

Faultless Key Recovery

Iteration-Skip and Loop-Abort Fault Attacks on LESS

Xiao Huang¹, Zhuo Huang², Yituo He², Quan Yuan³,
Chao Sun^{1,✉}, Mehdi Tibouchi⁴ and Yu Yu²

¹ Southeast University, Nanjing, China, 220245548@seu.edu.cn, sunchaopku12345@gmail.com

² Shanghai Jiao Tong University, Shanghai, China, sh1kaku@sjtu.edu.cn,
yituo_he@sjtu.edu.cn, yuyu@yuyu.hk

³ Shandong University, Qingdao, China, yuanquan@sdu.edu.cn

⁴ NTT Social Informatics Laboratories, Tokyo, Japan, mehdi.tibouchi@ntt.com

Abstract. To enhance the diversity of basic hard problems underlying post-quantum cryptography (PQC) schemes, NIST launched an additional call for PQC signatures in 2023. Among numerous candidate schemes, several code-based ones, which have successfully advanced to the second round, are constructed by applying the Fiat–Shamir transform to the parallel repetition of a (relatively low soundness) commit-and-prove sigma protocol similar to the Stern identification scheme.

In Fiat–Shamir-based signatures, it is well-known that key material will be leaked if an attacker can somehow obtain what amounts, in the sigma protocol, to the responses to different challenges with respect to the same commitment. This idea is for example at the basis of a famous differential fault attack against deterministic Fiat–Shamir-based signatures like EdDSA. It is usually difficult to mount a fault injection attack based on that principle against a properly randomized Fiat–Shamir-based scheme however (at least with single faults): since commitment collisions are ruled out, it typically involves obtaining the responses to multiple challenges with respect to the same commitment within a single execution of the signature, which is often impossible by construction (e.g., because the extra information will not fit in a single signature, or because it is hard to force the computation of both responses).

Due to the comparative inefficiency of signatures based on Stern-like protocols with parallel repetition, candidate constructions are led to use clever compression techniques to reduce signature size, in a way that increases the attack surface for physical attacks. In this paper, we demonstrate this against the LESS signature scheme, which uses so-called GGM trees for signature compression. We propose a simple fault attack on the construction of a binary array used to build the GGM tree, and show that a small number of faulty signatures suffice for full key recovery.

We provide a thorough mathematical model of the attack as well as extensive experimental validation with glitch attacks on a ChipWhisperer board, showing that, depending on the target parameter set and the precise fault model we consider, full key recovery can very often be achieved with just one or two faulty signatures, and never more than a couple hundred even in the least favorable scenario for the attacker.

Keywords: Fault Analysis · Post-Quantum Cryptography · LESS Signature Scheme · ChipWhisperer

1 Introduction

In 2016, the National Institute of Standards and Technology (NIST) launched the post-quantum cryptography standardization project [CML17] and called for post-quantum

cryptography (PQC) proposals. After 4 rounds of selection, Crystals-Kyber [SAB⁺22], Dilithium [LDK⁺22], FALCON [PFH⁺22], SPHINCS+ [HBD⁺22] and the later added scheme HQC [AAB⁺22] were selected for standardization.

Remarkably, three of those five selected schemes are lattice-based, leading to concerns regarding the repercussions of a hypothetical breakthrough in the cryptanalysis of lattice problems, structured or otherwise. Moreover, the selection process saw the elimination of certain candidates perceived as strong contenders, like Rainbow [DCP⁺20] and SIKE [JAC⁺20], at quite a late stage due to dramatic developments in the cryptanalysis of the respective underlying assumptions [Beu22, CD23, Rob23]. This made it desirable to further diversify the types of hard problems future post-quantum standards could be built upon. To do so, NIST launched an additional call for post-quantum signatures in 2023 that received numerous submissions based on a variety of assumptions related to multivariate equations, elliptic curve isogenies, codes, hash functions and more.

Code-based schemes in particular were well-represented among the candidates to this additional call, with the submission of three candidates: LESS [BBB⁺24b], CROSS [BBB⁺24a], and Wave [BCC⁺23] (LESS and CROSS have since progressed to the second round). This is interesting as, while code-based cryptography has long been a major source of encryption schemes, dating back to the McEliece cryptosystem [McE78] back in the infancy of public-key cryptography (and all the way to recent schemes like HQC), code-based signatures are far less common. Prior examples mostly include the Courtois–Finiasz–Sendrier (CFS) scheme [CFS01], some versions of which have faced significant security issues [FGO⁺13, PT16], and comparatively inefficient schemes obtained via the Fiat–Shamir transform from code-based identification protocols like Stern’s [Ste94]. Wave can be seen as a rigorous approach to CFS-like hash-and-sign constructions, while LESS and CROSS are modern, optimized takes on the Fiat–Shamir approach.

This paper focuses on LESS, which relies on the hardness of the Linear Equivalence Problem (LEP) [SS13]. First proposed in [BMP20], LESS was argued to be secure against both traditional and quantum adversaries, with specific resistance to the Leon [Leo82] and SSA [Sen99] attack algorithms. However, this original version was compromised shortly after by an improved Leon algorithm [Beu20]. To defend against this attack and reduce the signature size, an improvement [BBPS21] was proposed for the challenge phase, which introduced multiple secret keys and fixed the challenge weight. Later, the Information Set Linear Equivalence (IS-LEP) approach was proposed in [PS23] and adapted to LESS. This reduced communication costs by half by verifying the systematic form (SF) of non-pivot columns rather than the full matrix. These improvements were integrated into LESS-v1 [BBB⁺23], which was submitted to the first round of the additional call. To further optimize the signature size and accelerate the signing process, some new improvements have also been applied, including the introduction of “canonical form” (CF) in [CPS23], as well as optimized computation of the “row reduced echelon form” (RREF) and CF. Meanwhile, new features such as “pivot column reuse”, “variable-time RREF”, and “blinding techniques” were discussed in [BEP⁺25] to further improve the scheme. These recent improvements were incorporated into LESS-v2 [BBB⁺24b] submitted to NIST.

Another notable requirement in the additional call is that NIST emphasized the significance of side-channel attack resistance and implementation robustness for candidate schemes. This is because even if a scheme is provably secure, physical attacks remain a significant threat to its security. This domain has been extensively explored for lattice-based candidates, with significant quantitative research carried out regarding their physical vulnerabilities [IMS⁺22, FKK⁺22, ZLYW23, UMB⁺23, UMB⁺23, GGHS25, AWK⁺25, HPT25]. In contrast, equivalent research efforts for code-based schemes remain scarce. Although there have been several studies targeting McEliece [STM⁺08, PGD⁺22, GJJ22, BCM⁺23], physical attacks on signatures are few and far between: there appears to be only two published works related to LESS [MAKK24, CMR25] and one other work on

CROSS [SS25].

Yet, there are reasons to believe that schemes like LESS and CROSS may have structural vulnerabilities with respect to physical attacks and particularly fault analysis. Indeed, as discussed earlier, they are constructed by applying the Fiat–Shamir transform [FS87] to underlying commit-and-prove identification protocols (or rather, since those protocols have relatively low soundness, to many parallel repetitions of such protocols). Such protocols have the “special soundness” property that, if one can obtain the prover’s response to two different challenges with respect to the same commitment, one can directly extract the prover’s secret key. This property makes Fiat–Shamir-based signatures with *deterministic* commitment randomness very vulnerable to differential fault analysis, as demonstrated in [ABF⁺18, PSS⁺18, BP18] against schemes like EdDSA [BDL⁺11] and deterministic variants of ECDSA or Dilithium. Properly randomized Fiat–Shamir-based schemes cannot be attacked directly in the same way, however (at least not with single faults): since commitment collisions are ruled out, it requires obtaining the responses to multiple challenges with respect to the same commitment within a single execution of the signature, which is often impossible by construction (e.g., because the extra information will not fit in a single signature, or because it is hard to force the computation of both responses). This should also be the case for schemes with parallel repetition that apply Fiat–Shamir in a straightforward way; but doing so to the underlying identification protocols of schemes like LESS and CROSS would result in very inefficient signatures, which is why those schemes are much more clever in the way they pack the responses into a single signature. LESS, for example, relies on so-called GGM trees, making it less straightforward to ensure that overlapping responses will not be inadvertently released if the signature process is tampered with. Exploring the concrete security implications of that type of optimization is the main motivation of the current study, and may have broader ramifications beyond the specific case of the LESS signature schemes.

1.1 Our Contributions

In this paper, we propose a fault attack on the latest LESS-v2 scheme. Compared with existing attacks [MAKK24, MKN⁺25], our contributions can be summarized as follows.

Novel Fault Injection Target on LESS-v2. We propose a novel fault injection strategy targeting the `for` loop responsible for binary array generation before the seed tree construction in LESS-v2. This approach is significantly different from existing fault attack strategies, which typically corrupt the complex seed tree structure. Unlike previous approaches, our method enables the direct derivation of the fault location from applied attack parameters, without guesswork. By exploiting the target `for` loop’s linearity and isochronicity, we simplify fault analysis to a binary determination of fault occurrence at a single, precisely controlled location, avoiding heuristic fault position inference within complex data structures. Compared to tree-structured fault injection methodologies, our attack provides greater flexibility in targeting fault injection positions. Once a successful fault injection is performed, the adversary only needs to retain the same attack parameters and shift the fault trigger time by an integer multiple of the clock cycles for one loop iteration; a new fault can then be triggered at other sequential locations with high probability. This flexibility mitigates timing variations that may cause high failure rates when repeatedly targeting fixed locations in real-world settings. Furthermore, this dynamic injection capability equips adversaries with a more adaptable strategy, especially when extra information about the signing program’s internal state is available.

Comprehensive Fault Modeling. To make the attack practical, we analyzed the official C implementation of LESS-v2 across diverse hardware and operating systems and found that,

although the target array is uninitialized in the C code, two distinct internal states occur with high probability: the zero-filled case, where the system consistently populates the array with zeros automatically, and the statically-addressed case, where the array maintains its value from previous execution contexts, resembling the behavior of a static array. We also consider two distinct fault models that are frequently discussed in the literature: the “Single Iteration Skip” model, where only one iteration in the target `for` loop is skipped; and the “Loop Abort” model, in which the injected fault causes all subsequent iterations of the loop to be skipped at once. In accordance with expectations and previous works, our tests and experiments confirm that, in practice, most common array element states are as described above, and triggering the two faults is not difficult.

Efficient Key Recovery Algorithm. We introduce a novel, two-phase algorithm to efficiently recover the secret monomial matrices making up the signing key. In this key recovery algorithm, the first phase focuses on recovering the permutation of $\mathbf{Q}_{i^*}$ using $2 \log_2 n$ bits of leakage about it. By collecting these bits, we can extract intermediate values from the signing process without the key and retrieve the corresponding permutation via the set intersection method: each relevant leakage bit halves the size of the candidate set for specific position subsets in the permutation. Once the permutation is known, the second phase proceeds to completely recover the coefficients of $\mathbf{Q}_{i^*}$. This procedure successfully reconstructs an equivalent monomial matrix, $\mathbf{Q}'_{i^*}$, which is guaranteed to satisfy the crucial relationship $\text{RREF}(\mathbf{G}_0(\mathbf{Q}'_{i^*})^{-1}) = \text{RREF}(\mathbf{G}_0\mathbf{Q}_{i^*}^{-1})$. We prove that the recovered matrix $\mathbf{Q}'_{i^*}$ only differs from the actual secret monomial matrix $\mathbf{Q}_{i^*}$ by a non-zero scalar factor, making it effectively equivalent for all practical purposes. Even in the worst setting for the attacker, our results show that fewer than 300 faulty signatures suffice to recover all secret monomial matrices with high probability. In the Loop-Abort model, it is even the case that one or two faulty signatures are often sufficient.

