we lower bound the weight of a correlation trail for completeness in a similar way to differential cryptanalysis.

Given a function $f : \{0,1\}^n \to \{0,1\}^m$, the *weight* of a correlation $(\alpha, \beta) \in \{0,1\}^n \times \{0,1\}^m$ is defined by

$$w_l(\alpha \xrightarrow{f} \beta) := n - \log \left|2 \left|\{x \in \{0,1\}^n : \alpha^\top x = \beta^\top f(x)\}\right| - 2^n\right|.$$

The weight is not defined if there are exactly $2^{n-1}$ values for $x$ such that $\alpha^\top x = \beta^\top f(x)$. Otherwise, we say that $\alpha$ and $\beta$ are *compatible*.

A correlation trail is the composition of compatible correlations. For AIM2, a correlation trail from an input to the output (ignoring the feed-forward) can be represented as follows.

$$Q = \alpha_0 \xrightarrow{\mathsf{Mer}[e_0,\ldots,e_{\ell-1}]^{-1}} \alpha_1 \xrightarrow{\mathsf{Lin}} \alpha_2 \xrightarrow{\mathsf{Mer}[e_*]} \alpha_3.$$

Then the weight of the correlation trail $Q$ is defined as

$$w_l(Q) := \sum_{i=0}^{2} w_l(\alpha_i \to \alpha_{i+1}).$$

When $d$ is not a power-of-2 and $f(x) = x^d$ is invertible over $\mathbb{F}_{2^n}$, one has the following generic bound [KSW19].

$$\left|2 \left|\{x : \alpha^\top x = \beta^\top f(x)\}\right| - 2^n\right| \le (d-1)2^{n/2}$$

for any compatible correlation $(\alpha, \beta)$. Therefore the weight of a correlation trail of a Mersenne S-box is lower bounded by $w_l(Q) \ge \frac{n}{2} - e$. Then we have

$$\begin{aligned} w_l(Q) &= \sum_i w_l(\alpha_i \to \alpha_{i+1}) \\ &\ge \max_{i\in[\ell]}(n/2 - e_i) + w_l(\alpha_2 \to \alpha_3) \\ &\ge \max_{i\in[\ell]}(n/2 - e_i) + (n/2 - e_*) \\ &= n - e_0 - e_*. \end{aligned}$$

As $\mathsf{Lin}$ is a (full-rank) compression function, $\alpha_2$ cannot be the zero mask. Since linear cryptanalysis requires $2^{2w_l(Q)}$ plaintext-ciphertext pairs, AIM2 would be secure against linear cryptanalysis if

$$2(n - e_0 - e_*) \ge \lambda$$

which is the case for AIM2. We emphasize again that linear cryptanalysis is not practically relevant in our setting since AIM2 does not use any secret key, while all the inputs are kept secret and every user is assigned a distinct linear layer.

### 6.3.4 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]. In this section, we investigate if any specialized quantum algorithm targeting AIM2 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].

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

The quantum circuit size and the depth of AIM2 can be computed in a modular manner. AIM2 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].

In the context of quantum attacks, minimizing the circuit depth is crucial compared to classical attacks. Therefore, when employing Grover's algorithm for AIM2, it might be more efficient to compute the inputs and outputs of the linear layer in AIM2 for each candidate $\mathsf{pt}$ and check whether the intermediate variables satisfy proper linear equations, rather than searching for an $x$ that satisfies $\mathsf{AIM2}(\mathsf{iv}, x) = \mathsf{ct}$. For example, given $\mathsf{AIM2}(\mathsf{iv}, \cdot) = \mathsf{ct}$ with $\ell = 2$, one can find $x$ satisfying

$$\mathsf{Lin}[\mathsf{iv}](t_0, t_1) = t_* \text{ where } \begin{cases} t_0 := \mathsf{Mer}[e_0]^{-1}(x + \gamma_0), \\ t_1 := \mathsf{Mer}[e_1]^{-1}(x + \gamma_1), \\ t_* := \mathsf{Mer}[e_*]^{-1}(x + \mathsf{ct}). \end{cases} \quad (9)$$

Table 7 summarizes the total number of operations and the depth of operations for each type of operation to implement (9), where the number of operations required to evaluate addition chain for S-boxes are from Table 5. The depth of each operation for evaluating a single S-box is the same as the number of the same operations. Based on these numbers and the above references, we summarize the estimated cost of Grover's algorithm on AIM2 (in log) for each level of security in Table 7. We see that AIM2-I, AIM2-III, and AIM2-V satisfy the security level I, III and V, respectively.[7]

Comparatively, Song et al. proposed a quantum implementation of AIM2 and estimates cost of the Grover's algorithm on AIM2 [SJY+25]. The estimated costs

[7] 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.

are $2^{164}/2^{231}/2^{296}$. As they estimate the cost for honest computation of AIM2 rather than fast exhaustive search, this result shows a slightly larger value than the costs in Table 7.

