Single-Trace Side-Channel Attack on the KpqC Digital Signature Scheme AIMer v3

Jinnuo Li[1], Yiteng Sun[1], Chi Cheng[3], Weijia Wang[1,2] and Hao Cheng[1,2(✉)]

[1] School of Cyber Science and Technology, Shandong University, Qingdao, China
jinnuo.li@mail.sdu.edu.cn
[2] State Key Laboratory of Cryptography and Digital Economy Security, Shandong University, Qingdao, China
{wjwang,hao.cheng}@sdu.edu.cn
[3] Guangzhou University, Guangzhou, China

Abstract. The Korean Post-Quantum Cryptography (KpqC) competition selected AIMer, a post-quantum digital signature scheme based on the MPC-in-the-Head paradigm, as one of its final algorithms in 2025. However, detailed side-channel analysis (SCA) of AIMer remains limited, particularly for its latest version, AIMer v3. In this paper, we present, to the best of our knowledge, the first side-channel attack on AIMer v3. Specifically, we demonstrate a single-trace power analysis attack using deep learning to exploit power leakage from bits of the secret preimage during the computation of AIM3's input affine layer. Using a ChipWhisperer-HuskyPlus, we recover the full secret key from a single signature on an ARM Cortex-M4 processor with a success rate of 100%. We further evaluate the attack performance under different noise levels and propose potential countermeasures.

Keywords: Side-channel attacks, Post-quantum cryptography, AIMer, MPC-in-the-Head

1 Introduction

AIMer [KHS+23] is a post-quantum signature scheme selected in the Korean Post-Quantum Cryptography (KpqC) competition [Kpq25]. It follows the MPC-in-the-Head (MPCitH) paradigm, combining a customized BN++ proof system with a one-way function from the AIM family to construct compact signatures based on symmetric cryptographic primitives. Following algebraic attacks on earlier versions [BFU26], AIMer v3 adopts AIM3, a tweakable one-way function that introduces an input affine layer and enhanced Mersenne S-boxes [CHK+26].

Beyond resistance to cryptanalytic attacks, the practical security of cryptographic schemes also depends on their resistance to implementation attacks, including side-channel attacks [KJJ99]. In this context, the side-channel security of signature schemes based on in-the-head proof paradigms has attracted increasing attention, particularly among candidates in the NIST Additional Digital Signatures standardization process. Godard et al. [GAG+25] demonstrated a single-trace side-channel attack on a variant of SDitH based on TCitH. For schemes based on VOLEitH, Jendral and Dubrova [JD26] presented two single-trace attacks on a masked implementation of FAEST. Godard et al. [GAG+26] subsequently developed a non-supervised single-trace SASCA exploiting VOLE correlations to recover secret data in FAEST and SDitH. More recently, Li et al. [LHZ+26] systematically analyzed DPA-style attacks on FAEST and SDitH, identifying six classes of attacks.

Despite these advances, the power side-channel security of AIMer v3 remains insufficiently understood. Its specification [CHK+26] emphasizes constant-time execution and

discusses precautions for handling the secret preimage during field arithmetic. Although it acknowledges the risks of power and electromagnetic attacks, it does not provide an experimental evaluation of power leakage. To address this gap, we perform a side-channel attack on AIMer v3, exploiting power leakage from the processing of preimage bits in AIM3's input affine layer. Our main contributions are as follows:

1.1 Contributions

- We present, to the best of our knowledge, the first side-channel attack on the reference implementation of AIMer v3. We evaluate the attack on an ARM Cortex-M4 processor, demonstrating full key recovery from a single power trace.
- We identify and experimentally analyze side-channel leakage arising from mask generation and accumulation operations in AIM3's input affine layer. We train eight convolutional neural network (CNN) classifiers with the same architecture to recover preimage bits, achieving an average classification accuracy of 99.98%.
- We evaluate our attack on the AIMer-128f parameter set using power traces collected from an STM32F415RGT6 microcontroller with a ChipWhisperer-HuskyPlus. Our attack achieves a full key recovery success rate of 97.73% from a single power trace, which increases to 100% with a maximum enumeration complexity of 2^5.
- We further evaluate the attack performance under different noise conditions and discuss potential countermeasures to mitigate the identified leakage.

1.2 Organization

Section 2 introduces the MPCitH paradigm and AIMer. Section 3 analyzes leakage in AIM3's input affine layer. Sections 4 and 5 describe the experimental setup and attack procedure, respectively. Section 6 presents the experimental results. Section 7 discusses potential countermeasures, and Section 8 concludes the paper.

2 Background

2.1 MPC-in-the-Head Paradigm

The MPC-in-the-Head (MPCitH) paradigm [IKOS07] constructs zero-knowledge proofs of knowledge from secure multiparty computation. For a public relation $\mathcal{R}$ and statement x, the prover demonstrates knowledge of a witness w satisfying $\mathcal{R}(x, w) = 1$ without revealing w. It distributes shares of w among N virtual parties and simulates their joint computation locally. Each party's view records its input share, random tape, and exchanged messages; the prover commits to these views before the verifier selects which ones to inspect.