Graph-Based Filtering via Disjoint Set Union. In practice, fault injection attacks are unpredictable, meaning that the success rate is not 100%. Consequently, when collecting signature results, valid faulty signatures generated by successful attacks are mixed with invalid signatures resulting from unsuccessful attempts. Thus, extracting valid information for key recovery from large-scale collected signature datasets is equally critical. To address this challenge, we propose a novel graph-based filtering algorithm leveraging the disjoint set union (DSU) data structure for the efficient validation of collected information. Our method is founded on verifying the consistency of information pairs that successfully yield partial secret permutations. This validation relies on two key properties: First, the candidate sets derived from valid information pairs must be mutually disjoint. Second, there must exist an exact, verifiable relationship between the size of these candidate sets and their observed frequency in the combined dataset. By leveraging the DSU data structure, our algorithm efficiently constructs connected components among all valid faulty signatures, enabling the rapid identification of the largest consistent data subset and the subsequent extraction of a vast majority of valid attack results.

1.2 Related Work

As mentioned previously, only a handful of papers so far appear to have explored physical attacks against the LESS and CROSS code-based schemes.

In passive side-channel analysis, [CMR25] proposed a comprehensive key-recovery attack on LESS and CROSS, exploiting the link between short-term and long-term secrets in secret-vector/public-matrix multiplication to extract the full private key from a single power trace. Similarly, [SS25] designed a multi-stage CPA attack through a horizontal

side-channel attack on CROSS. It targets the syndrome calculation and achieves stepwise secret recovery with at most two power traces needed.

Regarding fault attacks, we are aware of only one published work on LESS-v1 [MAKK24] as well as a very recent preprint [MKN⁺25] applicable to LESS-v2.

The authors of [MAKK24] conducted a comprehensive fault analysis of LESS-v1 and CROSS. By conducting an attack against the seed-tree generation process, some internal nodes are leaked, which causes secret key leakage. However, that approach requires guessing the fault location, and it does not apply to LESS-v2 due to significant structural differences between the two schemes. In contrast, our work targets the current scheme LESS-v2 and requires no fault location guessing.

The attack of [MKN⁺25] again targets the GGM tree construction process itself and still involves fault location guessing. Moreover, it falls short of actual key recovery and only provides a way to forge signatures via fault injection. Unlike that work, our attack successfully recovers the secret key and again does not involve fault location guessing.

2 Preliminaries

2.1 Notations

In this paper, we use $\mathbb{Z}_q = \{0, 1, \dots, q-1\}$ to denote the least non-negative residues modulo q . Accordingly, we use $\mathbb{Z}_q^n$ and $\mathbb{Z}_q^{n \times m}$ to denote the sets of n -dimensional vectors and $n \times m$ matrices over $\mathbb{Z}_q$, respectively. Throughout, vectors and arrays are denoted by lowercase bold letters (e.g., $\mathbf{v}$), and matrices by uppercase bold letters (e.g., $\mathbf{A}$). We also reserve specific symbols for dedicated use: calligraphic $\mathcal{C}$ denotes linear codes, and combinations are represented by upright C_n^m (e.g., $C_4^2 = 6$).

Before introducing LESS and our attack methodologies, we also need to provide some extra definitions:

Definition 1 (Monomial matrix). An $n \times n$ **monomial matrix** is a square matrix with exactly one nonzero entry in each row and each column. It has the form $\mathbf{Q} = \mathbf{P}\mathbf{D}$ or $\mathbf{Q} = \mathbf{D}\mathbf{P}$ where $\mathbf{P}$ is a permutation matrix and $\mathbf{D}$ is an invertible diagonal matrix. In this paper, we use M_n to represent the set of $n \times n$ monomial matrices over $\mathbb{Z}_q$.

Definition 2 (Linear code). An (n, k) -linear code $\mathcal{C}$ over $\mathbb{Z}_q$ is a k -dimensional subspace of the vector space $\mathbb{Z}_q^n$. Equivalently, $\mathcal{C}$ can be described as the row space of a full-rank generator matrix $\mathbf{G} \in \mathbb{Z}_q^{k \times n}$. Each row vector of matrix $\mathbf{G}$ can be viewed as a codeword, and for any codeword $\mathbf{c} \in \mathcal{C}$, there exists a message vector $\mathbf{u} \in \mathbb{Z}_q^k$ such that $\mathbf{c} = \mathbf{u}\mathbf{G}$.

Definition 3 (Equivalent linear code [BMPS20]). Two (n, k) -linear codes $\mathcal{C}_1, \mathcal{C}_2$ with corresponding generators $\mathbf{G}_1, \mathbf{G}_2 \in \mathbb{Z}_q^{k \times n}$ are said to be **equivalent** if there exists a monomial matrix $\mathbf{Q} \in M_n$ and an invertible matrix $\mathbf{C} \in \mathbb{Z}_q^{k \times k}$ such that $\mathbf{G}_2 = \mathbf{C}\mathbf{G}_1\mathbf{Q}$.

Definition 4 (Linear equivalence problem [SS13]). For two equivalent (n, k) -linear codes $\mathcal{C}_1, \mathcal{C}_2$ with generators $\mathbf{G}_1, \mathbf{G}_2 \in \mathbb{Z}_q^{k \times n}$, find a solution matrix $\mathbf{Q} \in M_n$ such that $\mathbf{G}_2 = \mathbf{C}\mathbf{G}_1\mathbf{Q}$ as in Definition 3.

Definition 5 (Disjoint set union [Tar75]). A **disjoint set union** (DSU) is a data structure that maintains a collection of pairwise disjoint dynamic sets $S_0, S_1, \dots, S_{m-1}$, with $\bigcup_{i=0}^{m-1} S_i = U = \{0, 1, \dots, n-1\}$. For any $x, y \in U$, it supports the following operations:

- Query the set to which x belongs.
- Merge the sets containing x and y .

To support these operations, each set has a representative element, and the structure is maintained by the array $\mathbf{fa} : U \rightarrow U$, where $\mathbf{fa}_x = y$ denotes that the representative element of the set containing x is y . The key operations on array $\mathbf{fa}$ are as follows:

- **MakeDSU(n)**: Initialize $\mathbf{fa}_i \leftarrow i$ for all $i = 0, 1, \dots, n-1$, so that each element initially forms an independent set.
- **Find(x)**: Return the representative element of the set containing x :
 - If $\mathbf{fa}_x = x$, x is the representative element, and return x immediately.
 - If $\mathbf{fa}_x \neq x$, recursively evaluate **Find**($\mathbf{fa}_x$) until a result is obtained. Then set $\mathbf{fa}_x \leftarrow \mathbf{Find}(\mathbf{fa}_x)$ to compress the query path and return $\mathbf{fa}_x$ as the result.
- **Union(x, y)**: Merge the two sets containing x and y by setting $\mathbf{fa}_{\mathbf{Find}(x)} \leftarrow \mathbf{Find}(y)$.

2.2 Linear Equivalence Signature Scheme

The Linear Equivalence Signature Scheme (LESS) is adapted from the Sigma protocol (see Figures 1 and 2 in [BBB⁺23]) by iterating it t times through the Fiat–Shamir transform [FS87]. In the Sigma protocol, the signer convinces the verifier of their knowledge of the secret matrix $\mathbf{Q} \in \mathbf{M}_n$ relating two equivalent (n, k) -linear codes $\mathcal{C}_0, \mathcal{C}_1$ through repeated challenges in the zero-knowledge proof.

In LESS, the public parameters include the modulus $q = 127$, code size (n, k) with $n = 2k$, the number of challenges t , the number of non-zero challenges w , the number of key matrices s , and the public matrix $\mathbf{G}_0 = [\mathbf{I}_k \mid \mathbf{M}_{k \times k}] \in \mathbb{Z}_q^{k \times n}$. The signer’s private key contains $s-1$ monomial matrices $\mathbf{Q}_1, \mathbf{Q}_2, \dots, \mathbf{Q}_{s-1}$, with public keys $\mathbf{G}_i = \text{RREF}(\mathbf{G}_0 \mathbf{Q}_i^{-1})$ for $i = 1, 2, \dots, s-1$. For a challenge sequence $\mathbf{b} = (b_0, b_1, \dots, b_{t-1})$, the signer constructs the response for each i from either the nonce matrix $\tilde{\mathbf{Q}}_i \in \mathbf{M}_n$ alone (if $b_i = 0$) or from both $\tilde{\mathbf{Q}}_i$ and the private matrix $\mathbf{Q}_{b_i}$ (if $b_i \neq 0$).

To reduce the key and signature sizes, LESS uses seeds to represent randomly generated values: the secret key is a seed $\text{seed}_{\text{priv}}$ of at most 64 bytes, from which all secret monomial matrices can be derived, while the public parameter $\mathbf{G}_0$ and intermediate nonce values $\tilde{\mathbf{Q}}_i$ are also seed-derived. For further signature-size optimization, nonces $\tilde{\mathbf{Q}}_i$ ($i = 0, 1, \dots, t-1$) are organized into a GGM tree [GGM84], where each node stores a unique seed. The root seed is derived from $\text{seed}_{\text{priv}}$, and each child node’s seed is deterministically generated by applying a cryptographic hash function to the corresponding parent-node seed. By the one-wayness of the hash function, the parent-node seed remains computationally infeasible to recover even if all of its child-node seeds are published. For example, in Figure 1, with $t = 8$, $w = 4$, and $\mathbf{b} = (1, 0, 0, 0, 3, 1, 0, 2)$, the seeds of four nonce matrices ($\tilde{\mathbf{Q}}_1, \tilde{\mathbf{Q}}_2, \tilde{\mathbf{Q}}_3, \tilde{\mathbf{Q}}_6$) should be published. However, with GGM tree compression, only 3 seeds need to be published, as $\tilde{\mathbf{Q}}_2$ and $\tilde{\mathbf{Q}}_3$ can be derived from a common ancestor seed and the verifier can reconstruct the tree to retrieve all required seeds.

Building on the above seed-based optimizations, Algorithm 1 outlines the signing process of LESS¹. Here, the function **RREFWithTrack** returns both the RREF of the input matrix and a permutation matrix $\mathbf{T}_i$ (the inverse permutation of $\tilde{\mathbf{Q}}_i$). The function **CosetRep** extracts the first k entries of a permutation: for example, if $n = 8$ and $k = 4$, $\text{CosetRep}([1, 6, 4, 7, 0, 5, 3, 2]) = \{1, 4, 6, 7\}$. Thus, if $b_i \neq 0$ is the u -th non-zero challenge, the signer needs to return a k -element set $\text{rsp}_u = \text{CosetRep}(\mathbf{Q}_{b_i} \mathbf{T}_i)$ with $\text{seed}^{(i)}$ unpublished. In this case, publishing $\text{seed}^{(i)}$ would leak sensitive information about $\mathbf{Q}_{b_i}$, which forms the basis of our attack.

¹Due to page limits, some preliminary steps are omitted.