The total cost of Grover's algorithm might be further reduced than expected; a better representation of the AIM2 circuit with respect to the total cost might be proposed, or 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 | | |
| AIM2-I | 30, 11 | 380, 127 | 162.6 | I (≥157) |
| AIM2-III | 31, 11 | 572, 191 | 229.2 | III (≥221) |
| AIM2-V | 41, 11 | 1018, 255 | 294.9 | V (≥285) |

Table 7: The number of operations and the depth for each type of operation used in AIM2, and the total cost of Grover's algorithm on AIM2 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 AIM2 can be represented by Boolean quadratic equations using intermediate variables. Precisely, we have a system of $3(\ell+1)n + \binom{\ell+1}{2}n$ quadratic equations in $(\ell+1)n$ variables. For this system of equations, the time complexity of GroverXL is given as $2^{(1.1062+o(1))n}$ for AIM2-I, III and $2^{(1.3568+o(1))n}$ for AIM2-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 AIM2 can be represented by $3(\ell+1)n + \binom{\ell+1}{2}n$ 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 AIM2-I, III is minimized to $O(2^{1.047n})$ when 29%[8] of variables are guessed, and that for AIM2-V is minimized to $O(2^{1.320n})$ when 20% 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

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

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 AIM2, the hamming weight of the secret key should 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 AIM2). 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 AIM2), we believe that the above attacks do not apply to AIM2 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 a multiple number of 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 seperation 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. AIM2 generates its linear layer from a random iv, so not only each user has a different secret key (i.e., the input of AIM2), but also the functions themselves are all different. Moreover, similarly to the mitigation techniques described above, all inputs to hash functions hash at least $2\lambda$-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 $\sigma_A$ under a public key $pk_A$. Then the adversary tries to produce a fake public key $pk_E$ such that $\sigma_A$ 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 key generation and signing algorithms, which manipulate the secret key of AIMer, are executed 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 8 presents the sizes of the public key, secret key, and signature for various parameter sets using the NIST/SUPERCOP API[9] 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 | 5,888 |
| AIMer-128s | 32 | 48 | 4,160 |
| AIMer-192f | 48 | 72 | 13,056 |
| AIMer-192s | 48 | 72 | 9,120 |
| AIMer-256f | 64 | 96 | 25,120 |
| AIMer-256s | 64 | 96 | 17,056 |

Table 8: Key and signature sizes for various parameter sets.

[9] 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 AIM2 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. We experimentally checked the compatibility of the AVX2 optimized implementation of AIMer at all security levels with the Open Quantum Safe (OQS) project.[10] After creating X.509 certificates signed with AIMer, we were able to establish TLS 1.3 connections without message fragmentation, where the key exchange algorithm was the hybrid protocol with ECDH (p256/p384/p521) [BCR+18] and CRYSTALS-Kyber [SAB+22] (512/768/1024) algorithms in OQS.

[10] http://github.com/open-quantum-safe/liboqs

# 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.

[AFK+11] Frederik Armknecht, Ewan Fleischmann, Matthias Krause, Jooyoung Lee, Martijn Stam, and John Steinberger. The preimage security of double-block-length compression functions. In *ASIACRYPT 2011*, pages 233–251. Springer, 2011.

[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.

[BCR+18] Elaine Barker, Lily Chen, Allen Roginsky, Apostol Vassilev, and Richard Davis. Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography, 2018. NIST SP 800-56A Rev.3.

[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.

[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.

[DVA12] Joan Daemen and Gilles Van Assche. Differential Propagation Analysis of Keccak. In Anne Canteaut, editor, *FSE 2012*, pages 422–441. Springer, 2012.

[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.

[GBJ+23] Aldo Gunsing, Ritam Bhaumik, Ashwin Jha, Bart Mennink, and Yaobin Shen. Revisiting the Indifferentiability of the Sum of Permutations. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023*, pages 628–660. Springer, 2023.

[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.

[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.

[KSW19] Daniel J Katz, KU Schmidt, and A Winterhof. Weil sums of binomials: Properties applications and open problems. In *Combinatorics and Finite Fields: Difference Sets, Polynomials, Pseudorandomness and Applications*, volume 23, pages 109–134. De Gruyter, 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.

[LIM21a] Fukang Liu, Takanori Isobe, and Willi Meier. Cryptanalysis of full LowMC and LowMC-M with algebraic techniques. In *CRYPTO 2021*, pages 368–401. Springer, 2021.

[LIM21b] 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 Pseudorandom Permutations from Pseudo-random Functions. In *CRYPTO '85*, pages 447–447. Springer, 1986.

[LSW+22] Fukang Liu, Santanu Sarkar, Gaoli Wang, Willi Meier, and Takanori Isobe. Algebraic Meet-in-the-Middle Attack on LowMC. Cryptology ePrint Archive, Paper 2022/019, 2022. https://eprint.iacr.org/2022/019, to appear Asiacrypt 2022.

[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.

[RS04] Phillip Rogaway and Thomas Shrimpton. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In *FSE 2004*, pages 371–388. Springer, 2004.

[RST18] Christian Rechberger, Hadi Soleimany, and Tyge Tiessen. Cryptanalysis of Low-Data Instances of Full LowMCv2. *IACR Transactions on Symmetric Cryptology*, 2018(3):163–181, 2018.

[SAB$^+$22] Peter Schwabe, Roberto Avanzi, Joppe Bos, Leo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehle, and Jintai Ding. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2022, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[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.

[SJY$^+$25] Gyeongju Song, Kyungbae Jang, Seyoung Yoon, Minwoo Lee, and Hwajeong Seo. Quantum Circuit Implementation and Resource Analysis of AIM2. In *Proceedings of the 28th Annual International Conference on Information Security and Cryptology*, pages 266–279, 2025.

[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.