The verifier requests a randomly chosen subset of views and checks that their openings match the commitments, that the disclosed messages and local computations are mutually consistent, and that the computation accepts the public statement. The number of opened views stays within the privacy threshold of the underlying MPC protocol, so their disclosure does not reveal the witness. Committing before the opening challenge prevents the prover from adapting the views to the verifier's selection, while independent repetitions reduce the probability of accepting an invalid proof.

To obtain a signature scheme, the secret signing value serves as the witness for a statement determined by the public key. The Fiat–Shamir transform [FS87] replaces the verifier's challenges with hashes of the transcript and binds them to the signed message. The signature contains the resulting proof, which the verifier checks by reconstructing the

Algorithm 1 AIMer key generation, signing, and verification

AIMer.KeyGen()

Output: Public key pk, secret key sk

repeat
  pt $\xleftarrow{\$} \mathbb{F}_{2^n}$, iv $\xleftarrow{\$} \{0,1\}^n$
  ct $\leftarrow$ AIM3.NoZeroInpSB(pt, iv)
until ct $\neq \perp$
return pk := (iv, ct), sk := (pt, iv, ct)

AIMer.Verify()

Input: Public key pk, message M, signature σ, context ctx

Parse σ; derive μ, $(\epsilon_{k,j})_{k,j}$ and $(\overline{\imath}_k)_k$
for $k = 0, \ldots, \tau - 1$ **do**
  Reconstruct $N-1$ party views and apply the correction values
  Recompute their AIM3 MPC and BN++ checking values
  $v_k^{(\overline{\imath}_k)} \leftarrow -\sum_{i \neq \overline{\imath}_k} v_k^{(i)}$
end for
Reconstruct σ_1, σ_2; $h_1' \leftarrow H_1(\mu, \sigma_1)$, $h_2' \leftarrow H_2(h_1', \sigma_2)$
return $(h_1' = h_1) \wedge (h_2' = h_2)$

AIMer.Sign()

Input: Secret key sk, message M, context ctx

Output: Signature σ

$\rho \xleftarrow{\$} \{0,1\}^\lambda$, $M' \leftarrow (|\text{ctx}|/8) \,\|\, \text{ctx} \,\|\, M$
$\mu \leftarrow H_0(\text{iv}, \text{ct}, M')$; compute AIM3 S-box outputs $(y_j)_{j<\ell}$
$(\text{salt}, (\text{seed}_k)_k) \leftarrow H_3(\mu, \text{pt}, \rho)$
for $k = 0, \ldots, \tau - 1$ **do**
  Expand GGM tree; derive party shares and commitments using H_5
  Compute witness offsets and simulate AIM3.MPC
end for
Form σ_1; $h_1 \leftarrow H_1(\mu, \sigma_1)$; $(\epsilon_{k,j})_{k,j} \leftarrow$ ExpandH1(h_1)
Compute $\alpha_{k,j}^{(i)}$, $\alpha_{k,j}$, $v_k^{(i)}$; form σ_2; $h_2 \leftarrow H_2(h_1, \sigma_2)$
$(\overline{\imath}_k)_k \leftarrow$ ExpandH2(h_2)
for $k = 0, \ldots, \tau - 1$ **do**
  $\text{path}_k \leftarrow$ RevealAllBut($\text{nodes}_k, \overline{\imath}_k$)
  $\text{open}_k \leftarrow (\text{path}_k, \text{com}_k^{(\overline{\imath}_k)}, \Delta\text{pt}_k, (\Delta y_{k,j})_{j<\ell}, \Delta c_k, (\alpha_{k,j}^{(\overline{\imath}_k)})_{j \leq \ell})$
end for
return $\sigma = (\text{salt}, h_1, h_2, (\text{open}_k)_k)$

Table 1: Recommended parameter sets of AIMer v3.0.

Parameter set	λ	ℓ	$\boldsymbol{e}$	N	τ	$\lvert\sigma\rvert$ (bytes)
AIMer-128f	128	2	$(3, 7, 11)$	16	33	6,944
AIMer-128s	128	2	$(3, 7, 11)$	256	17	4,704
AIMer-192f	192	2	$(11, 23, 47)$	16	49	15,408
AIMer-192s	192	2	$(11, 23, 47)$	256	25	10,320
AIMer-256f	256	3	$(3, 7, 11, 19)$	16	65	31,360
AIMer-256s	256	3	$(3, 7, 11, 19)$	256	33	20,224

disclosed views and recomputing the challenges. AIMer instantiates this construction with a proof of knowledge of an AIM3 preimage.

