Single-Trace Power Analysis of LESS Key Generation

Süleyman Emir Akın[1], Abdullah Talayhan[2] and Özcan Öztürk[1]

[1] Faculty of Engineering and Natural Sciences, Sabancı University, Istanbul, Türkiye,
{emir.akin,ozcan.ozturk}@sabanciuniv.edu

[2] Computer Engineering Department, Bilkent University, Ankara, Türkiye,
talayhan@bilkent.edu.tr

Abstract. This paper presents a side-channel attack on the Linear Equivalence Signature Scheme (LESS) v2.0. LESS derives its security from the Linear Equivalence Problem and was evaluated as a candidate during Round 2 of the NIST post-quantum cryptography standardization process. LESS secret keys are used to generate monomial matrices, which are stored efficiently in two one-dimensional lists: the permutation list and the coefficient list. Recovering the secret monomial matrices is sufficient to forge signatures, as they are the values actually used during signing. We propose a profiled, single-trace horizontal attack on LESS key generation that recovers the full secret monomial matrices. First, the monomial coefficients that are multiplied by the dense part of the public generator matrix are recovered via power analysis of the matrix multiplication function. Next, we attack the reduced row echelon form function to recover the permutation list. We then complete the attack algebraically via two independent paths: the Primary attack and the Secondary attack. The Primary attack uses only the recovered coefficients in matrix multiplication together with their permutation positions, and a known parity-check matrix equation. The Secondary attack is an alternative that relies on another algebraic relation between the secret key and the public key and uses all the recovered values. We validated our attack on an ARM Cortex-M4 microcontroller. On the NIST Category 1 parameter set, the Primary attack achieves a 99.1% exact-recovery rate and the Secondary attack achieves a 99% exact-recovery rate, over 6000 independent keys. We also analyze potential countermeasures and show that independently shuffling the row processing order within each column reduces the success rate of our attack to negligible levels, providing protection against the specific attack vector demonstrated in this paper.

Keywords: Side-channel attacks · Post-quantum cryptography · LESS

1 Introduction

Traditional digital signature schemes such as RSA [RSA78], DSA [Nis92], and ECDSA [JMV01] rely on integer factorization and the discrete logarithm problem, which are computationally hard to solve for classical computers. However, with the rise of quantum computers and the algorithms they enable, an exponential speedup for solving these problems becomes feasible. In particular, Shor's algorithm [Sho99, Sho94] can solve both integer factorization and the discrete logarithm problem in polynomial time on a cryptographically relevant quantum computer, rendering these classical schemes insecure. To address this security issue, the National Institute of Standards and Technology (NIST) started a standardization process for post-quantum digital signature schemes in 2016. This process resulted in three digital signature schemes being chosen for standardization, namely ML-DSA [DKL+18], SLH-DSA [BHK+19], and FN-DSA [FHK+18].

In 2023, NIST issued an additional call for digital signature proposals, motivated by the limited diversity of the initial standards, as ML-DSA and FN-DSA both derive their security from lattice assumptions, while SLH-DSA relies on hash-based assumptions. The second round candidates, announced in 2024, are mostly based on other computational assumptions such as multivariate polynomials, codes, isogenies, and symmetric-key primitives [nis]. Among these, only LESS [BBB+25a] and CROSS [BBB+25b] are code-based signature schemes, with LESS built on the hardness of the Linear Equivalence Problem (LEP).

Although LESS was ultimately not advanced to the third round of NIST's additional signatures process, the implementation security of LEP-based signatures remains relevant: the LESS reference implementation is publicly available, and related schemes employing similar techniques remain potential research targets. NIST's decision was announced on May 14, 2026, after this work was completed.[1]

A prior side-channel attack on LESS v1.2 signing was presented in [CMR25]. Their attack uses correlation power analysis targeting the multiplication of a public generator matrix with an ephemeral secret monomial during signing, and recovers the secret key by aggregating information across multiple signature responses of the form $Q^{-1}\tilde{Q}$. This attack path is no longer viable in v2.0, as the updated signature compression scheme withholds the information required to link the ephemeral secret back to the persistent key. Our attack circumvents this limitation entirely by targeting key generation rather than signing, requiring only a single power trace and no signature responses.

A side-channel attack on LESS v2.0 signing was presented in [HKLM25]. They cast key recovery as a LESS-specific Hidden Number Problem: one bit of leakage of the ephemeral commitment randomness per run of the identification protocol, aggregated across many signatures together with their transcripts, suffices to reconstruct the persistent secret key. To obtain the required leakage they mount two independent attacks on two different operations, namely a timing attack on the systematic-form transformation RREF* and a power attack on the matrix multiplication, either of which suffices to instantiate the Hidden Number Problem. Both attacks target the commitment randomness that is freshly sampled in every protocol run. Using the Hidden Number Problem, they recover the permutation list and then complete the attack algebraically by recovering the coefficient list. Their Appendix A gives two forms of algebraic completion: a permutation-only variant, which they use to reduce their signature requirements, and a joint permutation and coefficient variant, which is not used elsewhere in their paper but which we use to complete our primary attack.

