Optimizations and First-Order Masking of AIMer on Cortex-M4

DongCheon Kim[1], Jaeseung Han[1], Dong-Guk Han, Seog Chung Seo*

Department of Cybersecurity, Kookmin University, 77 Jeongneung-ro, Seongbuk-gu, Seoul 02707, Republic of Korea

Abstract

We optimize AIMer-128f on Cortex-M4 and propose first-order masking of its signing. Since squaring is a Frobenius map about ×16 cheaper than multiplication, we restructure AIM2's inverse-Mersenne S-box into a Frobenius-shifted addition chain, cutting multiplications per S-box from 24/23 to 9/10. Since one multiplication-check operand is constant across $N = 16$ parties, we build the partial-product table once and reuse it. Both are orthogonal to single-multiplication acceleration, and with a Radix-16 multiplier on NUCLEO-L4R5ZI, keygen/sign/verify drop 18.55%/3.94%/7.30%. 1-PINI gadgets on every sensitive operation give first-order probing security. TVLA finds no first-order leakage in 100,000 traces. Masking costs ×10.5, itself improved 5.20%.

Keywords: AIMer, Optimization, Masking, Cortex-M4, Binary field arithmetic

1. Introduction

A post-quantum signature algorithm has to be evaluated together with its cost and its side-channel resistance on embedded devices, which have constrained computational resources and are physically accessible. *AIMer* [1], an algorithm selected in the final round of the Korean Post-Quantum Cryptography (KpqC) competition, is a digital signature algorithm based on the multi-party computation in the head (MPCitH) paradigm. Its security rests on the one-wayness of AIM2, a symmetric-key primitive defined over $\mathbb{F}_{2^n}$, and most of its computational load therefore arises in finite-field arithmetic.

This load concentrates in two places. One is the inverse-Mersenne exponentiation of the AIM2 S-box, and the other is the multiplication check that multiplies the challenge ε and the opening α into the party shares. On an embedded device with limited computational resources this load leads directly to signing latency, and making the single multiplication faster cannot by itself remove the cost that this repeated structure creates. A structural optimization is therefore needed in addition to a faster single multiplication.

An embedded device, however, is also an environment in which an attacker can physically access the device and in which information such as the elapsed time, the power consumption and the emitted electromagnetic radiation carries little noise. Because of these characteristics an embedded environment can be particularly vulnerable to side-channel analysis (SCA), so a countermeasure is required. In AIMer signing the secret key `pt` is fed into the S-box exponentiation for every signature and is also used repeatedly to derive the party shares and the tree seeds, so the operations pointed out above as optimization targets are themselves the targets of SCA.

Masking is the standard implementation-level countermeasure against this. It splits every intermediate value of the target algorithm into random shares so that no individual intermediate value is statistically dependent on the secret, which neutralizes statistical analyses such as differential power analysis [2]. Masking, however, multiplies the cost of the operations it protects, so performance optimization and side-channel countermeasures are not unrelated goals but interlock on the same operations. This paper treats the two together in a single

*Corresponding author

Email addresses: kindongsy@kookmin.ac.kr (DongCheon Kim), jae1115@kookmin.ac.kr (Jaeseung Han), christa@kookmin.ac.kr (Dong-Guk Han), scseo@kookmin.ac.kr (Seog Chung Seo)

[1]These authors contributed equally to this work.

implementation.

1.1. Related Work

The most recent optimization study of AIMer is that of Kwon *et al.* [3], whose `m4speed` is the latest AIMer implementation currently carried in pqm4 [4], the Cortex-M4 post-quantum cryptography (PQC) benchmarking framework, and is the baseline of this paper. For the single multiplication we *borrow* the optimized 32-bit binary multiplication of Jang *et al.* [5] and stack it. On the algorithmic side, squaring in $\mathbb{F}_{2^n}$ is a Frobenius map and is far cheaper than multiplication, and the addition chain technique that uses this is known as the Itoh–Tsujii inversion [6].

Side-channel countermeasures for MPCitH signatures were first addressed by Aranha *et al.* [7], who mask the KKW proof system [8] with preprocessing at an arbitrary order and report a first-order protected Cortex-M4 implementation of Picnic3 [9]. They further analyze which hash calls must be masked, show that a substantial number can be left unmasked, and exploit the fact that even the calls requiring protection have only one of their input and output sensitive, applying a half-masking technique that masks only half the rounds of SHAKE. These results do not carry over directly to AIMer, whose proof system and core primitive differ; this paper therefore presents a complete masking tailored to the structure of AIMer signing.