2.2 AIMer Signature Scheme

AIMer combines the tweakable one-way function AIM3 with a customized BN++ proof system [KZ22], which supports efficient proofs of arithmetic over large finite fields. The public key specifies a tweak iv and an output ct, while the signer knows a secret preimage $\text{pt} \in \mathbb{F}_{2^n}$, where $n = \lambda$. A signature proves knowledge of this preimage and binds the proof to the message through the Fiat–Shamir transform.

AIM3 evaluates the preimage through two affine layers and two layers of S-boxes, followed by a feed-forward operation. The public tweak determines the binary matrices A_j and constants b_j used in the affine layers. The input affine layer first maps the preimage to ℓ S-box inputs:

$$x_j = A_j(\text{pt}) + b_j, \qquad 0 \leq j < \ell. \tag{1}$$

Each input passes through an enhanced Mersenne S-box, $S[e](x) = x^{2^e-1} + x^{2^n-2}$. The

```
void aim3_sbox_outputs(
    gf sbox_outputs[AIM3_NUM_INPUT_SBOX+1],
    const aim_lin_t* lin, const gf pt, const gf ct) {
  ... Processing ...
  for (size_t i = 0; i < AIM3_NUM_INPUT_SBOX; i++) {
    // linear component: affine layer
    gf_mat_vec_mul(sbox_outputs[i], pt, lin->mat_A[i]);
    gf_add(sbox_outputs[i], sbox_outputs[i],
           lin->vec_b[i]);
    ... Processing ...
  }
  ... Processing ...
}
```

Figure 1: C code excerpt from AIMer's `aim3_sbox_outputs` function.

output affine layer combines the resulting values into the input of a final S-box, whose output is XORed with pt to produce ct. The input affine layer therefore operates directly on the secret preimage; its implementation is the focus of the leakage analysis in the next section.

The S-box structure allows BN++ to check each nonlinear operation using a single multiplication constraint. Specifically, a nonzero S-box input x_j and its output y_j satisfy $x_j y_j = x_j^{2^{e_j}} + 1$. The signer shares the preimage and the first ℓ S-box outputs among N virtual parties; the final S-box output follows from the preimage and the public output. It then simulates the computation and checks these constraints over τ parallel repetitions. Table 1 lists the recommended AIMer parameter sets. Algorithm 1 summarizes the three AIMer procedures described below, following the v3 specification.

Key generation The algorithm samples a preimage and a tweak uniformly at random and evaluates AIM3. It resamples both values whenever an S-box input is zero, as the proof requires nonzero inputs. For a valid pair, it publishes $\mathrm{pk} = (\mathrm{iv}, \mathrm{ct})$ and retains $\mathrm{sk} = (\mathrm{pt}, \mathrm{iv}, \mathrm{ct})$. Since iv and ct are public, recovering pt determines the full secret key.

Signing The signer computes the S-box outputs from the secret preimage and uses a GGM tree to generate the virtual parties' randomness in each repetition. Public correction values ensure that the resulting shares reconstruct the preimage, S-box outputs, and auxiliary correlation required by BN++.

Two successive challenges control the proof. The first challenge hash h_1 determines random coefficients for the multiplication check. Using these coefficients and the auxiliary correlation, the parties compute masked broadcast values α and checking shares v, whose sum is zero in an honest execution. The second challenge hash h_2 incorporates this checking transcript and selects one party in each repetition whose view remains unopened. The signature contains the salt, both challenge hashes, and the seed tree openings, commitments, correction values, and broadcast information needed to check the other $N - 1$ views.

Verification Verification reconstructs the disclosed views and checks their consistency through the two challenge hashes.

3 Leakage Analysis

Our attack targets the `gf_mat_vec_mul` call in `aim3_sbox_outputs` during signing, as shown in Figure 1. This call computes $A_j(\mathrm{pt})$ in AIM3's input affine layer. Its input `a` is the secret preimage pt, and `b` contains the columns of the public binary matrix A_j. The function `gf_add` then adds the public constant b_j.

```
void gf_mat_vec_mul(gf c, const gf a,
                    const gf b[AIM3_NUM_BITS_FIELD]) {
  const uint64_t* a_ptr = a;
  const gf* b_ptr = b;
  uint64_t temp_c0 = 0;
  uint64_t temp_c1 = 0;
  uint64_t mask;

  for (size_t i = AIM3_NUM_WORDS_FIELD; i; --i, ++a_ptr) {
    uint64_t index = *a_ptr;
    for (size_t j = AIM3_NUM_BITS_WORD; j;
         j -= 8, index >>= 8, b_ptr += 8) {
      mask = 0U - (index & 1);
      temp_c0 ^= (b_ptr[0][0] & mask);
      temp_c1 ^= (b_ptr[0][1] & mask);
      mask = 0U - ((index >> 1) & 1);
      temp_c0 ^= (b_ptr[1][0] & mask);
      temp_c1 ^= (b_ptr[1][1] & mask);
      mask = 0U - ((index >> 2) & 1);
      temp_c0 ^= (b_ptr[2][0] & mask);
      temp_c1 ^= (b_ptr[2][1] & mask);
      ... Processing ...
      mask = 0U - ((index >> 7) & 1);
      temp_c0 ^= (b_ptr[7][0] & mask);
      temp_c1 ^= (b_ptr[7][1] & mask);
    }
  }
  c[0] = temp_c0;
  c[1] = temp_c1;
}
```