The power side-channel they presented targets the matrix multiplication during signing. From this routine they recover partial information about the permutation of the ephemeral commitment randomness, namely a single bit of each permutation entry, and aggregate it across multiple signatures to reconstruct the persistent secret key. Although key generation executes the same function, this attack does not carry over: key generation invokes the multiplication only once and on the persistent secret monomial, emitting no signature and no transcript, so there is nothing to aggregate over. Our attack also targets the matrix multiplication, however we recover the monomial coefficient list rather than the permutation.

The timing side-channel targets the RREF* function during signing. Because RREF* additionally recovers the column permutation that brings the matrix into systematic form, it performs a column-index-dependent search through the commitment randomness permutation, terminated by an early `break`, whose timing leaks the permutation and can be exploited to recover ephemeral commitment randomness. This attack is likewise inapplicable to our setting, because the code fragment they exploit is not present in the plain RREF function used in key generation. In our setting we exploit the plain

[1] According to NIST IR 8610 [Moo26], the elimination was driven by mathematical cryptanalysis and performance considerations; NIST notes that no side-channel results in the second round were decisive in any candidate's elimination.

RREF to recover the permutation list using power-side channel, casting the recovery as a column-matching problem.

Our contributions are as follows:

- We present a profiled, single-trace horizontal attack against the key generation of LESS v2.0 that recovers the full secret monomial matrices. To the best of our knowledge, this is the first single-trace side-channel attack on the key generation of LESS v2.0.
- We identify two distinct leakage targets within the key generation pipeline: the matrix multiplication function, which leaks the monomial coefficients that are multiplied with the dense part of the public generator matrix, and the reduced row echelon form function, which leaks the monomial permutation. From the coefficients that are recovered in matrix multiplication and their permutation positions alone, the full secret key follows via a known algebraic relation between the secret key and the public key [HKLM25, CPS25]. As a secondary alternative, we derive our own algebraic relation that uses a different mathematical basis and uses all the recovered values.
- We validate the attack on an ARM Cortex-M4 microcontroller using the NIST Category 1 parameter set ($k = 126$, $n = 252$, $s = 2$), evaluated over 6000 independent keys. The Primary attack recovers the full secret monomial matrices with a 99.1% exact-recovery rate, while the Secondary attack achieves a 99% exact-recovery rate.
- We analyze countermeasures against our attack and propose independently shuffling the row processing order per column inside the matrix multiplication function. We experimentally confirm that this renders the attack infeasible.

2 Background

In this section, we first give an overview of the underlying mathematics behind LESS, followed by a brief overview of side-channel analysis concepts. We then explain the key generation process and conclude with our threat model.

2.1 Mathematical Preliminaries

Linear Code.

A linear code $\mathcal{C}$ with parameters $[n, k]$ defined over a finite field $\mathbb{F}_q$ consists of a k-dimensional subspace within $\mathbb{F}_q^n$. In the context of LESS, the field is fixed to $\mathbb{F}_{127}$, i.e., $q = 127$. For a matrix $A \in \mathbb{F}_q^{k \times n}$, we write $A[i][j]$ for the entry at row i and column j, with indices starting at 0.

Systematic Form.

A matrix of the form $G = (I_k \mid M)$ is said to be in systematic form. The systematic form is obtained by computing the Reduced Row Echelon Form (RREF) of a given matrix. We refer to the first k columns (those forming I_k) as the *identity columns* and to the remaining $n - k$ columns (those forming M) as the *dense columns*.

Monomial Matrices.

A monomial matrix (or generalized permutation matrix) is a square matrix with exactly one non-zero entry in each row and each column. We denote monomial matrices by Q and write $Q = D \cdot P$ as the product of a diagonal matrix D and a permutation matrix

P. Consequently, right-multiplication by Q scales each column by a non-zero coefficient and then permutes the columns. Since each of D and P can be stored efficiently using a one-dimensional list, LESS stores monomial matrices as two one-dimensional lists, which we call the *coefficient list*

$$\mathbf{c} = (c_0, c_1, \ldots, c_{n-1}) \in \left(\mathbb{F}_q^*\right)^n \qquad \text{(encoding the diagonal of } D\text{)}$$

and the *permutation list* $\pi \in S_n$ (the symmetric group on n elements, encoding P).

These monomial matrices are multiplied with the systematic form matrices. The monomial coefficients that are multiplied with the identity part are referred to as *identity coefficients*, and those multiplied with the dense part are referred to as *dense coefficients*; their corresponding permutations are referred to as the *identity permutation* and the *dense permutation*, respectively.