Algorithm 1 LESS signing algorithm

Input: Public parameter (n, k, t, w, s) , the public key seed length λ , private key seed $\text{seed}_{\text{priv}}$ (length: 2λ) and message msg

Output: The signature structure $\text{sig} = (\text{cmt}, \text{salt}, \text{path}, \text{rep})$

- 1: Use $\text{seed}_{\text{priv}}$ to extract all the secret monomial matrices $\mathbf{Q}_1, \mathbf{Q}_2, \dots, \mathbf{Q}_{s-1}$, the root seed of the GGM tree $\text{seed}_{\text{tree}}$.
- 2: Randomly sample the salt value with length 2λ .
- 3: Use $\text{seed}_{\text{tree}}$ to extract all the seeds for the nonce matrices: $\{\text{seed}^{(0)}, \text{seed}^{(1)}, \dots, \text{seed}^{(t-1)}\}$.
- 4: **for** $i = 0$ to $t - 1$ **do**
- 5: $\tilde{\mathbf{Q}}_i \leftarrow \text{SampleMonomial}(\text{seed}^{(i)} \parallel \text{salt} \parallel i)$
- 6: $([\mathbf{I}_k | \mathbf{A}_i], \mathbf{T}_i) \leftarrow (\text{RREFwithTrack}(\mathbf{G}_0 \tilde{\mathbf{Q}}_i))$
- 7: $\mathbf{B}_i \leftarrow \text{CF}(\mathbf{A}_i)$ //The “blind” operation on $\mathbf{A}_i$ is omitted here as it has no effect on $\mathbf{B}_i$
- 8: **end for**
- 9: $\text{cmt} \leftarrow \text{Hash}(\mathbf{B}_0 \parallel \mathbf{B}_1 \parallel \dots \parallel \mathbf{B}_{t-1} \parallel \text{salt} \parallel \text{msg})$
- 10: $\mathbf{b} = (b_0, b_1, \dots, b_{t-1}) \leftarrow \text{SampleChallenge}(\text{cmt})$
- 11: Create array $\mathbf{f} = (f_0, f_1, \dots, f_{t-1})$ where $f_i \leftarrow 0$ if $b_i = 0$ else 1.
- 12: $\text{path} \leftarrow \text{GGMPath}(\mathbf{f})$ // store the seed to be published into array path
- 13: $j \leftarrow 0$
- 14: **for** $i = 0$ to $t - 1$ **do**
- 15: **if** $b_i \neq 0$ **then**
- 16: $\text{rsp}_j \leftarrow \text{CosetRep}(\mathbf{Q}_{b_i} \mathbf{T}_i)$ // rsp_j is a k -element set.
- 17: $j \leftarrow j + 1$
- 18: **end if**
- 19: **end for**
- 20: $\text{rsp} \leftarrow (\text{rsp}_0, \text{rsp}_1, \dots, \text{rsp}_{w-1})$ // rsp is an array containing w k -element sets .
- 21: **return** $\text{sig} \leftarrow (\text{cmt}, \text{salt}, \text{path}, \text{rsp})$.

3 Attack Point and Fault Models

3.1 Technical Overview

In the signing algorithm of LESS, we observed that between sampling array $\mathbf{b}$ and executing GGMPath , there is an intermediate step in which array $\mathbf{b}$ is transformed into array $\mathbf{f}$. Crucially, the implementation of LESS relies on array $\mathbf{f}$ rather than $\mathbf{b}$ to determine which seeds are published. As can be seen from the official C implementation of LESS (Listing 1), array $\mathbf{f}$ is computed from array $\mathbf{b}$ through a simple **for** loop.

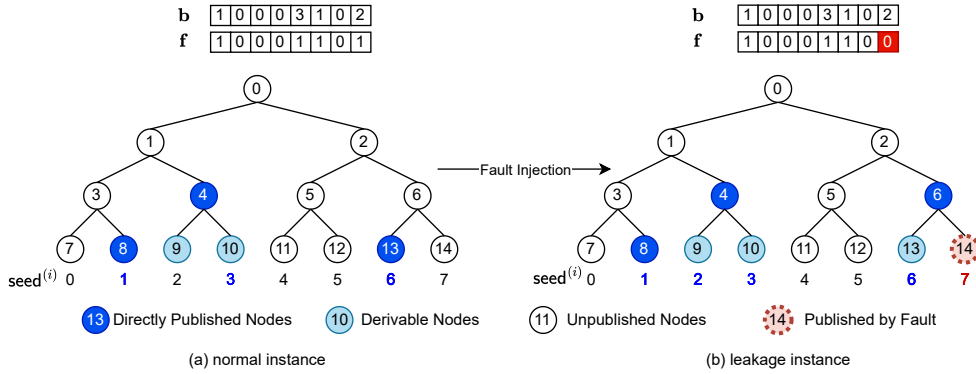


Figure 1: GGM tree structure and illustration of our fault injection

```

1  uint8_t fixed_weight_string[T]={0}; //Array b
2  SampleChallenge(fixed_weight_string, sig->digest); //b=SampleChallenge(cmt)
3
4  uint8_t indices_to_publish[T]; //Array f
5  for (uint32_t i = 0; i < T; i++) {
6      indices_to_publish[i] = !(fixed_weight_string[i]);
7  }
8  // build GGM tree according to the array indices_to_publish

```

Listing 1: LESS.c implementation code snippet (lines 227-234)

Therefore, if the `for` loop in Listing 1 is perturbed, some iterations may fail to execute, leaving specific elements of array `f` unmodified. As an uninitialized temporary array, the contents of array `f` should, strictly according to the C standard, be indeterminate. However, modern operating systems often incorporate memory reclamation and page-zeroing strategies [YBF⁺11, AMH⁺16, LKS⁺17, SS18]. As a result, each element of array `f` has a high probability of being set to zero, and sometimes the whole array `f` is zero-filled.

Meanwhile, we noticed all public parameters like n, t, w, s are defined as macros and determined during compilation, and all arrays defined in the signing function are fixed-length arrays. This deterministic execution and memory occupation mode results in a consistent stack frame layout of the signing function throughout the entire program runtime, and the memory allocation addresses of its internal temporary arrays exhibit high repeatability on some systems. Consequently, array `f` is prone to retaining residual values from its last invocation of the signing function, thereby creating a potential attack approach for adversaries.

Building on this analysis, we compiled the official C implementation of LESS for all parameter sets across different compilers, devices, and operating systems. Our experiments confirm that the initial values of array `f` match our expectation most of the time, indicating that the array's state can be inferred under specific conditions. Therefore, we can perturb the `for` loop through fault injection to trigger some $b_{i^*} \neq 0$ with corresponding $f_{i^*} = 0$. With knowledge of the faulty `f`, we can reconstruct the GGM tree to obtain the seed for the nonce matrix $\tilde{\mathbf{Q}}_{i^*}$, thereby gaining partial information about the secret monomial matrix $\mathbf{Q}_{b_{i^*}}$ and eventually recovering it (see Section 4).

For instance, as conceptually illustrated in the right subfigure of Figure 1, a fault is injected at index $i = 7$, causing $f_7 = 0$ while $b_7 = 2 \neq 0$. This triggers the leakage of the nonce matrix $\tilde{\mathbf{Q}}_7$. By computing $\mathbf{T}_7$ from $\tilde{\mathbf{Q}}_7$, partial information about the secret monomial matrix $\mathbf{Q}_{b_7} = \mathbf{Q}_2$ can be obtained, and the adversary can repeat the attack to gather more information until fully recovering $\mathbf{Q}_2$ or even other secret $\mathbf{Q}$ values. Since a successful fault injection attack can produce multiple cases with $b_i \neq 0$ and corresponding $f_i = 0$, we give the following two definitions:

Definition 6 (Leakage). In the signing process, if a fault occurs such that $b_i \neq 0$ but $f_i = 0$, we call it a **leakage**. A leakage means that a seed that should not be published is published because of the fault, thereby leaking information about the corresponding secret monomial matrix. Specifically, if $b_i = s^* \neq 0$, we call it a leakage related to the secret monomial matrix $\mathbf{Q}_{s^*}$.

Definition 7 (Signing Oracle Access). For an adversary performing the attack, we define a **signing oracle access** (or simply an access) as one instance in which the adversary obtains the output of the signing oracle during the attack process. Therefore, a single signing oracle access may contain one or more leakages, or no leakage because the attack failed or no attack was performed, but it always includes a potentially faulty signature returned by the signing oracle.

3.2 Fault Models

Before introducing our fault models, we first state the assumptions used in our fault analysis, which align with realistic adversarial capabilities in practical cryptanalytic scenarios:

- For each signing request, the signing oracle uses the fixed private key but employs a random salt value to sign the given message.
- The adversary has full access to the signing oracle, possesses complete public knowledge of the LESS scheme parameters, and can perform fault attacks on it. The adversary can also obtain a copy of the signing-oracle environment without the secret key installed for debugging. However, the adversary does not know the private key or any secret internal values of the oracle.

We also formalize the two common cases of $\mathbf{f}$ mentioned in Section 3.1 to standardize the subsequent notation.

Case 1: Zero-Filled Array. In this case, array $\mathbf{f}$ is fully initialized with zero values automatically by the operating system. This behavior stems from memory zeroing at the operating-system level or from compiler-enforced security mechanisms designed to prevent residual data leakage. In our tests, when the signing function is first invoked, this case occurs across mainstream environments, including Raspberry Pi OS and Kali Linux 24Q4 with the g++ compiler, as well as Ubuntu 24.04, 25.04, and 25.10 with the clang compiler.

This case is also consistent with what we observed on embedded devices. In such systems, after program execution the relevant stack space often does not retain random residual data; instead, it tends to appear in a deterministic all-zero state. This behavior makes the attack particularly realistic, because the adversary can directly exploit the semantics of the value 0 in the array to infer which information will be published. As a result, the zero-filled case provides a representative attack scenario for embedded implementations.

Case 2: Statically-Addressed Array. In this case, array $\mathbf{f}$ retains residual values from prior invocations of the signature function, because its memory address in the stack frame of the signing procedure remains fixed across repeated calls. The initial state of $\mathbf{f}$ need not be mostly or entirely zero; instead, it is determined by the last values written to its fixed memory region. In our tests, this case can sometimes be observed on Windows 10 19H2 with the MinGW g++ compiler.

This model is particularly relevant to servers and other high-frequency execution environments. In such settings, the signing algorithm may be invoked repeatedly, while the corresponding memory region is not fully cleared between calls. Consequently, the array may retain historical random-looking values generated during earlier executions. An attacker can then exploit these residual values directly, without relying on a specific initialization pattern. Together, the zero-filled and statically-addressed cases capture typical behaviors of embedded and high-performance systems respectively, showing that our attack model applies across a broad range of realistic deployment scenarios.

In this paper, we consider the following two fault models:

Fault Model 1: Single Iteration Skip. In this model, the adversary performs the attack to skip a specific iteration (e.g., the i^* -th iteration) in the `for` loop illustrated in Listing 1, and leave other iterations executed normally. This fault blocks the assignment $f_{i^*} \leftarrow \text{!!}(b_{i^*})$ from occurring, leaving f_{i^*} in its initial state. If $b_{i^*} \neq 0$ and the initial value of f_{i^*} is 0, then the adversary obtains a leakage related to $\mathbf{Q}_{b_{i^*}}$. Since the fault only affects one element in $\mathbf{f}$, at most one leakage can be obtained per single access.

Fault Model 2: Loop Abort. In this model, the **for** loop terminates **after** the i^* -th iteration, with previous iterations executed normally. It is equivalent to inserting a **break** statement at the end of the i^* -th iteration or bypassing the loop's jump-back instruction at the assembly level. This fault leaves all elements $f_{i^*+1}, f_{i^*+2}, \dots, f_{t-1}$ unmodified. Compared with the “Single Iteration Skip” model, more than one leakage may occur in a single access to the signing oracle.

4 Fault Injection Attack on LESS

In this section, we first provide our two-phase key recovery algorithm under the assumption of the physical attack being successfully performed. Then, we introduce the details of our physical attack. At last, we elaborate our filtering method to extract effective leakage data from practical fault injection outputs.

4.1 Key Recovery

Our key-recovery process consists of two steps: permutation recovery and coefficient recovery. For convenience, in this subsection we assume that all attack trials are successful, and that array $\mathbf{f}$ can be obtained to rebuild the GGM tree and retrieve the seeds for the target nonce matrices.

4.1.1 Step 1: Permutation Recovery

To recover the permutation of the s^* -th private monomial matrix $\mathbf{Q}_{s^*}$, we first extract the challenge array $\mathbf{b}$ from the `cmt` value in the signature. Then we fill the fault-injected array $\mathbf{f}$ according to the target index and the fault model to rebuild the GGM tree and extract the leaf seeds. Through this procedure, we can store all leakage information related to $\mathbf{Q}_{s^*}$. By exploiting the leaked nonce $\tilde{\mathbf{Q}}_i$, the permutation matrix $\mathbf{T}$ and its corresponding permutation $\mathbf{p} = (p_0, p_1, \dots, p_{n-1})$ inside the signing algorithm can be obtained. We then form a set V containing the first k values of permutation $\mathbf{p}$ and collect its corresponding set R from the response array `rsp` in the faulty signature. Such a pair (V, R) leaks information about the permutation of $\mathbf{Q}_{s^*}$ because $R = \mathbf{p}_V = \{p_v \mid v \in V\}$. The detailed extraction process is described in Algorithm 2.