Figure 2: C code excerpt from AIMer's `gf_mat_vec_mul` function.

```
gf_mat_vec_mul:
  ...
loop:
  ...
  sbfx  r6, r2, #1, #1   @ bit 1 mask
  ands  r7, r6
  ands  r6, r4
  eor.w lr, lr, r6
  ldr.w r6, [r3, #-108]
  sbfx  r4, r2, #1, #1   @ bit 1 mask
  ands  r6, r4
  eor.w r9, r9, r6
  ands  r4, r0
  ldrd  r6, r0, [r3, #-96]
  sbfx  fp, r2, #2, #1   @ bit 2 mask
  and.w r6, fp, r6
  eor.w r8, r8, r7
  eors  r1, r4
  ...
  adds  r3, #128         @ next 8 columns
  lsrs  r2, r2, #8
  cmp   r3, r1
  orr.w r2, r2, r5, lsl #24
  ...
  mov.w r5, r5, lsr #8
  bne.w loop
  ...
```

Figure 3: Disassembly code excerpt from AIMer's `gf_mat_vec_mul` function.

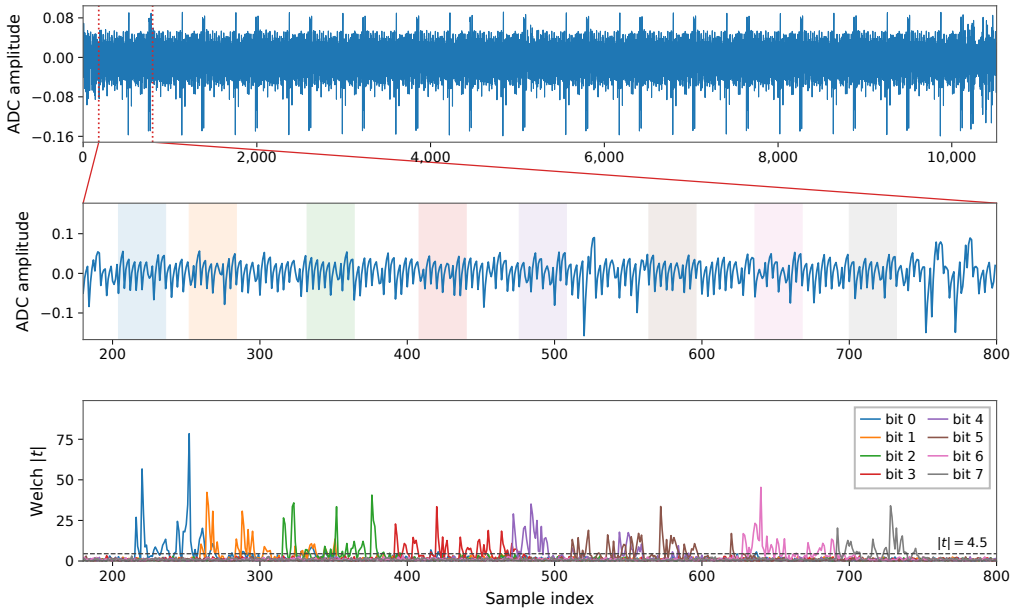


Figure 4: Trace segmentation and Welch's t-test results for `gf_mat_vec_mul`.

Figure 2 shows the C code snippet of `gf_mat_vec_mul`. We can see that a 64-bit word is loaded into `index` at each iteration of the outer loop. The unrolled inner loop processes eight bits from least to most significant, then shifts `index` and advances `b_ptr` by eight bits and columns, respectively. The two input words yield 16 iterations covering all 128 preimage bits. Each highlighted subtraction expands a bit $s \in \{0, 1\}$ into the mask

$$\begin{aligned} m &= 0 - s \pmod{2^{64}} \\ &= \begin{cases} \texttt{0x0000000000000000}, & s = 0, \\ \texttt{0xFFFFFFFFFFFFFFFF}, & s = 1. \end{cases} \end{aligned} \tag{2}$$

Then, the two AND operations yield either zero or the two words of the corresponding matrix column, which are XORed into `temp_c0` and `temp_c1`.

In the C implementation, `mask` is a 64-bit value with Hamming weight 0 or 64, depending on the preimage bit. On the 32-bit Cortex-M4, each 64-bit AND/XOR operation is compiled into two 32-bit operations. The disassembly in Figure 3 shows how the mask is constructed at this word size: `sbfx` replicates a preimage bit to form a 32-bit mask with Hamming weight 0 or 32. The subsequent AND/XOR operations use this mask to select and accumulate the matrix column. The sensitive intermediates identified in the C code therefore remain present in the firmware. In AIMer, changing the public tweak iv changes the matrix columns used in the AND/XOR operations. Each preimage bit is mapped to an all-zero or all-one mask determined solely by the corresponding bit of pt.