Linear Equivalence Problem.

The code equivalence problem is classical in coding theory. In this context, equivalence means the existence of a linear isometry between the two codes. The linear equivalence problem is a special case of the code equivalence problem that asks whether one can find an isometry between two linear codes using a monomial map.

To be more precise, LESS considers a computational version of the Linear Equivalence Problem. Given two systematic generator matrices G and G', find a monomial map Q such that $G' = \mathrm{RREF}(G \cdot Q)$. Given G and Q, one can easily compute G'. The inverse direction is computationally hard. Given G and G', finding a monomial map Q that links them is the problem on which the security of LESS rests.

2.2 Side-Channel Analysis

Side-channel analysis (SCA) recovers secrets from physical measurements of a device executing a cryptographic algorithm, most commonly the instantaneous power consumption captured as a time series, or *power trace*. Since the power drawn at each cycle depends on the data manipulated, traces leak information about intermediate values.

SCA attacks are broadly categorized along two axes. First, *profiled* attacks assume the attacker has access to a target device on which secrets can be chosen freely, allowing the attacker to build a model of the leakage prior to attacking the target; *non-profiled* attacks do not. Second, *vertical* attacks combine many traces captured under the same secret to amplify the signal-to-noise ratio, while *horizontal* attacks exploit many leakage points within a single trace [CFG+10], typically by targeting an operation that is repeated many times under the same secret. The attack presented in this paper is profiled, horizontal, and single-trace.

Classical profiled attacks (template attacks) model the leakage at each point of interest as a Gaussian distribution conditioned on the target intermediate value. More recent work replaces this fixed parametric model with a neural network trained on profiling traces.

2.3 LESS Key Generation

The full key generation pipeline is given in Algorithm 1. The procedure takes a single seed $\mathsf{seed}_{\mathsf{sk}}$ that is used to initialize the random number generator (XOF). Using this random generator, it first generates a public key seed $\mathsf{seed}_{\mathsf{pk}}$, which is used to expand the dense part M, yielding $G_0 = (I_k \mid M)$, which is already in systematic form.

After that, using the same random number generator, it creates $s-1$ private seeds for the monomial matrices. It then samples a monomial matrix Q_ℓ from each seed and computes its inverse Q_ℓ^{-1}. Each inverse Q_ℓ^{-1} is then applied by right-multiplication,

Algorithm 1: LESS Key Generation

Input: Private seed $\mathsf{seed}_{\mathsf{sk}}$
Output: Private key sk and public key pk

$\mathsf{state}_{\mathsf{sk}} \leftarrow \mathrm{XOF.init}(\mathsf{seed}_{\mathsf{sk}})$ // Initialize the XOF
$\mathsf{seed}_{\mathsf{pk}} \leftarrow \mathrm{XOF}(\mathsf{state}_{\mathsf{sk}})$ // Seed for the public key
$G_0 \leftarrow \mathrm{SampleGenerator}(\mathsf{seed}_{\mathsf{pk}})$ // Starting generator matrix $(I_k \mid M)$
$(\mathsf{seed}_1, \ldots, \mathsf{seed}_{s-1}) \leftarrow \mathrm{XOF}(\mathsf{state}_{\mathsf{sk}})$ // Seeds for monomial matrices
for $\ell = 1, \ldots, s-1$ **do**
  $Q_\ell \leftarrow \mathrm{SampleMonomial}(\mathsf{seed}_\ell)$ // Sample monomial matrix
  $Q_\ell^{-1} \leftarrow \mathrm{Invert}(Q_\ell)$ // Compute its inverse
  $G_\ell \leftarrow \mathrm{RREF}(G_0 \cdot Q_\ell^{-1})$ // ℓ-th public key component
$\mathsf{sk} \leftarrow \mathsf{seed}_{\mathsf{sk}}$
$\mathsf{pk} \leftarrow (\mathsf{seed}_{\mathsf{pk}}, G_1, \ldots, G_{s-1})$
return $(\mathsf{sk}, \mathsf{pk})$

$G_0 \cdot Q_\ell^{-1}$, and the result is brought into RREF to create the public key components $G_1, \ldots, G_{s-1}$.

The public key is $(\mathsf{seed}_{\mathsf{pk}}, G_1, \ldots, G_{s-1})$; including $\mathsf{seed}_{\mathsf{pk}}$ allows any party to reconstruct G_0 without transmitting it. The private key is $\mathsf{seed}_{\mathsf{sk}}$ alone, since all derived values, the public key seed, the monomial seeds, and the monomial matrices, can be regenerated from it.

For more information on LESS and LEP, the reader is referred to the following references [BBB+25a, BBPS21, BBN+22, BBPS22, BWMG23, PS23, CPS25].