Algorithm 2 Extract the leakage information related to $\mathbf{Q}_{s^*}$ from the faulty signatures

Input: Public parameter (n, k, t, w, s) with $n = 2k$, target key id s^* , faulty signature $\mathbf{fsig} = \{\text{cmt}, \text{salt}, \text{path}, \text{rsp}\}$, and faulty indexes $L = \{\ell_0, \ell_1, \dots, \ell_{m-1}\}$

Output: Permutation set V and corresponding response $R = \mathbf{fsig.rsp}_u$ associated with $\mathbf{Q}_{s^*}$

```

1:  $\mathbf{b} = (b_0, b_1, \dots, b_{t-1}) \leftarrow \text{SampleChallenge}(\mathbf{fsig.cmt})$ 
2: Fill the array  $\mathbf{f}$  based on the faulty position and the predefined error model.
3:  $\text{GGMTree} \leftarrow \text{RebuildGGMTree}(\mathbf{fsig.path}, \mathbf{f})$ 
4:  $\{\text{seed}_{\text{leaf}}^{(i)}\} \leftarrow \text{TreeLeaves}(\text{GGMTree})$ 
5: for  $u = 0$  to  $t - 1$  do
6:   if  $(b_u = s^*) \wedge (\mathbf{f}_u = 0)$  then
7:      $([\mathbf{I}_m \mid \mathbf{A}], \mathbf{T}) \leftarrow \text{RREFWithTrack}(\mathbf{G}_0 \tilde{\mathbf{Q}}_{\ell_u})$ 
8:     Convert the permutation matrix  $\mathbf{T}$  to the permutation sequence  $\mathbf{p} = (p_0, p_1, \dots, p_{n-1})$ .
9:      $V \leftarrow \{p_0, p_1, \dots, p_{k-1}\}$ 
10:     $z \leftarrow$  the number of non-zero values in  $\{b_0, b_1, \dots, b_{u-1}\}$ 
11:    Save the set  $V$  and  $R = \mathbf{fsig.rsp}_z$  to the output file as the input of Algorithm 3.
12:   end if
13: end for
```

After collecting enough (V, R) pairs, we can apply our set-intersection method to recover the permutation of the secret monomial matrix $\mathbf{Q}_{s^*}$, as defined precisely in Algorithm 3.

Algorithm 3 Recover permutation of secret key $\mathbf{Q}_{s^*}$ with set intersection method

Input: Public parameters (n, k, t, w, s) , number of collected samples m , the permutations $\{V_0, V_1, \dots, V_{m-1}\}$, and their corresponding responses $\{R_0, R_1, R_2, \dots, R_{m-1}\}$

Output: Permutation matrix $\mathbf{P}$ or $\perp$

```

1: Initialize  $\text{Candidate}_i \leftarrow \mathbb{Z}_n$  for  $i = 0, \dots, n-1$  //  $\text{Candidate}_i$ : set,  $\text{Candidate}$ : set array.
2: for  $u = 0$  to  $m-1$  do
3:   for  $i = 0$  to  $n-1$  do
4:     if  $i \in R_u$  then
5:        $\text{Candidate}_i \leftarrow \text{Candidate}_i \cap V_u$ 
6:     else
7:        $\text{Candidate}_i \leftarrow \text{Candidate}_i \cap V_u^c$ 
8:     end if
9:     if  $\text{Candidate}_i = \emptyset$  then
10:      return  $\perp$ 
11:    end if
12:  end for
13:  if All  $|\text{Candidate}_i| = 1$  and  $(\bigcup_{i=0}^{n-1} \text{Candidate}_i) = \mathbb{Z}_n$  then
14:    Record the unique element in each  $\text{Candidate}_i$  in order as permutation sequence  $\mathbf{p}'$ .
15:    return The permutation matrix  $\mathbf{P}$  corresponding to the sequence  $\mathbf{p}'$ .
16:  end if
17: end for
18: return  $\perp$ 

```

However, Algorithm 3 is notably vulnerable to irrelevant data from failed attacks, since any (V, R) pair not related to $\mathbf{Q}_{s^*}$ can cause it to return $\perp$. Therefore, it is critical to filter the relevant (V, R) pairs from all collected data before recovering $\mathbf{Q}_{s^*}$. We describe this filtering algorithm in Section 4.4.

4.1.2 Step 2: Coefficients Recovery

With Algorithms 2 and 3, we can recover the permutation of a specific private monomial matrix $\mathbf{Q}_{s^*}$. By iterating s^* from 1 to $s-1$, the permutations of all private monomial matrices can be recovered. However, recovering the actual secret matrices also requires coefficient recovery. Without loss of generality, we again use the s^* -th private monomial matrix as an example.

First, according to Definition 1, the monomial matrix $\mathbf{Q}_{s^*}$ has the form $\mathbf{Q}_{s^*} = \mathbf{D}\mathbf{P}$. Then we have

$$\mathbf{G}_{s^*}^{\text{raw}} = \mathbf{G}_0 \mathbf{Q}_{s^*}^{-1} = \mathbf{G}_0 \mathbf{P}^{-1} \mathbf{D}^{-1} \quad (1)$$

Since $\mathbf{G}_0$ and $\mathbf{P}$ are known, while $\mathbf{Q}_{s^*}$ and $\mathbf{D}$ remain unknown, we can compute an auxiliary value $\mathbf{H}^{\text{raw}}$ by treating $\mathbf{G}_0 \mathbf{P}^{-1}$ as a single entity:

$$\mathbf{H}^{\text{raw}} = \mathbf{G}_0 \mathbf{P}^{-1} \quad (2)$$

As $n = 2k$, the matrix $\mathbf{H}^{\text{raw}}$ can be regarded as the column-wise concatenation of two square matrices $\mathbf{B}, \mathbf{C} \in \mathbb{Z}_q^{k \times k}$, i.e., $\mathbf{H}^{\text{raw}}$ admits the form:

$$\mathbf{H}^{\text{raw}} = [\mathbf{B} | \mathbf{C}] \quad (3)$$

Therefore, performing RREF on $\mathbf{H}^{\text{raw}}$ is essentially equivalent to computing

$$\mathbf{H} = \text{RREF}(\mathbf{H}^{\text{raw}}) = \text{RREF}([\mathbf{B} | \mathbf{C}]) = \mathbf{B}^{-1}[\mathbf{B} | \mathbf{C}] = [\mathbf{I} | \mathbf{B}^{-1}\mathbf{C}] \quad (4)$$

Substituting (3) into (1), we obtain:

$$\mathbf{G}_{s^*}^{\text{raw}} = [\mathbf{B} | \mathbf{C}] \mathbf{D}^{-1} \quad (5)$$

Given that $\mathbf{D}$ is an $n \times n$ diagonal matrix, it can be partitioned into a block diagonal structure comprising $\mathbf{D}_1$ and $\mathbf{D}_2$, both of which are $k \times k$ diagonal matrices, i.e.:

$$\mathbf{D} = \begin{bmatrix} \mathbf{D}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_2 \end{bmatrix} \quad (6)$$

Substituting (6) into (5), we obtain:

$$\mathbf{G}_{s^*}^{\text{raw}} = [\mathbf{B}|\mathbf{C}] \begin{bmatrix} \mathbf{D}_1^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_2^{-1} \end{bmatrix} = [\mathbf{B}\mathbf{D}_1^{-1}|\mathbf{C}\mathbf{D}_2^{-1}] \quad (7)$$

Therefore, it can be derived that

$$\mathbf{G}_{s^*} = \text{RREF}(\mathbf{G}_{s^*}^{\text{raw}}) = (\mathbf{B}\mathbf{D}_1^{-1})^{-1}[\mathbf{B}\mathbf{D}_1^{-1}|\mathbf{C}\mathbf{D}_2^{-1}] = [\mathbf{I}|\mathbf{D}_1\mathbf{B}^{-1}\mathbf{C}\mathbf{D}_2^{-1}] \quad (8)$$

Let the right halves of $\mathbf{H}$ and $\mathbf{G}_{s^*}$ be $\mathbf{H}_R$ and $\mathbf{G}_R$ respectively, then we have:

$$\mathbf{G}_R = \mathbf{D}_1\mathbf{H}_R\mathbf{D}_2^{-1} \quad (9)$$

Therefore, for $\mathbf{H}_R = [h_{ij}]$, $\mathbf{G}_R = [g_{ij}]$, $\mathbf{D}_1 = \text{diag}(d_{(1,0)}, d_{(1,1)}, \dots, d_{(1,k-1)})$ and $\mathbf{D}_2 = \text{diag}(d_{(2,0)}, d_{(2,1)}, \dots, d_{(2,k-1)})$, we have:

$$g_{ij} = \frac{d_{(1,i)}h_{ij}}{d_{(2,j)}} \quad (10)$$

Herein, we define the matrix $\mathbf{U} = [u_{ij}] \in \mathbb{Z}_q^{k \times k}$ with $u_{ij} = \frac{g_{ij}}{h_{ij}}$. Specifically, if $h_{ij} = 0$, we define $u_{ij} = 0$. Theoretically, in the absence of such singular cases, $\mathbf{U}$ is expected to be a rank-1 matrix with no zero elements. Nevertheless, since both $\mathbf{G}_R$ and $\mathbf{H}_R$ are defined over the modular field $\mathbb{Z}_q$, some $g_{ij} = 0$ or $h_{ij} = 0$ disrupted the inherent rank-1 structure of $\mathbf{U}$. Notably, even in the presence of such zeros, the ratio between the non-zero elements in the corresponding rows or columns of $\mathbf{U}$ remains invariant. Therefore, we can still attempt to recover the proportional relationships between any two rows and between any two columns of $\mathbf{U}$, with all zero entries omitted in the calculation.

For each set of row vectors $(\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_{k-1})$ and column vectors $(\mathbf{u}'_0, \mathbf{u}'_1, \dots, \mathbf{u}'_{k-1})$ of $\mathbf{U}$, we formally define the zero-omitted ratios between any two rows or two columns using the notational forms $\frac{\mathbf{u}_i}{\mathbf{u}_j}$ and $\frac{\mathbf{u}'_i}{\mathbf{u}'_j}$ respectively. Then, we can compute:

$$\tilde{\mathbf{d}}_1 = (\tilde{d}_{(1,0)}, \tilde{d}_{(1,1)}, \dots, \tilde{d}_{(1,k-1)}) = \left(\frac{\mathbf{u}'_0}{\mathbf{u}'_0}, \frac{\mathbf{u}'_1}{\mathbf{u}'_0}, \dots, \frac{\mathbf{u}'_{k-1}}{\mathbf{u}'_0} \right) \quad (11)$$

and

$$\tilde{\mathbf{d}}_2 = (\tilde{d}_{(2,0)}, \tilde{d}_{(2,1)}, \dots, \tilde{d}_{(2,k-1)}) = \left(\frac{\mathbf{u}_0}{\mathbf{u}_0}, \frac{\mathbf{u}_1}{\mathbf{u}_0}, \dots, \frac{\mathbf{u}_{k-1}}{\mathbf{u}_0} \right) \quad (12)$$

After obtaining $\tilde{\mathbf{d}}_1$ and $\tilde{\mathbf{d}}_2$, we can derive the coefficient matrix $\mathbf{D}_{s^*} = \text{diag}(d_0, d_1, \dots, d_{n-1})$ ($0 \leq i \leq k-1$) through the following relationship:

$$\tilde{d}_{(2,i)} = \frac{d_i}{d_0}, \quad \tilde{d}_{(1,i)} = \frac{d_k}{d_{k+i}} \quad (13)$$

Therefore, the solution space can be determined by fixing $d_0 = 1$, enumerating all possible values of d_k , and generating the corresponding candidate matrices $\mathbf{D}'$ and private-key candidates $\mathbf{Q}'_{s^*}$. We then validate each candidate by computing $\text{RREF}(\mathbf{G}_0(\mathbf{Q}'_{s^*})^{-1})$ and comparing the result to $\mathbf{G}_{s^*}$. If $\mathbf{G}_{s^*} = \text{RREF}(\mathbf{G}_0(\mathbf{Q}'_{s^*})^{-1})$ holds, then $\mathbf{Q}'_{s^*}$ is a valid solution equivalent to the secret matrix in the actual private key. This equivalence stems from the scale invariance of the RREF operation: for any non-zero scalar $a \in \mathbb{Z}_q$, $\text{RREF}(\mathbf{G}) = \text{RREF}(a\mathbf{G})$, and $\mathbf{Q}'_{s^*}$ differs from the real $\mathbf{Q}_{s^*}$ only by a constant scalar factor.