At the top of Figure 4, we show the mean power trace, which clearly exhibits a repeated pattern corresponding to the eight preimage bits processed in each loop iteration. We also present an enlarged view of the processing of the first byte, with shaded windows marking the segments used to classify its eight bits. For each preimage bit, we apply a Welch t-test to 1,000 traces grouped by bit value (0 or 1) and show the results at the bottom. For all eight bits, the selected windows contain peaks exceeding the threshold $|t| = 4.5$. These peaks follow the bit processing order, indicating leakage from the corresponding operations. We further analyze the power distributions for bit values 0 and 1 at a representative sample

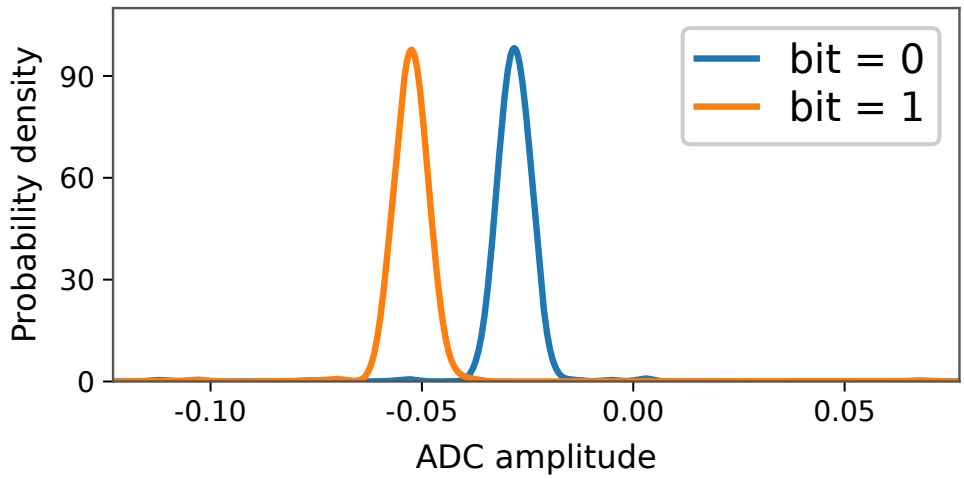


Figure 5: Power distributions at a selected sample point.

point during the processing of bit 0, as shown in Figure 5. Since the overlap between the two distributions reflects the difficulty of distinguishing between the bit values, their clear separation suggests that 0 and 1 can be distinguished easily. We will verify this experimentally in the following sections.

4 Experimental Setup

Our experiments evaluate the attack on AIMer v3 using the AIMer-128f parameter set, which has a 128-bit secret preimage. Power traces are captured with a ChipWhisperer-HuskyPlus connected to a CW308 UFO board and a CW308T-STM32F4 target board. The latter houses an STM32F415RGT6 microcontroller based on the ARM Cortex-M4 core. With the target running at 7.38 MHz, a sampling rate of 29.52 MS/s provides four samples per clock cycle. For experimental convenience, profiling and attack measurements use the same physical device.

The firmware is based on the reference C implementation of AIMer version 3.0, compiled with `arm-none-eabi-gcc` at optimization level `-O3`. Since full signature generation exceeds the memory available on the target, we execute the targeted functions in isolation, with minor simplifications to the surrounding code and parameters. CNN training and evaluation run on a PC equipped with an Intel Core i5-14600KF CPU at 3.50 GHz, 32 GB of RAM, and an NVIDIA GeForce RTX 5060 GPU with 8 GB of memory. We use the Adam optimizer with a learning rate of 10^{-3} and a batch size of 512. Training runs for up to 200 epochs, with an early-stopping patience of 30 epochs.

5 Attack Methodology

Attacker Model We consider a profiled single-trace attack against AIMer. In the profiling stage, we assume access to a device running the target implementation, allowing us to collect power traces using known secret keys. In the attack stage, we have access to the target public key and can record a power trace of a single signing execution. Our goal is to recover the secret key sk.

Table 2: CNN architecture and hyperparameters. For Conv/Pool outputs, $C \times L$ denotes C channels and L samples. (k, s, p) denotes kernel size, stride, and padding, respectively. BN denotes batch normalization.

Layer	Operator	Output	BN	Activation
Input	input	32	Yes	-
Conv1	conv1d $(5, 1, 2)$	16×32	Yes	ReLU
Conv2	conv1d $(3, 1, 1)$	32×32	Yes	ReLU
Pool	avgpool1d $(2, 2, 0)$	32×16	No	-
Conv3	conv1d $(3, 1, 1)$	64×16	Yes	ReLU
FLT	flatten	1024	No	-
FC1	dense	128	Yes	ReLU
FC2	dense	2	No	Softmax

Table 3: Bit classification accuracy by byte and bit position.