2.4 Threat Model

We assume the adversary has physical access to the target device and can measure its power consumption. The adversary is also assumed to have knowledge of the software being executed and can approximate the timing of the computations. Our attack requires a profiling phase, during which the attacker has access to the target device on which they can execute keygen with chosen seeds and capture arbitrarily many power traces. During the attack phase, the attacker observes a single keygen execution on the same device with an unknown seed and must recover the secret monomial matrices from that single trace. We use both Gaussian templates and Neural Networks as a distinguisher.

3 The Proposed Attack

Our attack recovers the full secret monomial matrices from a single power trace using two side-channel stages followed by an algebraic completion. In Stage 1 we recover the dense coefficients by attacking the matrix multiplication function, and in Stage 2 we recover the monomial permutation by attacking the RREF function. In Stage 3 the full secret key is then obtained algebraically along two independent paths: a primary path that uses only dense coefficients and dense permutations together with a known relation; and a secondary alternative that instead uses all the recovered values as well as our own derivation.

Throughout this section, we describe the attack on a single target inverse monomial Q_ℓ^{-1}, where $\ell \in \{1, \ldots, s-1\}$ is fixed. The same attack is applied independently to each of the $s-1$ inverse monomial matrices.

3.1 Recovering Dense Coefficients

Algorithm 2: Generator–monomial multiplication (`generator_monomial_mul`)

```
1 function generator_monomial_mul(res, G, monom):
2     for j ← 0 to n − 1 do
3         for i ← 0 to k − 1 do
4             res.values[i][monom.permutation[j]] ←
              fq_mul(G.values[i][j], monom.coefficients[j])
```

Generator–monomial multiplication.

In the computation $G_0 \cdot Q_\ell^{-1}$ of Algorithm 1 (line 8), the publicly known generator matrix G_0 is multiplied by the secret monomial matrix Q_ℓ^{-1}. Because Q_ℓ^{-1} is a monomial matrix, the multiplication reduces to independently scaling and permuting the columns of G_0, and the implementation therefore differs from a general matrix multiplication routine.

Algorithm 2 gives the pseudocode of the C implementation of this multiplication. The function takes three inputs. The first is the result generator matrix, the second is the generator matrix that is going to be multiplied, and the third is the monomial matrix. The outer loop iterates over the columns of G_0. In each iteration, the function scales the entire known column by the same unknown coefficient and writes the result to the permuted column index. Since arithmetic is performed over $\mathbb{F}_{127}$, a Barrett's reduction technique [Bar86] is employed as a specialized modular multiplication routine

Leakage Structure.

In every call to `fq_mul` issued by `generator_monomial_mul`, one operand is the public value $G_0[i][j]$ while the other is the secret coefficient $c_j^{(\ell)}$. Since the same unknown coefficient is multiplied with every entry of a known column, an attacker can directly target the recovery of $c_j^{(\ell)}$ by aggregating leakage evidence across all k rows of that column.

This reasoning does not apply to identity columns. In such a column, all elements are zero except for one. Consequently, the product in those rows is exactly 0 regardless of the value of $c_j^{(\ell)}$, providing no exploitable variance in the power consumption for those operations. Only the single row containing an entry of one leaks information, which is insufficient for reliable recovery.

To recover the secret coefficient from the traces, we build a full-covariance Gaussian template for each candidate coefficient given the known input. The unknown coefficient $c_j^{(\ell)}$ is then identified by aggregating the per-row log-likelihoods via Maximum Likelihood Estimation.

3.2 Recovering the Monomial Permutation

With the dense coefficients in hand, in this stage we recover the permutation list $\pi^{(\ell)}$ of Q_ℓ^{-1} by attacking the RREF function. The pseudocode of this function is given in Algorithm 3.

This is a standard implementation of RREF. The outer loop iterates k times, once per pivot. In each iteration, the algorithm first searches for a pivot by scanning rows from top to bottom within each column and advancing columns from left to right, stopping at the first nonzero entry. Once the pivot is found, its row is swapped to position row_to_reduce (line 6) and then scaled by the inverse of the pivot value so that the pivot entry becomes 1.

Algorithm 3: Reduced Row Echelon Form (RREF)

Input: $G \in \mathbb{F}_q^{k \times n}$
Output: G in reduced row echelon form

```
for row_to_reduce ∈ [k] do
    // Find the pivot
    for j ∈ [row_to_reduce, n) do
        for i ∈ [row_to_reduce, k) do
            if G[i, j] ≠ 0 then
                go to 6
    // Swap rows
6   G[row_to_reduce, :], G[i, :] ← G[i, :], G[row_to_reduce, :]
    // Scale the pivot row
7   G[row_to_reduce, :] ← G[row_to_reduce, j]^{-1} · G[row_to_reduce, :]
    // Eliminate all other rows
    for row ∈ [k] \ {row_to_reduce} do
        G[row, :] ← G[row, :] − G[row, j] · G[row_to_reduce, :]
return G
```

In the elimination loop that follows, the algorithm makes the pivot the only nonzero entry in its column by subtracting, from every other row, the pivot row scaled by that row's entry in the pivot column.