4.2 Clock Glitch Attack Implementation

The clock glitch attack is a fault injection technique that induces momentary perturbations in the target device's clock signal. These perturbations cause transient timing violations, which can result in instruction skipping or disturbance in program execution.

We implemented the clock glitch attack using the ChipWhisperer toolkit [OC14], with the ChipWhisperer Lite as the capture scope and the CW308 UFO board equipped with an STM32F303 microcontroller as the target device.

According to the ChipWhisperer documentation, the attack parameters used in the experiment are defined by the triple $(\text{ext_offset}, \text{width}, \text{offset})$:

- **ext_offset**: A positive integer representing the number of clock cycles between the execution of the `trigger_high` flag and the initiation of glitch injection.
- **width**: A real number in $[0, 50]$, denoting the glitch pulse width as a percentage of the full clock cycle duration.
- **offset**: A real number in $[-48.8, +48.8]$, specifying the percentage adjustment of the glitch trigger position within a single clock cycle. The cumulative glitch trigger point is calculated as $\text{ext_offset} + 0.01 \times \text{offset}$ clock cycles after `trigger_high`.

4.3 Finding the Proper Attack Parameter

To conduct an effective fault-attack experiment on the ChipWhisperer platform, we integrated the `trigger_high()` and `trigger_low()` functions before and after the target loop. Furthermore, to verify whether the configured attack parameters achieve the expected results, we inserted an extra loop to recompute array **f** and validate the effect. This supplementary check has no impact on the attack parameters and will be removed in the formal experiments.

By testing all LESS parameter sets, we found that each loop iteration takes around 20 cycles on our device. Therefore, we set `ext_offset` in $[80, 180]$, `width` in $[6, 25]$, and `offset` in $[-12, 12]$, and recorded the number of successful `offset` values under different $(\text{ext_offset}, \text{width})$ pairs. Some representative results are shown in Figure 2.

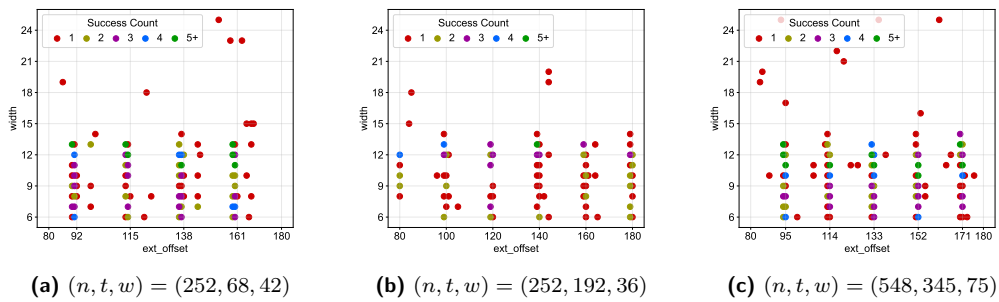


Figure 2: Success attack count with different `ext_offset` and `width`

From Figure 2, we observe that successful attacks cluster in the range $6 \leq \text{width} \leq 13$. Given that a single iteration consumes c clock cycles, and we identified a set of attack parameter triples $(\text{ext_offset}, \text{width}, \text{offset})$ that achieve a high success rate when targeting the i^* -th iteration, the modified triple $(\text{ext_offset} + c, \text{width}, \text{offset})$ will likewise yield a high attack success rate for the $(i^* + 1)$ -th iteration. As the appropriate `ext_offset` value can be determined here, we can fix it and enumerate the $(\text{width}, \text{offset})$ pair more precisely in the test to find the (approximate) optimal parameters to perform our attack.

It is critical to note that in the “Single Iteration Skip” model, the ideal attack parameter should maximize the conditional probability $\Pr[\text{Skip the iteration} \mid b_{i^*} \neq 0]$ rather than the marginal probability $\Pr[\text{Skip the iteration}]$. This is because skipping the iteration at index i^* when $b_{i^*} = 0$ is operationally meaningless. However, the “Loop Abort” model is immune to this effect, as all subsequent iterations will not be executed after the i^* -th iteration.

4.4 Identifying Effective Leakage Information

In practical fault injection attacks, the fault-injection process is not always successful, which produces irrelevant data that interferes with the output of Algorithm 3. Therefore, identifying valid faulty signatures from large collections of samples is also an important challenge for effective key recovery.

In Algorithm 3, for any two leakage tuples (V_0, R_0) and (V_1, R_1) extracted by Algorithm 2, if both are relevant to the permutation of the same secret matrix $\mathbf{Q}_{s^*}$, then for any set Candidate_i , the following must hold (where the universal set is $\mathbb{Z}_n$):

$$\text{Candidate}_i \in \{V_0 \cap V_1, V_0^c \cap V_1, V_0 \cap V_1^c, V_0^c \cap V_1^c\} \quad (14)$$

From (14), for all sets in the Candidate array and for any distinct integers $i, j \in \mathbb{Z}_n$, either $\text{Candidate}_i = \text{Candidate}_j$ or $\text{Candidate}_i \cap \text{Candidate}_j = \emptyset$. By leveraging this property, we propose Algorithm 4 to validate the collected information and discard invalid data while retaining the majority of related (V, R) pairs. This algorithm is based on disjoint set union and has time complexity $O(m^2n)$. Here, m denotes the number of collected (V, R) tuples.

For the scheme parameters with $s > 2$, if we only take the largest set in the DSU, we may require an excessive amount of leakage information to recover all the secret monomial matrices, leading to the loss of numerous useful leakages. Therefore, a threshold value K (preferably between 3 and 5) can be introduced in Algorithm 4, which returns all sets in DSU whose size exceeds K and clusters them by the number of their occurrences. This modification can effectively reduce the leakage loss and enable a more efficient recovery process for the permutation of these secret matrices.

5 Attack Results

5.1 Attack Result in Single Iteration Skip Model

For the “Single Iteration Skip” model with zero-filled $\mathbf{f}$, we performed the attack at $i = 4$ on ChipWhisperer, and the results are presented in the first seven rows of Table 1. According to the results, for all seven parameter sets, the average number of signing-oracle accesses required in our attacks is consistently below 300.

To evaluate the successful skip rate across different target indices during the attack, we carried out experiments at distinct positions within the target loop. Beyond the baseline target index $i = 4$, we additionally selected four supplementary target indices for each parameter set, positioned at $0.25t$, $0.5t$, $0.75t$, and near the termination of the **for** loop. The experimental results are summarized in Table 2. In the experiment, we observed that some specific parameters yielded a higher successful skip rate when we changed the target index, and attack at these indices can effectively reduced the number of signing-oracle accesses. The corresponding data are detailed in the 8th to the 10th rows of Table 1.

The fluctuation in the successful skip rate can be understood in two aspects. The first aspect is related to our strategy for selecting attack parameters: In the “Single Iteration Skip” model, the most efficient attack parameter is the one that maximizes $\Pr[\text{Successful Skip} \mid b_i \neq 0]$. After determining a satisfactory attack parameter, we

Algorithm 4 Extracting valid leakage from collected data in Algorithm 2

Input: Public parameters (n, k, t, w, s) , m collections $\{(V_0, R_0), (V_1, R_1), \dots, (V_{m-1}, R_{m-1})\}$ from Algorithms 2 related to $\mathbf{Q}_{s^*}$

Output: The filtered information (V, R) as the input of Algorithm 3.

```

1: DSU  $\leftarrow$  MakeDSU( $m$ )
2: for  $i = 0$  to  $m - 1$  do
3:   if DSU.Find( $i$ )  $\neq i$  then
4:     continue
5:   end if
6:   Initialize  $\text{Candidate}_j \leftarrow \mathbb{Z}_n$  for  $j = 0$  to  $n - 1$ .
7:   for  $j = i + 1$  to  $m - 1$  do
8:     if DSU.Find( $j$ )  $\neq j$  then
9:       continue
10:    end if
11:    for  $t$  in  $\{i, j\}$  do
12:      for  $u = 0$  to  $n - 1$  do
13:        if  $u \in R_t$  then
14:           $\text{Candidate}_u \leftarrow \text{Candidate}_u \cap V_t$ 
15:        else
16:           $\text{Candidate}_u \leftarrow \text{Candidate}_u \cap V_t^c$ 
17:        end if
18:      end for
19:    end for
20:     $\text{CandidateS} \leftarrow \{S \mid S \in \text{Candidate}\}$  //CandidateS has four elements.
21:    Check consistency: each  $\text{CandidateS}_u$  has exactly  $|\text{CandidateS}_u|$  copies in  $\text{Candidate}$ 
22:    Check disjointness: all  $\text{CandidateS}_u \cap \text{CandidateS}_v = \emptyset$  for  $u, v \in \mathbb{Z}_4, u \neq v$ 
23:    if checks pass then
24:      DSU.Union( $i, j$ )
25:    end if
26:  end for
27: end for
28: Take the largest subset  $S_{\max}$  from DSU.
29: return  $\{(V_i, R_i) \mid i \in S_{\max}\}$  as the input of Algorithm 3

```

evaluate the overall successful skip rate under the real scheme parameters, and record the total successful skip rate as

$$\Pr[\text{successful skip}] = \Pr[\text{successful skip} \mid b_i \neq 0] + \Pr[\text{successful skip} \mid b_i = 0].$$

However, we only maximize the conditional probability corresponding to $b_i \neq 0$ as skipping a $b_i = 0$ has no utility in our attack. For most scheme parameters except $(548, 345, 75, 2)$, the glitch parameters required to skip $b_i \neq 0$ and $b_i = 0$ at index i differ, so under our optimal parameter, we almost exclusively skip instances where $b_i \neq 0$. Consequently, the corresponding average total successful skip rate is mostly close to w/t . However, for the scheme parameter $(548, 345, 75, 2)$, the glitch parameters required to skip zero and nonzero values at b_i coincide in most cases, which enables us to achieve a 100% successful skip rate under this parameter set.

Second, and more surprisingly, we incidentally discovered during experiments an implementation bug in the `SampleChallenge` function when $s > 2^2$. This bug causes its output to be non-uniform. We tested and analyzed this function, which is designed to sample the integer array $\mathbf{b}$ of length t with exactly w nonzero elements in the range $[1, s - 1]$. However, our tests reveal that the probability of nonzero entries varies noticeably

²See line 144 at https://github.com/less-sig/LESS/blob/c57593900d4fd68f643d053b24f229e9dea43aea/Optimized_Implementation/common/lib/utils.c

Table 1: Attack result in “Single Iteration Skip” model (zero-filled $\mathbf{f}$)

(n, t, w, s)	Skip index	Skip rate	Trials	Min	Max	Average
(252, 192, 36, 2)	4	18.32%	32	44	165	83.59
(252, 68, 42, 4)	4	47.94%	32	95	261	156.81
(252, 45, 34, 8)	4	71.58%	32	167	307	223.94
(400, 220, 68, 2)	4	31.51%	32	32	105	60.53
(400, 102, 61, 4)	4	51.03%	32	94	170	131.88
(548, 345, 75, 2)	4	100.00%	32	48	132	85.13
(548, 137, 79, 4)	4	46.84%	32	102	210	144.09
(252, 68, 42, 4)	32	70.50%	32	79	199	121.34
(400, 102, 61, 4)	50	90.28%	32	54	103	74.16
(548, 137, 79, 4)	68	95.29%	32	62	181	82.34

Table 2: Successful skip rate with skipping at different indices

(n, t, w, s)	Index 1	Rate 1	Index 2	Rate 2	Index 3	Rate 3	Index 4	Rate 4
(252, 192, 36, 2)	50	22.2%	96	23.4%	142	20.2%	188	20.6%
(252, 68, 42, 4)	18	45.6%	32	70.5% $\uparrow$	46	55.1%	60	55.1%
(252, 45, 34, 8)	13	79.7%	22	75.1%	31	73.4%	40	75.5%
(400, 220, 68, 2)	57	29.8%	110	29.2%	163	31.1%	216	30.3%
(400, 102, 61, 4)	27	48.4%	50	90.2% $\uparrow$	73	63.6%	96	58.4%
(548, 345, 75, 2)	88	100.0%	172	100.0%	256	20.0% $\downarrow$	340	100.0%
(548, 137, 79, 4)	36	45.3%	68	95.3% $\uparrow$	100	57.9%	132	56.8%

across different positions in array $\mathbf{b}$ for certain parameter sets (Figure 3), which causes the successful skip rate fluctuations presented in Table 2.