1.2. Contributions

- ***Frobenius S-box Chain.*** We restructure the S-box exponentiation into a short addition chain that trades multiplications for the far cheaper squarings, reducing the multiplications per S-box from 24/23 to 9/10 and improving keygen by 12.60%.
- ***Constant-Operand Table Reuse.*** One operand of the multiplication check is constant across all parties, so we build the partial-product table once outside the party loop and reuse it, improving verify by 6.24%.
- ***Complete First-Order Masking of AIMer Signing.*** We classify every elementary operation of AIMer signing by the sensitivity of the values it handles and apply a first-order probe-isolating non-interference (1-PINI) gadget to all operations that require masking.
- ***Cortex-M4 Implementation and Empirical Leakage Evaluation.*** We implement the proposed masking scheme on an Cortex-M4 target and evaluate the isolated masked S-box using a fixed-vs-random Test Vector Leakage Assessment (TVLA). No statistically significant first-order leakage was detected in 100,000 traces per implementation.

The two optimizations are orthogonal to the single-multiplication acceleration, so stacking a constant-time Radix-16 multiplier accumulates the improvement to 18.55% for keygen, 3.94% for sign and 7.30% for verify. The gain grows under masking, where each multiplication becomes a 1-PINI gadget, shortening signing by 26.77% and lowering the masking overhead from 2.03× to 1.55×. To the best of our knowledge this is the first complete first-order masked implementation of AIMer.
The source code is provided in a public repository with the camera-ready version.

2. Background and Motivation

2.1. AIMer and Its Finite-Field Bottlenecks

The one-way function `AIM2` [1], which takes the secret plaintext `pt` $\in \mathbb{F}_{2^{128}}$ as input, combines three linear components with two parallel inverse-Mersenne S-boxes of exponents e_1=49 and e_2=91, and since each S-box is a power by the 128-bit inverse exponent $(2^e-1)^{-1}$, the two together require several hundred squarings and several tens of multiplications. Signing repeats T times in the MPCitH manner, and *AIMer-128f*, the variant targeted in this paper, uses N=16 parties and T=33 repetitions. Each repetition computes the commitment hash h_1 and the multiplication-check hash h_2 induced by ε, and verification reproduces the same check with the $N-1$ opened shares. Profiling in prior work points to binary-field multiplication as the second largest load after SHAKE [3]. Even on this baseline, whose single multiplication is already optimized in M4 assembly, the S-box accounts for 31.4% of keygen, the MPC simulation `aim2_mpc` for 27.8% of sign and of verify each, and the α/v multiplication check for about 17%. This load, which remains even after every optimization that makes the single multiplication faster, is the starting point of this paper.

The pqm4 implementation provides three primitives over $\mathbb{F}_{2^{128}}$. `GF_mul` looks up a nibble-wise table built from a public operand at 1,338 cycles, while

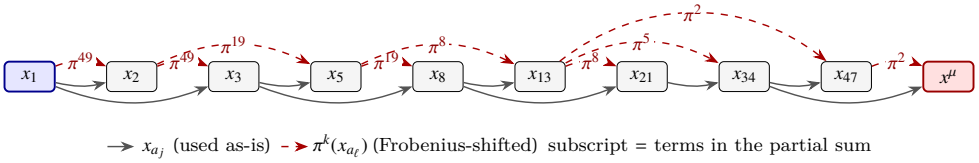


Figure 1. Frobenius-shifted addition chain for the S-box exponent $e_1 = 49$.

a product of two secret operands uses the lookup-free `GF_mul_s`, which recomputes the whole product with bit operations following the recommendation of Bernstein [10] and is the most expensive at 2,538 cycles. The squaring `GF_sqr_s`, implemented with a bit spread, costs only 164 cycles, about 1/16 of a secret multiplication, and this asymmetry is the starting point of Section 3.1.

2.2. Masking and Probing Security

Masking [2] represents a secret x by shares $(x_0, \ldots, x_{d-1})$ obtained by splitting x with randomness, so that no individual intermediate value is statistically dependent on x. Boolean masking is natural for AIMer, which operates over a binary field, and a first-order countermeasure is realized with d=2 by setting $x = x_0 \oplus x_1$ and drawing x_1 anew on every execution.