Leakage target.

We attack the inner update of the elimination loop in the *first* outer iteration ($\mathsf{row_to_reduce} = 0, j = 0$). In this update,

$$\textbf{for } \mathsf{row} \in [k] \setminus \{0\} \textbf{ do}$$
$$G[\mathsf{row}, :] \leftarrow G[\mathsf{row}, :] - G[\mathsf{row}, 0] \cdot G[0, :]$$

the load of $G[\mathsf{row}, :]$ reads an entire row of the matrix from SRAM, the multiplication performs n `fq_mul` calls against the pivot row, and the subtraction writes n updated entries back. All of these loads, multiplications, and stores leak through the power side channel, giving one leakage point per matrix entry. We restrict our attention to this first iteration because its operands are directly related to the input $M' = G_0 \cdot Q_\ell^{-1}$ and can therefore be predicted from public data and the Stage-1 output, from the second outer iteration onward, the matrix has been overwritten with linear combinations of earlier rows, and the operands of `fq_mul` inside the elimination loop are no longer simple scaled entries of G_0. Attacking those later iterations would require tracking the RREF update rule symbolically through every previous pivot, which we avoid.

Attack idea.

The inner update of the first outer iteration touches every entry of $G_0 \cdot Q_\ell^{-1}$ through a load, a multiplication, and a store, producing distinguishable power segments within a single trace, one per matrix entry. We use a neural network here in place of Gaussian templates. For this leakage target, the Gaussian template is conditioned on the exact pair $(x, y) \in \mathbb{F}_{127} \times \mathbb{F}_{127}$, where x is the known subtracted term and y is the secret minuend entry, and recovers the Hamming weight distribution by marginalising the per-value likelihoods

over all field elements sharing the same Hamming weight. The neural network nevertheless yielded higher permutation recovery accuracy on this leakage target, as quantified in Section 4.2. The network estimates the Hamming weight distribution of each entry from its corresponding trace segment. Writing $Q_\ell^{-1} = D^{(\ell)}P^{(\ell)}$, recovering $\pi^{(\ell)}$ reduces to matching each observed column of $G_0 \cdot Q_\ell^{-1}$ to its source column in $G_0 \cdot D^{(\ell)}$, whose dense part is recovered from Stage 1. We cast this as a minimum-cost bipartite assignment and solve it optimally with the Hungarian algorithm [Kuh55].

The matching cost between a source column and an observed column is the ℓ_2 distance between their Hamming weight distributions. For dense columns, the coefficient recovered in Stage 1 determines the scaled column exactly, so the target distribution is a one-hot vector at the known Hamming weight. For identity columns, the coefficient is still unknown, so we use only the coarser zero/nonzero indicator; each identity column has exactly one nonzero entry, which is the strongest signal available without that coefficient. Assembling these per-pair costs and running the Hungarian algorithm yields our estimate $\hat{\pi}^{(\ell)}$.

Swap Problem.

If the pivot is not in the first row, a row swap occurs, and the column moved into position 0 must be identified. Since the column count is small (252 for NIST Category 1), we handle this by trying each candidate column as the swapped-in column, recovering a candidate key for each, and validating against the public key to identify the true one.

Further Improving the Attack.

With this setup, one can already match the columns, but the attack can be further improved, which we now do. Looking closely at lines 8–9, as stated, the first outermost iteration performs $G[\mathsf{row}, :] - G[\mathsf{row}, 0] \cdot G[0, :]$. Since we know which column was swapped into the first column, one can find the output of $G[\mathsf{row}, 0] \cdot G[0, :]$.

Assuming the first column comes from the identity part, then after the swap, the column becomes entirely zero except for the first element, which is not multiplied. Therefore, the output of $G[\mathsf{row}, 0] \cdot G[0, :]$ becomes 0.

If the column comes from the dense part, then what we do is multiply each unknown element of a row by a known dense column; this is exactly what we did in the matrix multiplication, but with a different ordering. Therefore, we can find each element in a row, hence finding $G[\mathsf{row}, 0] \cdot G[0, :]$.

In both cases, we obtain, for every row and every column j, the exact value of $G[\mathsf{row}, 0] \cdot G[0, :]$. After finding the output of $G[\mathsf{row}, 0] \cdot G[0, :]$, we can feed the Hamming weight of this output at each specific element while guessing the element on the left-hand side of the subtraction. Since the power traces are correlated with both Hamming weight and Hamming distance, this increases the accuracy [MOW17].

3.3 Completing the Secret Key Recovery

We have now recovered the permutation matrix (P) and the dense coefficients. We complete the recovery of the secret monomial from these quantities along two independent paths.