The **SampleChallenge** function executes two sequential phases: random number sampling (phase 1) and array shuffling (phase 2). To enable efficient random number generation, the function employs a 64-bit buffer z for temporary storage of random bits and a counter c to monitor the remaining usable chunks. The chunk size is configured as $m_1 = \lceil \log_2 s \rceil \in [2, 3]$ in phase 1, and updated to $m_2 = \lceil \log_2 t \rceil \in [7, 10]$ in phase 2. For each phase $i \in \{1, 2\}$, c is initialized to $\lfloor 64/m_i \rfloor$, representing the maximum number of m_i -bit chunks storable in the 64-bit buffer. Each sampling operation decrements c by 1 and extracts the m_i least significant bits (LSBs) from z . When c reaches zero, it will be reset to $\lfloor 64/m_i \rfloor$ with z refreshed with new random bits.

The critical bug occurs when switching from phase 1 to phase 2: the counter c is not reset to match the new chunk size m_2 , but instead retains its residual value from phase 1, which corresponds to the smaller chunk size m_1 . This creates a critical mismatch: during phase 2 sampling, m_2 bits are consumed per operation, yet the counter c still holds the larger residual value computed for m_1 . Consequently, the buffer z , which should be refreshed after consuming cm_1 bits, is only refreshed after consuming cm_2 bits. This causes z to be exhausted in advance, resulting in a stream of consecutive zeros before the refresh process, which distorts the probability distribution and directly leads to the non-uniformity of $\Pr[b_i \neq 0]$ across different indices.

For the “Single Iteration Skip” model, if array $\mathbf{f}$ is statically addressed with no initial values guaranteed, the attack algorithm can be performed as follows: We first access the signing oracle to obtain a normal signature and record the indices where $b_i = 0$ (corresponding to $f_i = 0$). Subsequently, we immediately perform an adaptive fault injection attack targeting one of these recorded indices. If the successful skip rate in the attack maintains, the required number of signing-oracle accesses will be twice that of the “zero-filled” case.

Table 3: Attack result in “Loop Abort” model (zero-filled $\mathbf{f}$)

(n, t, w, s)	Index	Abort Rate	Trials	Min	Max	Average
(252, 192, 36, 2)	20	100.00%	400	1	3	1.03
(252, 68, 42, 4)	4	100.00%	228	2	15	5.50
(252, 45, 34, 8)	4	100.00%	181	5	24	10.76
(400, 220, 68, 2)	20	100.00%	266	1	1	1.00
(400, 102, 61, 4)	4	100.00%	216	1	3	1.80
(548, 345, 75, 2)	20	100.00%	209	1	2	1.01
(548, 137, 79, 4)	20	100.00%	231	1	3	1.55

5.2 Attack and Simulation Result in Loop Abort Model

5.2.1 Attack Result in Zero-Filled Array Case

In the “Loop Abort” model, we observed that the root of the GGM tree is forced not to be published in LESS, so we must avoid letting $\mathbf{f}$ contain all zeros, since no seeds would be published in that case. Therefore, we conducted the experiment by aborting the loop at $i = 20$ if the challenge is sparse; otherwise, we aborted it at $i = 4$. The experimental results are shown in Table 3.

From the attack results, we observe that the number of signing-oracle accesses is significantly reduced compared with the “Single Iteration Skip” model, as more than one leakage can be obtained through a single signing-oracle access. For most scheme parameters, only two successful attacks is sufficient to recover all the secret matrices, and even for the most costly parameter set, the average number of successful attacks remains at approximately 10.

Similar to the “Single Iteration Skip” model, we also observed the fluctuation on the successful loop-abort rate here. This fluctuation is caused by the glitch parameters we use. In our early experiments, the large search space with time-consuming search process made us fail to identify the optimal glitch parameters for certain scheme configurations, resulting in relatively low successful abort rates under those scheme parameters. However, through extensive experimental exploration, we determined more effective glitch parameters and re-conducted the attacks with these improved configurations, thus achieving significantly enhanced attack performance.

5.2.2 Simulation Result in Statically-Addressed Array Case

In this case, we performed simulations on Ubuntu 25.10 and set the successful abort rate to 100%. Notably, we did not account for information loss during the extraction, collection, and filtering processes. Leveraging the static property of array $\mathbf{f}$, we adopted the strategy of executing one attack after each normal execution. Although a single normal execution

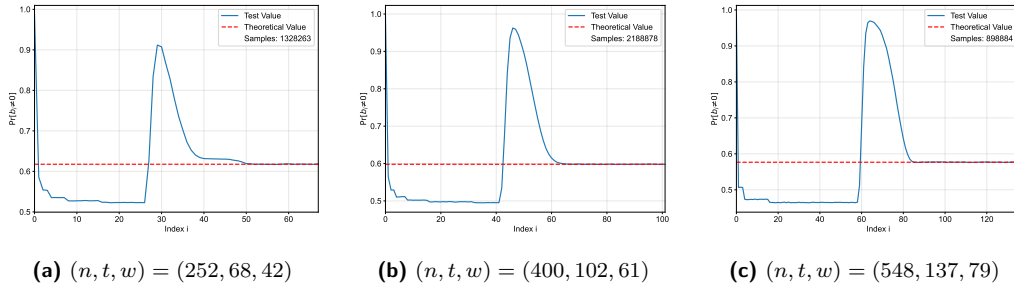
**Figure 3:** Experimental results on $\Pr[b_i \neq 0]$ versus index i under some LESS parameters

Table 4: Simulation result in “Loop Abort” model (statically-addressed $\mathbf{f}$)

(n, t, w, s)	Trials	Min	Max	Average
(252, 192, 36, 2)	985	2	4	2.02
(252, 68, 42, 4)	280	6	14	9.23
(252, 45, 34, 8)	113	30	82	43.73
(400, 220, 68, 2)	990	2	4	2.00
(400, 102, 61, 4)	292	4	10	6.79
(548, 345, 75, 2)	344	2	4	2.01
(548, 137, 79, 4)	284	4	10	5.67

can be followed by multiple attacks to achieve higher efficiency, the inherent uncertainty of fault injection introduces unpredictability into the state of array $\mathbf{f}$. The corresponding simulation results are summarized in Table 4.

6 Theoretical Analysis

6.1 Leakage Demand Analysis

After examining the experimental results, we can theoretically analyze the number of leakages and signing-oracle accesses required to recover all secret matrices and derive an approximate estimate.

6.1.1 Leakage Demand for Single Secret Matrix Recovery

For a particular secret matrix $\mathbf{Q}_{i^*}$, if we obtain its permutation, then its coefficients can be directly recovered. Then, the number of leakages needed to recover the secret matrix is just the number of leakages required to recover its permutation, and we can obtain the following theorem:

Theorem 1. *Recovering the permutation of a single private monomial matrix $\mathbf{Q}_{i^*}$ requires approximately $2 \log_2 n$ related leakages.*

Proof. This problem can be abstracted as the following “labelling” problem: Suppose there are $n = 2k \geq 252$ different elements, and we randomly label k elements with 0 and the remaining k elements with 1. We perform the procedure T times, and each element’s label sequence can be written as a vector $\mathbf{s}_i \in \{0, 1\}^T$.

Since exactly k elements are labelled 0 and the remaining k elements are labelled 1 in each round of labelling, the probability that any two distinct elements i, j receive different labels in a single round is:

$$p_{\text{diff}} = \frac{2 \cdot C_{n-2}^{k-1}}{C_n^k} = \frac{2k^2}{n(n-1)} \stackrel{2k=n}{=} \frac{n}{2(n-1)} \stackrel{n \geq 252}{\approx} \frac{1}{2}. \quad (15)$$

As each labelling process is independent, the probability that the $\mathbf{s}_i = \mathbf{s}_j$ retains after T times labelling iterations is $(1 - p_{\text{diff}})^T \approx 2^{-T}$. We require the expected number of collisions to be less than 1 to guarantee valid recovery with high probability. As there are $C_n^2 = \frac{n^2-n}{2}$ different (i, j) pairs, the expected number of collisions is $\frac{n^2-n}{2} \cdot 2^{-T}$. Therefore, we obtain:

$$\frac{n^2-n}{2} \cdot 2^{-T} < 1 \implies T > 2 \log_2 n - 1 + \log_2 \left(1 - \frac{1}{n}\right) \approx 2 \log_2 n - 1. \quad (16)$$

Thus, the expected number of leakage instances required is about $T \approx 2 \log_2 n$. For the convenience of subsequent analysis, we use its ceiling value $\lceil 2 \log_2 n \rceil$. $\square$

Table 5: $\text{Cop}(s-1, \lceil 2 \log_2 n \rceil)$ on LESS parameters

(n, t, w, s)	$s-1$	$\lceil 2 \log_2 n \rceil$	$\text{Cop}(s-1, \lceil 2 \log_2 n \rceil)$
(252, 192, 36, 2)	1	16	16.00
(252, 68, 42, 4)	3	16	58.35
(252, 45, 34, 8)	7	16	152.44
(400, 220, 68, 2)	1	18	18.00
(400, 102, 61, 4)	3	18	64.97
(548, 345, 75, 2)	1	19	19.00
(548, 137, 79, 5)	3	19	68.27

6.1.2 Leakage Demand for All Secret Matrices Recovery

Since we have $s-1$ distinct secret matrices, each effective leakage corresponds to one of them. However, the average number of required effective leakages exceeds the intuitive value $2(s-1) \log_2 n$. This is because some accesses may be wasted once certain secret matrices have already been fully recovered while others have not yet reached their required thresholds. Determining the actual expected value reduces to the multiple coupon collection problem [New60, FS14]: given N coupon types, we draw one random coupon per trial, aiming to collect at least M coupons of each type. We use $\text{Cop}(N, M)$ to denote the expected number of trials needed. For $M \geq 2$, the exact value can be calculated using (17) [New60].

$$\text{Cop}(N, M) = \int_0^\infty \left(1 - \left[1 - e^{-t} \sum_{k=0}^{M-1} \frac{t^k}{k!} \right]^N \right) dt \quad (17)$$

For the LESS parameters, we can use (17) to calculate all $\text{Cop}(s-1, \lceil 2 \log_2 n \rceil)$ values, as shown in Table 5.

6.2 Required Signing Oracle Accesses in Fault Models

In the previous section, we only considered the amount of leakage required to recover all secret matrices. However, after determining the required number of leakages, it is also necessary to evaluate the number of signing oracle accesses in different fault models.

6.2.1 Required Accesses in Single Iteration Skip Model

For the “Single Iteration Skip” model with zero-filled array $\mathbf{f}$, we selected attack parameters that maximize $\Pr[\text{successful skip} \mid b_i \neq 0]$. If we set this conditional probability to be ϵ , then the overall successful skip rate is:

$$\Pr[\text{successful skip}] = \epsilon \cdot \Pr[b_i \neq 0] = \epsilon \cdot \frac{w}{t} \quad (18)$$

As array $\mathbf{f}$ contains all zeros in this case, each successful skip indicates one effective leakage. Therefore, the expression of expected number of accesses is:

$$\mathbb{E}_{11}[\text{accesses}] = \frac{\text{Cop}(s-1, \lceil 2 \log_2 n \rceil)}{\Pr[\text{successful skip}]} = \frac{t \cdot \text{Cop}(s-1, \lceil 2 \log_2 n \rceil)}{w \cdot \epsilon} \quad (19)$$

According to (19), we can calculate the expected number of signing-oracle accesses for different successful-skip rates. The values are shown in Table 6, and the experimental results fit the formula well.