The security of masking is usually argued in the *probing model* [11]. When an attacker can observe t internal wires of the circuit and that observation is independent of the secret, the implementation satisfies t-th order probing security, and masking with d=2 corresponds to t=1. The difficulty is that the security of individual gadgets gives no guarantee that the circuit obtained by connecting them is secure.

Probe-isolating non-interference (PINI) [12], proposed by Cassiers and Standaert, resolves this difficulty. PINI requires *share isolation*, so that a probe in one share domain gives no information about the inputs of another share domain, and its key property is that an arbitrary composition of PINI gadgets is again PINI. If every operation constituting the circuit is PINI, the whole circuit therefore obtains $(d-1)$-th order probing security without inserting an additional refresh. A linear operation is trivially PINI because applying the original operation share by share makes output share i depend only on input share i. The only operation for which the trivial treatment is impossible is a product whose two operands are both secret, and only there is a dedicated gadget that handles the cross terms $a_i b_j$ required.

3. Proposed Work

3.1. Optimization

Frobenius-Shifted Addition Chain. This part adapts the Itoh–Tsujii addition chain to the inverse-Mersenne exponent of AIMer. The inverse-Mersenne S-box computes $x \mapsto x^\mu$ by an addition chain. Writing n for the extension degree and e for the S-box exponent, the target inverse exponent is $\mu \equiv (2^e - 1)^{-1} \pmod{2^n - 1}$. In characteristic 2 the squaring is the $\mathbb{F}_2$-linear Frobenius map π and π^k multiplies the exponent by 2^k, so squaring reduces to a data-independent shift.

The key observation is that μ is the sum of a geometric sequence of common ratio 2^e. Writing $d \equiv e^{-1} \pmod{n}$ for the number of terms,

$$\mu \equiv \sum_{i=0}^{d-1} 2^{ei} \pmod{2^n - 1} \tag{1}$$

holds, and in AIMer-128f $d = 81$ for $e_1 = 49$ and $d = 83$ for $e_2 = 91$. Writing T_r for the partial sum of the first r terms only, two partial sums combine as $T_{a+b} \equiv T_a + 2^{ea} T_b$, so at the element level this becomes $x^{T_{a+b}} = x^{T_a} \cdot \pi^{ea}(x^{T_b})$ and two partial sums can be merged with one multiplication and $ea \bmod n$ squarings. Computing x^μ therefore reduces to the problem of finding an addition chain that starts at 1 and reaches d, and the number of multiplications equals the length of that chain.

The actual cost of a chain depends on the cost of one multiplication and on the shift length of each step, so we obtain the minimum-cost chain by an offline shortest-path search weighted with the measured primitive costs.

The search yields a chain with 9 multiplications for $e_1 = 49$ and 10 for $e_2 = 91$, shown in Fig. 1. As Table 1 shows, the reduced multiplications outweigh the added squarings, so the cost of one S-box pair decreases from 153,945 to 104,415 cycles, a reduction of 32.2%, and this leads to a 10.1% improvement of keygen, which the S-box dominates. Adding a fused assembly routine that handles the consecutive squarings produced by the chain in

Table 1
Inverse-Mersenne S-box cost with and without the Frobenius chain.

S-box	GF_mul calls	GF_sqr calls	Cycles
Baseline (e_1)	24	127	78,216
Frobenius (e_1)	9	161	45,653
Baseline (e_2)	23	127	75,729
Frobenius (e_2)	10	236	58,762

one register set, the keygen improvement becomes 12.60%.

Constant-Operand Table Reuse across Parties. Verification recomputes the broadcast values for every opened party in each repetition [1].

$$\begin{aligned} \alpha^{(i)} &= a^{(i)} + \textstyle\sum_{j=1}^{\ell+1} \varepsilon_j \, x_j^{(i)}, \\ v^{(i)} &= c^{(i)} + \textstyle\sum_{j=1}^{\ell+1} \varepsilon_j \, z_j^{(i)} - \alpha \, \mathtt{pt}^{(i)} \end{aligned} \tag{2}$$

Here ℓ is the number of S-boxes and $a^{(i)}$ and $c^{(i)}$ are shares obtained from the tape. In these expressions ε_j and α, one side of the multiplication, stay the same as the party changes, and only the share on the other side differs from party to party. This part uses this property to lift the setup stage of the multiplier out of the party loop.