Byte	Bit position in byte								**avg**
	0	1	2	3	4	5	6	7	
0	0.9993	1.0000	0.9997	0.9997	1.0000	1.0000	0.9997	0.9983	0.9996
1	1.0000	1.0000	1.0000	1.0000	0.9997	1.0000	1.0000	0.9993	0.9999
2	0.9990	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9993	0.9998
3	0.9990	1.0000	1.0000	0.9997	1.0000	1.0000	1.0000	0.9993	0.9998
4	0.9997	1.0000	1.0000	0.9997	1.0000	1.0000	1.0000	1.0000	0.9999
5	0.9977	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9997	0.9997
6	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9993	0.9999
7	0.9997	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
8	0.9997	1.0000	0.9993	1.0000	1.0000	0.9997	1.0000	0.9997	0.9998
9	0.9997	1.0000	0.9997	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999
10	0.9993	1.0000	1.0000	1.0000	1.0000	0.9997	1.0000	0.9997	0.9998
11	0.9990	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9997	0.9998
12	0.9990	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9990	0.9998
13	0.9990	1.0000	0.9997	0.9993	1.0000	1.0000	1.0000	0.9997	0.9997
14	0.9993	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999
15	0.9997	0.9997	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999
avg	0.9993	1.0000	0.9999	0.9999	1.0000	1.0000	1.0000	0.9996	**0.9998**

Profiling Stage In the profiling stage, we collect 10,000 power traces T_i of the affine computation using randomly generated AIMer-128f keys with secret preimages denoted by pt_i. Each trace covers the processing of all 16 bytes of pt_i. We extract 32 consecutive samples for each bit from the windows where the Welch t-test indicates leakage, as shown in Figure 4. The segment $\mathbf{x}_{i,j} \in \mathbb{R}^{32}$ corresponding to bit j is assigned the label $y_{i,j} = \mathrm{pt}_i[j] \in \{0, 1\}$, where $0 \leq j < 128$. Bits are indexed in execution order, starting with the least significant bit of the first byte. The traces are divided into 70% for profiling and 30% for evaluation. The profiling set is further split into 80% for training and 20% for validation, with disjoint secret preimages across all three sets.

We employ convolutional neural networks (CNNs) to construct the classifiers, as they have demonstrated strong performance in previous side-channel analyses of post-quantum cryptographic algorithms [TUX+23, UXT+22]. The eight bit positions within a byte are processed by slightly different instruction sequences. Moreover, the Cortex-M4's three-stage pipeline allows subsequent instructions to be fetched and decoded before the current

Table 4: Bit classification accuracy and success rate of full key recovery before enumeration under added Gaussian noise.

Noise σ	0.000	0.001	0.002	0.003	0.004	0.005	0.006	0.007	0.008	0.009	0.010
p_{bit}	0.9998	0.9998	0.9993	0.9978	0.9938	0.9855	0.9725	0.9573	0.9384	0.9180	0.8990
p_{key}	0.9773	0.9703	0.9147	0.7577	0.4507	0.1557	0.0303	0.0050	0.0003	0.0000	0.0000

Table 5: The success rate of full key recovery for different enumeration sizes.

Enum	0	2^1	2^2	$\geq 2^5$
p_{key}	0.9773	0.9983	0.9993	1.0000

instruction finishes executing, which may make the trace segment for each bit depend on neighbouring instructions. We therefore train eight CNN classifiers using the architecture in Table 2, one for each bit position. Since the same loop body is repeated for each byte, classifier $\mathcal{N}_k : \mathbb{R}^{32} \to [0,1]^2$ is applied to position $0 \leq k < 8$ in all 16 bytes. Its output for segment $\mathbf{x}_{i,j}$ is

$$\mathbf{p}_{i,j} = \mathcal{N}_{j \bmod 8}(\mathbf{x}_{i,j}), \tag{3}$$

where $p_{i,j}(b)$ estimates the probability that the corresponding preimage bit equals $b \in \{0,1\}$ given the observed segment.

Attack Stage In the attack stage, we record a power trace T of the input affine computation during signing. We then extract the 128 bit segments $\mathbf{x}_j$ using the windows established during profiling and feed them to the trained classifiers. Let $p_j(b)$ denote the probability of bit value b for segment $\mathbf{x}_j$. We predict each bit by selecting the value with the higher probability:

$$\widehat{\text{pt}}[j] = \arg\max_{b \in \{0,1\}} p_j(b), \qquad 0 \leq j < 128. \tag{4}$$

Reassembling these predictions in their original bit order gives $\widehat{\text{pt}}$, which we combine with the public values iv and ct to obtain $\widehat{\text{sk}} = (\widehat{\text{pt}}, \text{iv}, \text{ct})$.