If array $\mathbf{f}$ is statically addressed with no guaranteed initial value, the attacker needs to obtain a normal signature before conducting the attack. Therefore, compared to the

Table 6: Theoretical trials in “Single Iteration Skip” model with zero-filled array $\mathbf{f}$

(n, t, w, s)	$\epsilon = 0.6$	$\epsilon = 0.8$	$\epsilon = 0.99$	Experiment
(252, 192, 36, 2)	142.22	106.67	86.20	83.59
(252, 68, 42, 4)	157.45	118.09	95.43	156.81
(252, 45, 34, 8)	336.26	252.20	203.80	223.94
(400, 220, 68, 2)	97.06	72.79	58.82	60.53
(400, 102, 61, 4)	181.06	135.80	109.74	131.88
(548, 345, 75, 2)	145.67	109.25	88.28	85.13
(548, 137, 79, 4)	197.32	147.99	119.59	144.09

“zero-filled” case, twice as many accesses are needed, and we obtain the expected number of accesses as:

$$\mathbb{E}_{12}[\text{accesses}] = \frac{2t \cdot \text{Cop}(s-1, \lceil 2 \log_2 n \rceil)}{w \cdot \epsilon} \quad (20)$$

6.2.2 Required Accesses in Loop Abort Model

With zero-filled array $\mathbf{f}$: In the “Loop Abort” model, as one successful attack may trigger multiple leakages, we can calculate the expected number of leakages for one trial after the i^* -th iteration to be $(t-1-i^*) \cdot \frac{w}{t}$. However, if array $\mathbf{f}$ contains all zeros after the fault injection, no seed will be published. To derive the expected access numbers, we can calculate the probability of obtaining exact u leakages if we perform the fault injection attack after the i^* -th iteration $\text{prob}_u(i^*)$:

$$\text{prob}_u(i^*) = \frac{1}{C_t^{t-i^*-1}} \cdot \begin{cases} C_w^u \cdot C_{t-w}^{t-i^*-1-u}, & 1 \leq i^* \leq w-1 \\ C_{t-w}^{t-i^*-1} + C_{t-w}^{t-w-i^*-1}, & u=0 \text{ and } t-i^*-1 \geq w \\ C_{t-w}^{t-i^*-1}, & u=0 \text{ and } t-i^*-1 < w \end{cases} \quad (21)$$

Therefore, for a fixed i^* we can calculate the expected number of accesses using the following recurrence relation. For simplicity, we consider the required number of leakages to be the integer $\lceil \text{Cop}(s-1, \lceil 2 \log_2 n \rceil) \rceil$.

$$\text{rec}_\ell(i^*) = \begin{cases} \frac{1 + \sum_{j=1}^{w-1} (\text{prob}_j(i^*) \cdot \text{rec}_{(\ell+j)}(i^*))}{1 - \text{prob}_0(i^*)}, & \ell < \lceil \text{Cop}(s-1, \lceil 2 \log_2 n \rceil) \rceil \\ 0, & \text{otherwise} \end{cases} \quad (22)$$

Following (22), the value $\text{rec}_0(i^*)$ is the expected effective accesses. Considering the successful abort rate ϵ , we obtain (23) and its plot for the first three LESS parameters in Figure 4.

$$\mathbb{E}_{21}[\text{accesses}](i^*) = \frac{\text{rec}_0(i^*)}{\epsilon} \quad (23)$$

With statically-addressed array $\mathbf{f}$: In this case, array $\mathbf{f}$ is statically addressed with unknown initial values, and the “one attack after a normal signature” strategy should be adopted. Since array $\mathbf{f}$ definitely contains a 1 after a normal execution of the signing process, there is no need to consider the situation in which an all-zero $\mathbf{f}$ leads to no seed publication. Here, the optimal attack point occurs when $i^* = 0$. This implies that only f_0 is updated by the process if the attack succeeds. The number of effective leakages in a successful attack is:

Table 7: Theoretical result in “Loop Abort” model with statically-addressed array

(n, t, w, s)	$\epsilon = 0.99$	Simulation	(n, t, w, s)	$\epsilon = 0.99$	Simulation
(252, 192, 36, 2)	2.02	2.02	(400, 102, 61, 4)	6.06	6.79
(252, 68, 42, 4)	8.08	9.23	(548, 345, 75, 2)	2.02	2.01
(252, 45, 34, 8)	38.38	43.73	(548, 137, 79, 4)	6.06	5.67
(400, 220, 68, 2)	2.02	2.00			

$$L = w \cdot \frac{t-1}{t} \cdot \frac{t-w}{t} = \frac{w \cdot (t-1) \cdot (t-w)}{t^2} \quad (24)$$

Performing an attack requires accessing the signing oracle twice, and the expected leakage demand is $\text{Cop}(s-1, \lceil 2 \log_2 n \rceil)$. Assuming a successful abort rate of ϵ for a single attack, we obtain (25) as the expected number of accesses. For the seven parameter sets, the corresponding $\mathbb{E}_{22}[\text{accesses}]$ values are displayed in Table 7.

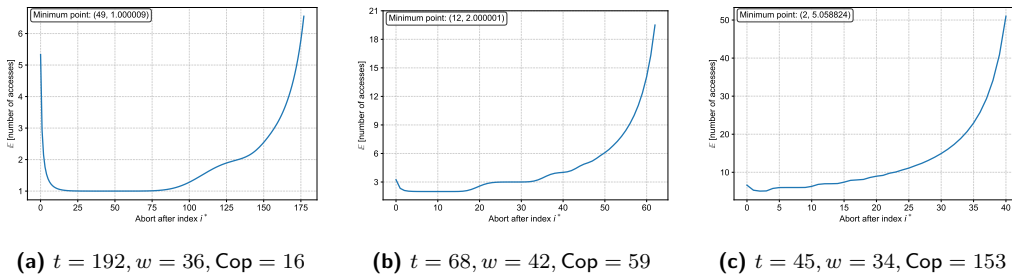
$$\mathbb{E}_{22}[\text{accesses}] = \frac{2}{\epsilon} \left\lceil \frac{\text{Cop}(s-1, \lceil 2 \log_2 n \rceil)}{L} \right\rceil = \frac{2}{\epsilon} \left\lceil \frac{t^2 \cdot \text{Cop}(s-1, \lceil 2 \log_2 n \rceil)}{w \cdot (t-1) \cdot (t-w)} \right\rceil \quad (25)$$

7 Countermeasures and Conclusion

7.1 Countermeasures

To prevent similar attacks, one direct mitigation is to randomly initialize the array $\mathbf{f}$ before use to eliminate its predictability, like the zero-fill mechanism on some devices, or its residual values resulted from the fixed address on the stack space. However, such a countermeasure remains limited under fault-attack models, because an adversary may bypass the initialization step by instruction skipping or fault injection, thereby restoring the vulnerable execution state. A more robust solution is therefore to eliminate the structural dependency exploited by the attack itself, namely by removing the intermediate array and using the challenge array $\mathbf{b}$ directly to control seed release in the GGM tree mechanism.

Nevertheless, even this structural change does not rule out stronger fault models. For example, an adversary may still target the GGM tree construction procedure or the array-access process in order to induce overlapping outputs and gradually leak secret information. To address this broader risk, we recommend introducing a consistency check before the signature is returned, verifying that the output data remain consistent with the challenge and do not contain abnormal overlaps. Compared with computational costs such as RREF computations, this additional check is relatively inexpensive, yet it can substantially improve resistance against single- and double-fault attacks.

**Figure 4:** Plot of $\mathbb{E}_{21}[\text{accesses}](i^*)$ for LESS parameters with $n = 252, \epsilon = 1$

With regard to the non-uniformity of the sample results in the `SampleChallenge` function, we offer two fix methods: The first method is to initialize the counter c to zero to force the buffer refresh between the two phases and eliminate the counting mismatch. A more robust method is to initialize c to 64 when refreshing the buffer, and subtract the exact bit count per iteration, and refresh when insufficient bits remain. Although this alters the expected random bits by at most 64, the impact is negligible, and it fully restores the uniform probability distribution, while optimizing random bit management.

7.2 Conclusion and Future Work

In this work, we presented a novel fault injection attack targeting LESS. Unlike existing fault attacks that focus on the complex GGM tree structure, our approach exploits vulnerabilities inherent in the computation of the binary array $\mathbf{f}$. Through extensive cross-device testing, we identified two common initial-state cases for array $\mathbf{f}$ and proposed two fault models. Our attack enables bidirectional derivation between attack parameters and fault position, both eliminating heuristic guessing of the fault position and enhancing the attacker’s adaptability. Based on this, we designed a series of supporting algorithms and performed attack experiments and simulations that confirm the high efficiency of our approach. Beyond experimental validation, we also provide a theoretical analysis of the expected number of signing-oracle accesses required for successful recovery.

In the future, we intend to generalize our attack framework to additional post-quantum cryptosystems, and systematically evaluate how such loop perturbation attacks may affect their physical security in real applications. Additionally, it is also interesting to evaluate how the implementation bug in `SampleChallenge`, which triggers non-uniformly distributed values of $\Pr[b_i \neq 0]$ across different indices, affect the security level of the scheme.

Acknowledgements

We are thankful to the anonymous reviewers of CHES 2026 for their useful and constructive comments, suggestions and feedbacks.

Chao Sun is supported by the National Natural Science Foundation of China (62572121) and the Frontier Technology R&D Program of Jiangsu (BF2025067). Quan Yuan is supported by the Shandong Provincial Natural Science Fund for Excellent Young Scientists Fund Program (Overseas) (2025HWYQ-023). Yu Yu is supported by the National Natural Science Foundation of China (Grant Nos. 62125204 and 92270201) and the Innovation Program for Quantum Science and Technology (Nos. 2021ZD0302901 and 2021ZD0302902).

References

- [AAB⁺22] Carlos Aguilar-Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Gilles Zémor, Jurjen Bos, Arnaud Dion, Jerome Lacan, Jean-Marc Robert, and Pascal Veron. HQC. Technical report, National Institute of Standards and Technology, 2022. available at <https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions>.
- [ABF⁺18] Christopher Ambrose, Joppe W. Bos, Björn Fay, Marc Joye, Manfred Lochter, and Bruce Murray. Differential attacks on deterministic signatures. In Nigel P. Smart, editor, *CT-RSA 2018*, volume 10808 of *LNCS*, pages 339–353. Springer, Cham, April 2018.