The native `GF_mul` of pqm4 splits into a *setup* stage that builds the sixteen-entry partial-product table from one operand and a *use* stage that looks it up nibble by nibble. The reference goes through both stages for every multiplication and therefore performs the setup $3N$ times in one repetition, and since the operand it is multiplied with is the same across the parties, the $3(N-1)$ times that remain after subtracting 3 of them rebuild the same table. Moving the table out of the party loop removes this redundancy, and the gain grows as the number of parties grows.

Two-level Karatsuba divides a 128×128 product into nine 32×32 blocks, so one constant operand has nine sets of tables, and the constant operands that verification uses are four, three ε_j and one α. One entry is 4 byte, so the additional memory is 2,304 byte, which is less than 0.4% of the 640 KB of RAM the target board has.

Phase 2/3 of signing has the same structure, so the same technique can be applied. In that case, however, the index that looks the table up becomes a secret share held by the signer and a secret-dependent memory access arises, whereas the index looked up in verification is only a value reconstructed from the public signature and challenges. This paper therefore applies this technique to verify only and keeps the table lookup-free of pqm4 in sign so that the constant-time implementation is preserved.

Table 2
Classification of AIMer signing operations.

Operation	Input	Secret
H_3	$\mathtt{pt}, \mu, \rho$	**Yes**
ExpandTree (H_4)	$\mathtt{seed}_k$	**Yes**
H_5	$\mathtt{seed}_k^{(i)}$	**Yes**
AIM2_MPC, Sqr, Add	shares	**Yes**
Mul (secret×public)	share, public	**Yes**
Mul (secret×secret)	shares	**Yes**
H_0, GenerateLinear	$\mathtt{iv}, \mathtt{ct}, m$	No
H_1, H_2	$\mu, \sigma_1, h_1, \sigma_2$	No
ExpandH1, ExpandH2	h_1, h_2	No
RevealAllBut	$\mathtt{nodes}_k, \bar{i}_k$	No

3.2. Masking

Classification of Operations. The starting point of a masking design is to define which operations handle secrets. Side-channel leakage occurs at the moment a value is processed, so the criterion is the secrecy of the inputs an operation absorbs rather than whether its output is public. Table 2 classifies all operations of AIMer signing under this criterion. The classification is immediate except for H_1 and H_2, whose absorbed values a verifier holding a valid signature can recover and which are therefore regarded as public. H_3, ExpandTree (H_4) and H_5, by contrast, absorb the secret key `pt` or seeds derived from it, and are subject to masking.

Masked Field Arithmetic. By the observation in Section 2.2 every operation of the masked signing circuit must satisfy PINI, and in AIMer most of them hold trivially: addition over $\mathbb{F}_{2^{128}}$, squaring (Frobenius) and multiplication by a public matrix are $\mathbb{F}_2$-linear, and the multiplications in `aim2_mpc` and in Phase 2/3 multiply a public value such as the challenge ε or the opening α into a secret share, so leaving the public operand as it is and multiplying it into each share is again trivially PINI. A dedicated gadget is required only for the products whose two operands are both secret, namely those inside the S-box exponentiation and $a^{(i)} \cdot \mathtt{pt}$.

This paper implements and compares both multiplication gadgets that CS20 [12] proves to be PINI. Alg1 is a double-SNI construction that applies an SNI refresh to one operand and then performs an ISW multiplication, using 4 multiplications and 2 random values at d=2; Alg2 (PINI1) hides the cross

Table 3
Cycle counts and improvement over the pqm4 baseline.

Build	keypair	sign	verify
Base	490,317	28,508,888	26,686,122
Ours	428,544	28,450,568	25,019,785
	(+12.60%)	(+0.20%)	(+6.24%)
R16 [5]	422,206	27,408,949	25,759,933
	(+13.89%)	(+3.86%)	(+3.47%)
Ours+R16	399,351	27,386,539	24,737,228
	(+18.55%)	(+3.94%)	(+7.30%)

terms directly in the form $z_{ij} = (1+a_i)r+a_i(r+b_j)$, using 6 multiplications and 1 random value. Writing M for the cost of one multiplication and R for that of generating one random value, ALG1 is advantageous when $R < 2M$. This paper provides both variants so that a choice can be made.