Primary path.

The dense coefficients recovered in Stage 1 and their corresponding permutation entries recovered in Stage 2 enable us to find the rest of the secret monomial matrix. By Theorem A.4 in [HKLM25], knowing k entries of the coefficient list together with the corresponding permutations suffices to recover the full secret monomial matrix in polynomial time, provided that the dense part of the public generator matrix G_0 is full rank.

The equations works by first considering the parity-check matrix $(-G_1' \mid I_k)^T$ of G_1, which satisfies $G_1 \cdot (-G_1' \mid I_k)^T = 0$. Substituting the known public-private key relation $G_1 = RG_0Q$ into this identity and using that R is invertible, the unknown R cancels out, yielding $G_0\, Q\, (-G_1' \mid I_k)^T = 0$. This bilinear equation in Q is then linearized via the Kronecker product identity $A_1 G A_2^T = C \iff (A_1 \otimes A_2)\,\mathrm{vec}(G) = \mathrm{vec}(C)$, turning it into a linear system in the entries of $\mathrm{vec}(Q)$. Substituting in the k already-known permutation and coefficient entries and restricting the system to the remaining unknown block leaves exactly k^2 equations in k^2 unknowns (recall the full rank assumption). We therefore invoke this theorem directly to complete the primary path, and refer the interested reader to Appendix A of [HKLM25] for the proof and more details.

Secondary path.

This path uses the full permutation matrix P recovered in Stage 2 together with the dense coefficients, and derives the identity coefficients directly. Before we proceed, let us fix notation. We denote the RREF matrix as R. Writing $[I_k \mid M] \cdot D = [I_d \mid M_d]$, we note that I_d denotes the scaled identity block (a diagonal matrix whose entries are the coefficients of D corresponding to the identity columns, not the identity matrix itself) and M_d denotes the scaled dense block (the dense part M with each column scaled by the corresponding coefficient of D).

$$
\begin{aligned}
G' &= \mathrm{RREF}(G \cdot Q) && && (1)\\
&= R \cdot G \cdot Q && (\text{RREF as left-multiplication by } R) && (2)\\
&= R \cdot G \cdot D \cdot P && (Q = DP) && (3)\\
&= R \cdot \big[I_k \mid M\big] \cdot D \cdot P && (\text{Substitute systematic form of } G) && (4)\\
&= R \cdot \big[I_d \mid M_d\big] \cdot P && && (5)\\
&= \big[R \cdot I_d \mid R \cdot M_d\big] \cdot P && && (6)\\
G'\,P^{-1} &= \big[R \cdot I_d \mid R \cdot M_d\big] && && (7)
\end{aligned}
$$

For a matrix written as the horizontal concatenation of a $k \times k$ left block and a $k \times (n-k)$ right block, we denote its left and right blocks as $(\cdot)_1$ and $(\cdot)_2$, respectively. We define:

$$L \;=\; (G'P^{-1})_1, \qquad B \;=\; (G'P^{-1})_2,$$

Then, by (7), $L = RI_d$ and $B = RM_d$. Because R is invertible and I_d is diagonal with nonzero entries, the matrix $L = RI_d$ is invertible. Therefore,

$$
\begin{aligned}
L^{-1}B &= (RI_d)^{-1}RM_d && (\text{Substitute } L = RI_d,\ B = RM_d) && (8)\\
&= I_d^{-1}R^{-1}RM_d && && (9)\\
&= I_d^{-1}M_d && && (10)
\end{aligned}
$$

Since I_d is diagonal, the equation $L^{-1}B = I_d^{-1}M_d$ acts row-wise. If $I_d = \mathrm{diag}(d_0, d_1, \ldots, d_{k-1})$, then for every row i and dense-column index j,

$$(L^{-1}B)[i,j] = d_i^{-1} \cdot M_d[i,j]. \tag{11}$$

Consequently, the remaining identity coefficients are obtained by the row-wise formula

$$d_i = \frac{M_d[i,j]}{(L^{-1}B)[i,j]}, \qquad \text{for any } j \text{ with } M_d[i,j] \neq 0. \tag{12}$$

In practice, one may compute this value for all nonzero entries in row i of M_d and verify that they all agree, providing an additional consistency check on the recovered permutation and dense coefficients. The only requirement for this row-wise recovery to succeed is that every row of M_d contains at least one nonzero entry. We now show that this condition holds with overwhelming probability.