6 Experimental Results

We evaluate the attack over 3,000 tests without added noise using two metrics: bit classification accuracy p_{bit} and success rate of full key recovery p_{key}. Full key recovery requires all 128 preimage bits to be correct. The per-bit results in Table 3 yield a mean accuracy of 99.98%, while the success rate of full key recovery reaches 97.73%. Bit positions 0 and 7 within a byte have the lowest mean accuracies, 0.9993 and 0.9996, respectively. We then assess the effect of noise on prediction performance. Specifically, we add independent zero-mean Gaussian noise to each raw trace and then extract the 128 model inputs:

$$\widetilde{T}[i] = T[i] + \epsilon_i, \qquad \epsilon_i \sim \mathcal{N}(0, \sigma^2). \tag{5}$$

We use a ChipWhisperer-HuskyPlus for collecting traces, whose ADC represents trace amplitudes in the range $[-0.5, 0.5)$, so $\sigma = 0.010$ corresponds to 1% of the width of this range. As shown in Table 4, the success rate of full key recovery decreases from 0.9147 at $\sigma = 0.002$ to 0.4507 at $\sigma = 0.004$. At $\sigma = 0.010$, bit classification accuracy drops to 0.8990, and no full key recovery is achieved.

Algorithm 2 Full key recovery with enumeration

Input: $\mathrm{pk} = (\mathrm{iv}, \mathrm{ct})$, $(p_j(b))_{j,b}$, enumeration width w
Output: Recovered secret key sk or $\perp$

for $j \in [0..128)$ **do**
  $\widehat{\mathrm{pt}}[j] \leftarrow \arg\max_{b\in\{0,1\}} p_j(b)$
  $c_j \leftarrow \max_{b\in\{0,1\}} p_j(b)$
end for
$S \leftarrow \mathrm{argsort}(c)[0..w)$ ▷ Least confident bits
for $\mathrm{pt}' \in \mathcal{C}(S)$ in decreasing order of $s(\mathrm{pt}')$ **do**
  if $V_{\mathrm{pk}}(\mathrm{pt}') = 1$ **then**
    return $(\mathrm{pt}', \mathrm{iv}, \mathrm{ct})$
  end if
end for
return $\perp$

To further improve the success rate of full key recovery, we use the classifiers' confidence scores to guide enumeration. The procedure is presented in Algorithm 2. The w positions with the smallest scores c_j form S. The candidate set $\mathcal{C}(S)$ contains all 2^w assignments to these positions, with the other $128 - w$ bits fixed to their initial predictions. Candidates are ranked by the product score $s(\mathrm{pt}') = \prod_{j\in S} p_j(\mathrm{pt}'[j])$ and checked in decreasing score order. For each candidate, the public-key verifier V_{pk} rejects any zero S-box input and accepts only if $\mathrm{AIM3}(\mathrm{pt}', \mathrm{iv}) = \mathrm{ct}$. The accepted candidate yields the secret key $(\mathrm{pt}', \mathrm{iv}, \mathrm{ct})$, following AIMer key generation. As shown in Table 5, the success rate of full key recovery increases from 97.73% to 99.83% with a maximum enumeration complexity of 2^1 and improves further with a larger enumeration budget. We can recover the secret key sk with a success rate of 100% and a maximum enumeration complexity of 2^5.

Moreover, if the same secret key were reused, the preimage and public tweak would remain unchanged, so the targeted affine computation would process the same operands across signatures. Information from multiple traces could therefore be combined using majority voting or negative log-likelihood (NLL) aggregation [TUX+23, UXT+22] to further improve the attack.

7 Countermeasures

In this section, we discuss how masking and shuffling could mitigate the attack presented in this paper. We first consider masking [ISW03], a common countermeasure that splits sensitive values into random shares. For the targeted operation, the secret preimage could be split into shares whose XOR equals the original value [LWW+26]. Since `gf_mat_vec_mul` is linear in its secret input, it could process each share separately while keeping the outputs shared. Each share bit would be independent of the preimage bit, but the unchanged routine would still expand it into a mask filled with zeros or ones. If an attacker could classify the corresponding bits in all shares from one trace, their XOR would reveal the preimage. Masking the input alone may therefore not prevent an adaptation of our attack.

Another common countermeasure is shuffling [VCMKS12], which can randomize the order of the column contributions while preserving the XOR result. This can disrupt the correspondence between trace segments and bit positions; independent permutations for different shares would also complicate combining their bit estimates. However, shuffling does not eliminate the leakage of bit values, and leakage from indices or accesses to known matrix columns may reveal the processing order [BNGD23].

8 Conclusion

In this paper, we identified power leakage from the processing of secret preimage bits in AIM3's input affine layer. Expanding these bits into word masks creates a large Hamming weight contrast that allows their values to be inferred from power traces. By exploiting this leakage, we demonstrated an attack against the AIMer v3 reference implementation using CNN classifiers, achieving full key recovery from a single power trace. We also discussed masking and shuffling as potential countermeasures against the attack. These findings highlight the need to protect operations on the secret preimage against power leakage in AIMer implementations.