Here the Frobenius chain produces a direct synergy with masking. In an unmasked implementation the reduction of the S-box multiplications ($47 \rightarrow 19$) saves the cost of those multiplications, whereas in a masked implementation each one of them is a call to a PINI gadget that needs an internal refresh, so the randomness consumption falls together with it.

Masked Hash. H_3, H_4 and H_5 are all realized with SHAKE128, so protecting them requires masking the Keccak-f[1600] permutation itself. In its round function θ, ρ, π and ι are $\mathbb{F}_2$-linear and trivially PINI when applied share by share, and the only nonlinear layer is the AND of χ. Unlike a field $\mathbb{F}_{2^{128}}$ multiplication the AND is light, so $R > 2M$ holds and the masked AND adopts ALG2. Every operation of the round function is then PINI, hence so are the round function and the whole hash.

4. Evaluation

4.1. Performance

All measurements are performed with pqm4 on a 20 MHz Cortex-M4 STM32L4R5ZI, and Table 3 summarizes the results per build.

Our own two techniques `Frob` and `Tbl` are implemented with the existing pqm4 primitives only and reduce different operations, so their improvements are combined as they are in `Ours`. The borrowed Radix-16 [5] reduces the cost of one multiplication, so placing our techniques on top of it superimposes the two improvements and `Ours+R16` reaches keygen 18.55%, sign 3.94% and verify 7.30%. This paper takes this stack as the default deployment.

Table 4 is the cost of applying first-order masking to the signing path. The masked signature is about 10.5× the unmasked one, and most of this overhead comes from the hashes and their randomness generation. The lower block of the table is a build in which the hashes are not masked and only the Galois field (GF) operations are. The cost that hash masking adds is common to both builds and enlarges the denominator, so the improvement rate of the stacked deployment is 5.20% when the hashes are masked as well and 26.77% when only the GF operations are masked.

Table 4
Cycle counts of first-order masked signing.

Build	Gate	sign	vs. unmasked	Δ over Base
GF operations and hashes masked				
Base	Alg 1	298,148,139	×10.46	—
	Alg 2	298,523,082	×10.47	—
Ours+R16	Alg 1	282,902,888	×10.33	+5.11%
	Alg 2	283,006,645	×10.33	+5.20%
GF operations masked, hashes unmasked				
Base	Alg 1	57,603,176	×2.02	—
	Alg 2	57,978,305	×2.03	—
Ours+R16	Alg 1	42,358,293	×1.55	+26.47%
	Alg 2	42,460,325	×1.55	+26.77%

The optimization also lowers the cost that masking adds. The factor over the unmasked implementation is 2.03 for Base and 1.55 for the stacked deployment, because the optimization aimed at multiplication reduces in advance the operations that masking expands.

The stack that the whole masked signing path requires is 211 KB, which operates comfortably within the 640 KB of RAM the target board has.

4.2. Side-Channel Leakage Evaluation

Experimental Setup. The side-channel evaluation is performed on an STM32F4 (Cortex-M4) target mounted on the CW308 UFO board of the ChipWhisperer [13] platform, and the traces are acquired through a CW-Husky board at a sampling rate of 10MS/s. The evaluation target is not the whole signature but a target in which the inverse-Mersenne S-box, the core of the masking, is isolated, and one S-box is evaluated. The objects of comparison are `Ours+R16` without masking and the same build with ALG1 masking.

Evaluation Method. We use the non-specific fixed-vs-random test of the TVLA [14]. The fixed group keeps `pt` constant and the random group draws `pt` anew for every trace, and the two groups are acquired in a randomly mixed order so that environmental variation is reflected equally in both groups. At each time point we compute the Welch

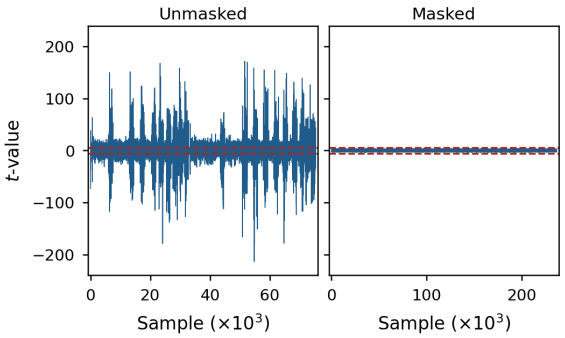


Figure 2. S-box TVLA with 100,000 traces. Left: `Ours+R16` without masking ($\max|t|$=212.95, threshold 5.27). Right: the same build with ALG1 masking (4.70, threshold 5.48).

t statistic of the two groups and compute a threshold for each from the significance level $\alpha = 0.01$ and a multiplicity correction according to the trace length, which we take as the criterion for judging leakage. For each implementation we collected 50,000 traces for the fixed group and 50,000 traces for the random group.