Let $m := n - k$ denote the number of dense columns, so that $M_d \in \mathbb{F}_q^{k \times m}$. The entries of the dense block M are produced uniformly over $\mathbb{F}_q$. Since each column of M_d is obtained by scaling the corresponding column of M by a nonzero element of $\mathbb{F}_q$, an entry of M_d is zero if and only if the corresponding entry of M is zero. The probability that every row of M_d contains at least one nonzero entry is $(1 - q^{-m})^k$. By the union bound, the failure probability of the row-nonzero condition is

$$\Pr\big[\exists \text{ an all-zero row in } M_d\big] \;=\; 1 - \big(1 - q^{-m}\big)^k \;\leq\; k\, q^{-m}.$$

For the NIST Category 1 parameter set ($q = 127$, $k = 126$, $m = n - k = 126$), this yields

$$1 - \big(1 - 127^{-126}\big)^{126} \;\approx\; 126 \cdot 127^{-126} \;\approx\; 10^{-263},$$

which is negligible. The same bound yields an even smaller failure probability for the Category 3 and Category 5 parameter sets, where the matrix dimensions k and n are larger. The row-wise recovery in (12) therefore succeeds except with negligible probability.

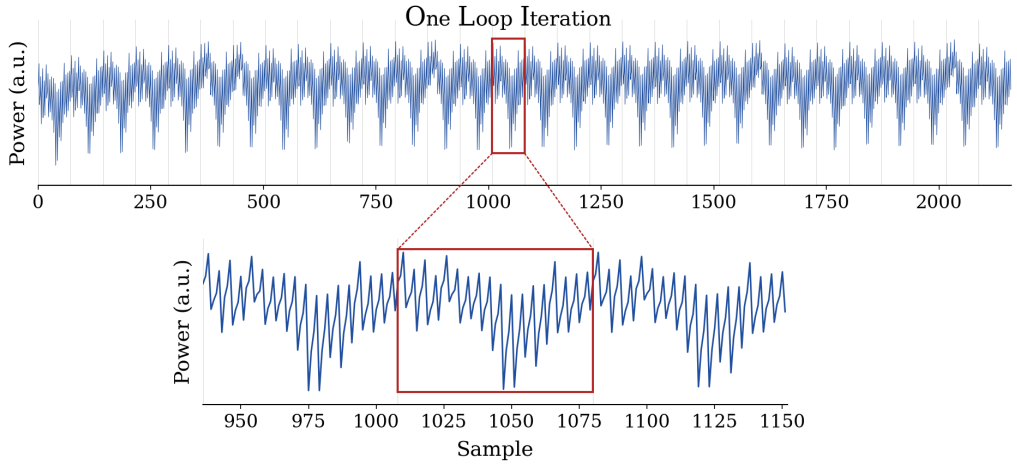
It is worth pausing to motivate why we pursue this secondary path at all. Its success condition, a nonzero entry in every row of M_d, is independent of the full-rank requirement of the primary path, so the secondary path remains viable precisely when the primary path is not. It also rests on a different algebraic relation than the primary path, making it an independent formulation of the completion step rather than a redundant one. With these operations, we conclude the attack, and the attacker has the secret monomial matrices, which are sufficient to run the signing algorithm.

4 Results

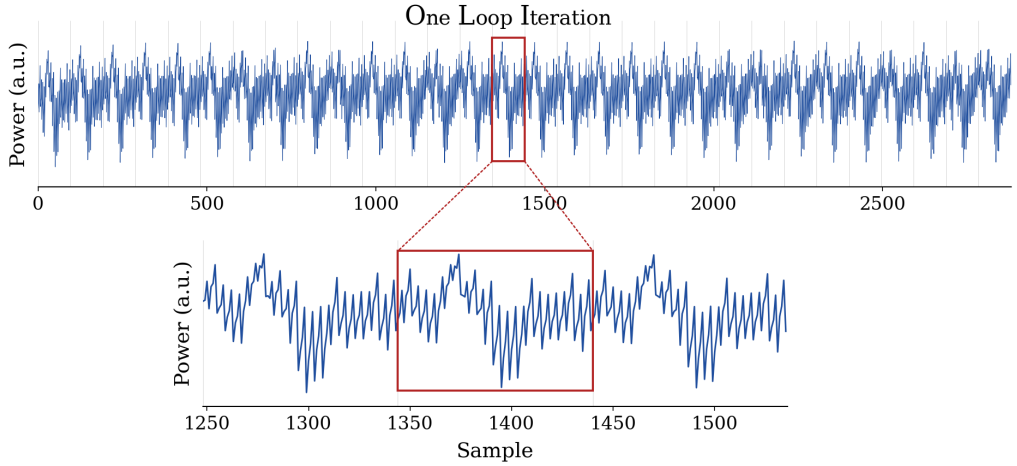
4.1 Setup

We target the official C reference implementation of LESS v2.0 from the NIST Round 2 submission, compiled with `arm-none-eabi-gcc -Os` for an `STM32F303RCT7` ARM Cortex-M4 clocked at 7.38 MHz. Power traces are captured with a ChipWhisperer-Lite (CW1173) at 29.54 MHz, yielding four samples per clock cycle. All experiments use the NIST Category 1 parameter set ($k = 126$, $n = 252$, $s = 2$). Due to the limited SRAM of the target, we isolate and port only the attacked routine, adapting `generator_monomial_mul` to operate in place. The adaptation affects only the destination of the write. In the reference implementation, the result is written to a separate output matrix, whereas in our variant, it is written to a temporary array and then copied. The `fq_mul` call issued against the secret coefficient compiles to identical instructions in both variants, and the leakage target is therefore unaffected by this change.