References

[BFU26] Alex Biryukov, Pablo García Fernández, and Aleksei Udovenko. Magic pot: Cryptanalysis of full AIM2 in the standard and related-/reused-key settings using new elimination framework. In *Advances in Cryptology (EUROCRYPT)*, pages 63–90. Springer, 2026. https://doi.org/10.1007/978-3-032-25333-0_3.

[BNGD23] Linus Backlund, Kalle Ngo, Joel Gärtner, and Elena Dubrova. Secret key recovery attack on masked and shuffled implementations of CRYSTALS-Kyber and Saber. In *Applied Cryptography and Network Security Workshops (ACNS)*, LNCS 13907, pages 159–177. Springer, 2023. https://doi.org/10.1007/978-3-031-41181-6_9.

[CHK+26] Jihoon Cho, Jincheol Ha, Seongkwang Kim, Jihoon Kwon, Byeonghak Lee, Joohee Lee, Jooyoung Lee, Sangyub Lee, Dukjae Moon, Mincheol Son, and Hyojin Yoon. The AIMer signature scheme, version 3.0, 2026. https://aimer-signature.org. Reference code available at https://github.com/samsungsds-opensource/AIMer.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology (CRYPTO)*, LNCS 263, pages 186–194. Springer, 1987. https://doi.org/10.1007/3-540-47721-7_12.

[GAG+25] Julie Godard, Nicolas Aragon, Philippe Gaborit, Antoine Loiseau, and Julien Maillard. Single trace side-channel attack on the MPC-in-the-Head framework. In *Post-Quantum Cryptography (PQCrypto)*, LNCS 15578, pages 267–293. Springer, 2025. https://doi.org/10.1007/978-3-031-86602-9_10.

[GAG+26] Julie Godard, Nicolas Aragon, Philippe Gaborit, Antoine Loiseau, and Julien Maillard. Non-supervised SASCA on the VOLEitH framework. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2026(2):468–494, 2026. https://doi.org/10.46586/tches.v2026.i2.468-494.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *ACM Symposium on Theory of Computing (STOC)*, pages 21–30. ACM, 2007. https://doi.org/10.1145/1250790.1250794.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *Advances in Cryptology (CRYPTO)*,

LNCS 2729, pages 463–481. Springer, 2003. https://doi.org/10.1007/978-3-540-45146-4_27.

[JD26] Sönke Jendral and Elena Dubrova. Side-channel attacks on VOLEitH signature schemes: Breaking masked FAEST. *IACR Communications in Cryptology (CiC)*, 2(4):24, 2026. https://doi.org/10.62056/aebngyl7s.

[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 *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 401–415. ACM, 2023. https://doi.org/10.1145/3576915.3616579.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology (CRYPTO)*, LNCS 1666, pages 388–397. Springer, 1999. https://doi.org/10.1007/3-540-48405-1_25.

[Kpq25] KpqC Team. Selected algorithms from the KpqC competition round 2, 2025. Announcement dated January 16, 2025. https://kpqc.or.kr/competition_02.html.

[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.

[LHZ+26] Jinnuo Li, Shu Han, Yunyan Zou, Fan Zhang, Puwen Wei, Weijia Wang, and Hao Cheng. DPA-style attacks on VOLE-in-the-head signature schemes. Cryptology ePrint Archive, Paper 2026/2519, 2026. https://eprint.iacr.org/2026/2519.

[LWW+26] Guowei Liu, Weijia Wang, Lixuan Wu, Chaoran Wang, Yanhong Fan, Jinliang Wang, and Meiqin Wang. Amortizing randomness cost: Efficient masked implementation of SDitH signatures with common shares. In *35th USENIX Security Symposium (USENIX Security 26)*, pages 517–531. USENIX Association, 2026. https://www.usenix.org/conference/usenixsecurity26/presentation/liu-guowei.

[TUX+23] Yutaro Tanaka, Rei Ueno, Keita Xagawa, Akira Ito, Junko Takahashi, and Naofumi Homma. Multiple-valued plaintext-checking side-channel attacks on post-quantum KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2023(3):473–503, 2023. https://doi.org/10.46586/tches.v2023.i3.473-503.

[UXT+22] Rei Ueno, Keita Xagawa, Yutaro Tanaka, Akira Ito, Junko Takahashi, and Naofumi Homma. Curse of re-encryption: A generic power/EM analysis on post-quantum KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2022(1):296–322, 2022. https://doi.org/10.46586/tches.v2022.i1.296-322.

[VCMKS12] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In *Advances in Cryptology (ASIACRYPT)*, LNCS 7658, pages 740–757. Springer, 2012. https://doi.org/10.1007/978-3-642-34961-4_44.