Experimental Result. Fig. 2 compares `Ours+R16` without masking and the same build with ALG1 masking, so that the difference isolates the effect of masking alone. Without masking $|t|$ exceeds its threshold by more than an order of magnitude, which serves as a positive control showing that the evaluation procedure is able to detect real leakage. With masking $|t|$ stays below its threshold at the same number of traces, so no first-order leakage was observed.

5. Conclusion

This paper showed that a structural optimization of AIMer stacks on top of a faster single multiplication, and built a complete first-order masking on the resulting code with no first-order leakage observed. The optimization matters more under masking, where each multiplication becomes a PINI gadget, so the two should be designed together rather than in sequence.

Conflict of interest

The authors declare that there is no conflict of interest in this paper.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. RS-2025-25411243).

References

[1] J. Lee, J. Cho, J. Lee, S. Kim, J. Ha, M. Son, B. Lee, D. Moon, S. Lee, J. Kwon, H. Yoon, The AIMer signature scheme, version 2.1, KpqC Competition submission document (2024).
URL https://aimer-signature.org/

[2] S. Chari, C. S. Jutla, J. R. Rao, P. Rohatgi, Towards sound approaches to counteract power-analysis attacks, in: CRYPTO 1999, Vol. 1666 of LNCS, 1999, pp. 398–412.

[3] J. Kwon, S. Lee, B. Lee, H. Seo, J. Cho, Efficient implementations of AIMer post-quantum signature scheme for low-end to high-end IoT devices, IEEE Internet of Things Journal 12 (22) (2025) 46817–46837. doi:10.1109/JIOT.2025.3597415.

[4] M. J. Kannwischer, M. Krausz, R. Petri, S.-Y. Yang, pqm4: Benchmarking NIST additional post-quantum signature schemes on microcontrollers, in: Fifth NIST PQC Standardization Conference, 2024, https://eprint.iacr.org/2024/112.

[5] J. Jang, M. Lee, D. Kwon, S. Hong, S. Kim, S. Lee, Efficient polynomial multiplication for HQC on ARM Cortex-M4, Cryptology ePrint Archive, Paper 2025/1939 (2025).
URL https://eprint.iacr.org/2025/1939

[6] T. Itoh, S. Tsujii, A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases, Information and Computation 78 (3) (1988) 171–177.

[7] D. F. Aranha, S. Berndt, T. Eisenbarth, O. Seker, A. Takahashi, L. Wilke, G. Zaverucha, Side-channel protections for Picnic signatures, IACR Transactions on Cryptographic Hardware and Embedded Systems 2021 (4) (2021) 239–282.

[8] J. Katz, V. Kolesnikov, X. Wang, Improved non-interactive zero knowledge with applications to post-quantum signatures, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 525–537.

[9] D. Kales, G. Zaverucha, Improving the performance of the picnic signature scheme, IACR Transactions on Cryptographic Hardware and Embedded Systems (2020) 154–188.

[10] D. J. Bernstein, AIMer software recommendations, KpqC-bulletin, https://groups.google.com/g/kpqc-bulletin/c/_Gb2n7ZwlTo/m/K9lwRPiwBwAJ (2024).

[11] Y. Ishai, A. Sahai, D. Wagner, Private circuits: Securing hardware against probing attacks, in: CRYPTO 2003, Vol. 2729 of LNCS, 2003, pp. 463–481.

[12] G. Cassiers, F.-X. Standaert, Trivially and efficiently composing masked gadgets with probe isolating non-interference, IEEE Transactions on Information Forensics and Security 15 (2020) 2542–2555.

[13] C. O'Flynn, Z. Chen, ChipWhisperer: An open-source platform for hardware embedded security research, COSADE 2014 (2014).

[14] G. Goodwill, B. Jun, J. Jaffe, P. Rohatgi, A testing methodology for side-channel resistance validation, in: NIAT (Non-Invasive Attack Testing) Workshop, 2011.