- [AMH⁺16] Amro Awad, Pratyusa Manadhata, Stuart Haber, Yan Solihin, and William Horne. Silent shredder: Zero-cost shredding for secure non-volatile main memory controllers. *ACM SIGPLAN Notices*, 51(4):263–276, 2016.
- [AWK⁺25] Samy Amer, Yingchen Wang, Hunter Kippen, Thinh Dang, Daniel Genkin, Andrew Kwong, Alexander Nelson, and Arkady Yerukhimovich. Pq-hammer: End-to-end key recovery attacks on post-quantum cryptography using rowhammer. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 3567–3582. IEEE, 2025.
- [BBB⁺23] Marco Baldi, Alessandro Barenghi, Luke Beckwith, Jean-François Biasse, Andre Esser, Kris Gaj, Kamyar Mohajerani, Gerardo Pelosi, Edoardo Persichetti, Markku-Juhani O. Saarinen, Paolo Santini, and Robert Wallace. LESS — Linear Equivalence Signature Scheme. Technical report, National Institute of Standards and Technology, 2023. available at <https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures>.
- [BBB⁺24a] Marco Baldi, Alessandro Barenghi, Michele Battagliola, Sebastian Bitzer, Marco Gianvecchio, Patrick Karl, Felice Manganiello, Alessio Pavoni, Gerardo Pelosi, Paolo Santini, Jonas Schupp, Edoardo Signorini, Freeman Slaughter, Antonia Wachter-Zeh, and Violetta Weger. CROSS — Codes and Restricted Objects Signature Scheme. Technical report, National Institute of Standards and Technology, 2024. available at <https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures>.
- [BBB⁺24b] Marco Baldi, Alessandro Barenghi, Luke Beckwith, Jean-François Biasse, Tung Chou, Andre Esser, Kris Gaj, Patrick Karl, Kamyar Mohajerani, Gerardo Pelosi, Edoardo Persichetti, Markku-Juhani O. Saarinen, Paolo Santini, Robert Wallace, and Floyd Zveydinger. LESS — Linear Equivalence Signature Scheme. Technical report, National Institute of Standards and Technology, 2024. available at <https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures>.
- [BBPS21] Alessandro Barenghi, Jean-François Biasse, Edoardo Persichetti, and Paolo Santini. LESS-FM: Fine-tuning signatures from the code equivalence problem. In Jung Hee Cheon and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021*, pages 23–43. Springer, Cham, 2021.
- [BCC⁺23] Gustavo Banegas, Kévin Carrier, André Chailloux, Alain Couvreur, Thomas Debris-Alazard, Philippe Gaborit, Pierre Karpman, Johanna Loyer, Ruben Niederhagen, Nicolas Sendrier, Benjamin Smith, and Jean-Pierre Tillich. Wave. Technical report, National Institute of Standards and Technology, 2023. available at <https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures>.
- [BCM⁺23] Marcus Brinkmann, Chitchanok Chuengsatiansup, Alexander May, Julian Nowakowski, and Yuval Yarom. Leaky McEliece: Secret key recovery from highly erroneous side-channel information. Cryptology ePrint Archive, Report 2023/1536, 2023.
- [BDL⁺11] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES 2011*, volume 6917 of *LNCS*, pages 124–142. Springer, Berlin, Heidelberg, September / October 2011.

- [BEP⁺25] Luke Beckwith, Andre Esser, Edoardo Persichetti, Paolo Santini, and Floyd Zveydinger. LESS is even more: Optimizing digital signatures from code equivalence. Cryptology ePrint Archive, Report 2025/1424, 2025.
- [Beu20] Ward Beullens. Not enough LESS: An improved algorithm for solving code equivalence problems over $\mathbb{F}_q$. In Orr Dunkelman, Michael J. Jacobson, Jr., and Colin O’Flynn, editors, *SAC 2020*, volume 12804 of *LNCS*, pages 387–403. Springer, Cham, October 2020.
- [Beu22] Ward Beullens. Breaking rainbow takes a weekend on a laptop. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 464–479. Springer, Cham, August 2022.
- [BMPS20] Jean-François Biasse, Giacomo Micheli, Edoardo Persichetti, and Paolo Santini. LESS is more: Code-based signatures without syndromes. In Abderrahmane Nitaj and Amr M. Youssef, editors, *AFRICACRYPT 20*, volume 12174 of *LNCS*, pages 45–65. Springer, Cham, July 2020.
- [BP18] Leon Groot Bruinderink and Peter Pessl. Differential fault attacks on deterministic lattice signatures. *IACR TCHES*, 2018(3):21–43, 2018.
- [CD23] Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 423–447. Springer, Cham, April 2023.
- [CFS01] Nicolas Courtois, Matthieu Finiasz, and Nicolas Sendrier. How to achieve a McEliece-based digital signature scheme. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 157–174. Springer, Berlin, Heidelberg, December 2001.
- [CML17] Lily Chen, D Moody, and Y Liu. Nist post-quantum cryptography standardization. *Transition*, 800(131A):164, 2017.
- [CMR25] Maciej Czaprynski, Anisha Mukherjee, and Sujoy Sinha Roy. Correlation power analysis of LESS and CROSS. In Abderrahmane Nitaj, Svetla Petkova-Nikova, and Vincent Rijmen, editors, *AFRICACRYPT 25*, volume 15651 of *LNCS*, pages 270–295. Springer, Cham, July 2025.
- [CPS23] Tung Chou, Edoardo Persichetti, and Paolo Santini. On linear equivalence, canonical forms, and digital signatures. Cryptology ePrint Archive, Report 2023/1533, 2023.
- [DCP⁺20] Jintai Ding, Ming-Shing Chen, Albrecht Petzoldt, Dieter Schmidt, Bo-Yin Yang, Matthias J. Kannwischer, and Jacques Patarin. Rainbow. Technical report, National Institute of Standards and Technology, 2020. available at <https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions>.
- [FGO⁺13] Jean-Charles Faugère, Valérie Gauthier-Umaña, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. A distinguisher for high-rate mceliece cryptosystems. *IEEE Trans. Inf. Theory*, 59(10):6830–6844, 2013.
- [FKK⁺22] Michael Fahr, Hunter Kippen, Andrew Kwong, Thinh Dang, Jacob Lichtinger, Dana Dachman-Soled, Daniel Genkin, Alexander Nelson, Ray A. Perlner, Arkady Yerukhimovich, and Daniel Apon. When frodo flips: End-to-end key recovery on FrodoKEM via rowhammer. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 979–993. ACM Press, November 2022.

- [FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Berlin, Heidelberg, August 1987.
- [FS14] Marco Ferrante and Monica Saltalamacchia. The coupon collector's problem. *Materials mathematics*, pages 0001–35, 2014.
- [GGHS25] Lewis Glabush, Felix Günther, Kathrin Hövelmanns, and Douglas Stebila. Verifiable decapsulation: Recognizing faulty implementations of post-quantum KEMs. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part III*, volume 16002 of *LNCS*, pages 543–574. Springer, Cham, August 2025.
- [GGM84] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. In *25th Annual Symposium on Foundations of Computer Science, 1984.*, pages 464–479, 1984.
- [GJJ22] Qian Guo, Andreas Johansson, and Thomas Johansson. A key-recovery side-channel attack on classic McEliece implementations. *IACR TCHES*, 2022(4):800–827, 2022.
- [HBD⁺22] Andreas Hülsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kölbl, 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. available at <https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022>.
- [HPT25] Calvin Abou Haidar, Quentin Payet, and Mehdi Tibouchi. Crowhammer: Full key recovery attack on falcon with a single rowhammer bit flip. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part V*, volume 16004 of *LNCS*, pages 103–135. Springer, Cham, August 2025.
- [IMS⁺22] Saad Islam, Koksul Mus, Richa Singh, Patrick Schaumont, and Berk Sunar. Signature correction attack on Dilithium signature scheme. In *2022 IEEE European Symposium on Security and Privacy*, pages 647–663. IEEE Computer Society Press, June 2022.
- [JAC⁺20] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, David Urbanik, Geovandro Pereira, Koray Karabina, and Aaron Hutchinson. SIKE. Technical report, National Institute of Standards and Technology, 2020. available at <https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions>.
- [LDK⁺22] Vadim Lyubashevsky, Léoucas, Eike Kiltz, Tancrede Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2022. available at <https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022>.
- [Leo82] J. Leon. Computing automorphism groups of error-correcting codes. *IEEE Transactions on Information Theory*, 28(3):496–511, 1982.

- [LKS⁺17] Juneyoung Lee, Yoonseung Kim, Youngju Song, Chung-Kil Hur, Sanjoy Das, David Majnemer, John Regehr, and Nuno P Lopes. Taming undefined behavior in llvm. *ACM SIGPLAN Notices*, 52(6):633–647, 2017.
- [MAKK24] Puja Mondal, Supriya Adhikary, Suparna Kundu, and Angshuman Karmakar. ZKFault: Fault attack analysis on zero-knowledge based post-quantum digital signature schemes. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part VIII*, volume 15491 of *LNCS*, pages 132–167. Springer, Singapore, December 2024.
- [McE78] Robert J McEliece. A public-key cryptosystem based on algebraic. *Coding Thv*, 4244(1978):114–116, 1978.
- [MKN⁺25] Puja Mondal, Suparna Kundu, Hikaru Nishiyama, Supriya Adhikary, Daisuke Fujimoto, Yuichi Hayashi, and Angshuman Karmakar. Fault to forge: Fault assisted forging attacks on LESS signature scheme. Cryptology ePrint Archive, Report 2025/1838, 2025.
- [New60] Donald J Newman. The double dixie cup problem. *The American Mathematical Monthly*, 67(1):58–61, 1960.
- [OC14] Colin O’Flynn and Zhizhang (David) Chen. ChipWhisperer: An open-source platform for hardware embedded security research. In Emmanuel Prouff, editor, *COSADE 2014*, volume 8622 of *LNCS*, pages 243–260. Springer, Cham, April 2014.
- [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. available at <https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022>.
- [PGD⁺22] Sabine Pircher, Johannes Geier, Julian Danner, Daniel Mueller-Gritschneider, and Antonia Wachter-Zeh. Key-recovery fault injection attack on the classic McEliece KEM. Cryptology ePrint Archive, Report 2022/1529, 2022.
- [PS23] Edoardo Persichetti and Paolo Santini. A new formulation of the linear equivalence problem and shorter LESS signatures. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part VII*, volume 14444 of *LNCS*, pages 351–378. Springer, Singapore, December 2023.
- [PSS⁺18] Damian Poddebniak, Juraj Somorovsky, Sebastian Schinzel, Manfred Lochter, and Paul Rösler. Attacking deterministic signature schemes using fault attacks. In *2018 IEEE European Symposium on Security and Privacy*, pages 338–352. IEEE Computer Society Press, April 2018.
- [PT16] Aurélie Phezzo and Jean-Pierre Tillich. An efficient attack on a code-based signature scheme. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016*, pages 86–103. Springer, Cham, 2016.
- [Rob23] Damien Robert. Breaking SIDH in polynomial time. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 472–503. Springer, Cham, April 2023.

- [SAB⁺22] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehlé, and Jintai Ding. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2022. available at <https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022>.
- [Sen99] Nicolas Sendrier. *The support splitting algorithm*. PhD thesis, INRIA, 1999.
- [SS13] Nicolas Sendrier and Dimitris E. Simos. The hardness of code equivalence over and its application to code-based cryptography. In Philippe Gaborit, editor, *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013*, pages 203–216. Springer, Berlin, Heidelberg, June 2013.
- [SS18] Kanad Sinha and Simha Sethumadhavan. Practical memory safety with rest. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 600–611. IEEE, 2018.
- [SS25] Jonas Schupp and Georg Sigl. A horizontal attack on the codes and restricted objects signature scheme (CROSS). Cryptology ePrint Archive, Report 2025/116, 2025.
- [Ste94] Jacques Stern. A new identification scheme based on syndrome decoding. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 13–21. Springer, Berlin, Heidelberg, August 1994.
- [STM⁺08] Falko Strenzke, Erik Tews, H. Gregor Molter, Raphael Overbeck, and Abdulhadi Shoufan. Side channels in the McEliece PKC. In Johannes Buchmann and Jintai Ding, editors, *Post-quantum cryptography, second international workshop, PQCRYPTO 2008*, pages 216–229. Springer, Berlin, Heidelberg, October 2008.
- [Tar75] Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM (JACM)*, 22(2):215–225, 1975.
- [UMB⁺23] Vincent Quentin Ulitzsch, Soundes Marzougui, Alexis Bagia, Mehdi Tibouchi, and Jean-Pierre Seifert. Loop aborts strike back: Defeating fault counter-measures in lattice signatures with ILP. *IACR TCHES*, 2023(4):367–392, 2023.
- [YBF⁺11] Xi Yang, Stephen M Blackburn, Daniel Frampton, Jennifer B Sartor, and Kathryn S McKinley. Why nothing matters: The impact of zeroing. *Acm Sigplan Notices*, 46(10):307–324, 2011.
- [ZLYW23] Shiduo Zhang, Xiuhan Lin, Yang Yu, and Weijia Wang. Improved power analysis attacks on falcon. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part IV*, volume 14007 of *LNCS*, pages 565–595. Springer, Cham, April 2023.