Additionally, following the approach of [CMR25], we also implemented only the inner loop of `generator_monomial_mul`, the column scaling, which reduces the computation to multiplying a single $k \times 1$ column by a single secret coefficient. We reproduce this setup to address any concerns about whether our in-place adaptation of the multiplication routine introduces any unintended differences in leakage behavior. Since the isolated column



(a) Generator–monomial multiplication.



(b) Row-reduced echelon form (RREF) computation.

Figure 1: Representative power traces captured during key generation, showing the part of the targeted functions that are attacked. In each figure, the highlighted region marks the targeted one-loop iteration, expanded below in detail.

scaling experiment requires no such adaptation, a consistent result between the two setups confirms that our in-place implementation does not affect the attack.

We acquired 10,000 full key-generation traces with independently drawn seeds, partitioned into 4,000 profiling and 6,000 attack traces. For the isolated column scaling experiment, we additionally acquired 20,000 traces, partitioned into 8,000 profiling and 12,000 attack traces. Figure 1 shows a representative power trace captured during key generation, focusing on the inner loop operations within the generator–monomial multiplication function and the RREF computation that are targeted in our analysis.

Classifiers.

For Stage 1 and the first part of Stage 2, we build a full-covariance Gaussian template indexed by the input pair $(x, y) \in \mathbb{F}_{127} \times \mathbb{F}_{127}^*$, where $x = G_0[i][j]$ is the known public operand and $y = c_j^{(\ell)}$ is the secret coefficient. Each inner-loop iteration produces a fixed-

Table 1: Row-normalized confusion matrices of Hamming weight predictions for Gaussian templates (left) and the neural network (right). Rows correspond to actual (A) and columns to predicted (P) Hamming weights.

(a) Gaussian templates.

A\P	0	1	2	3	4	5	6
0	0.99	0.01	0.00	0.00	0.00	0.00	0.00
1	0.02	0.65	0.30	0.04	0.00	0.00	0.00
2	0.00	0.04	0.68	0.25	0.03	0.00	0.00
3	0.00	0.00	0.09	0.72	0.18	0.01	0.00
4	0.00	0.00	0.01	0.16	0.73	0.10	0.00
5	0.00	0.00	0.00	0.02	0.28	0.66	0.04
6	0.00	0.00	0.00	0.00	0.04	0.39	0.56

(b) Neural network.

A\P	0	1	2	3	4	5	6
0	0.99	0.01	0.00	0.00	0.00	0.00	0.00
1	0.00	0.81	0.17	0.02	0.00	0.00	0.00
2	0.00	0.06	0.74	0.18	0.02	0.00	0.00
3	0.00	0.01	0.08	0.75	0.14	0.02	0.00
4	0.00	0.00	0.01	0.11	0.73	0.13	0.02
5	0.00	0.00	0.00	0.01	0.10	0.75	0.14
6	0.00	0.00	0.00	0.00	0.00	0.08	0.92

length power segment of $L = 72$ samples aligned to the loop boundary, within which the `fq_mul` call is issued. To reduce noise and covariance estimation cost, we apply ANOVA F-statistic feature selection per public value x, retaining the top $k = 40$ time samples. A pooled full-covariance matrix is then estimated for each x from the profiling traces. At inference time, per-row log-likelihoods are aggregated across all k rows of column j, and the secret coefficient is recovered by maximum-likelihood estimation.

For Stage 2, we train a small neural network in PyTorch to predict the Hamming weight of each matrix entry from its corresponding power segment. The input is the concatenation of the trace segment and a one-hot encoding of the known subtracted term Hamming weight. The network consists of two blocks, each comprising a 64-unit fully connected layer, 1D batch normalization, and ReLU activation, followed by a single linear head with 7 output units (one per possible Hamming weight in $\mathbb{F}_{127}$). The model is trained with the Adam optimizer (learning rate 10^{-3}) and cross-entropy loss. Of the 4,000 profiling traces, 3,000 are used for training and 1,000 are held out as a validation set to monitor convergence and select the final model checkpoint. At inference time, the network outputs a Hamming weight probability distribution per matrix entry, which is used as input to the Hungarian algorithm for column matching.

4.2 Attack Results

The maximum-likelihood estimator recovers the $n - k = 126$ dense coefficients with a 100% success rate, regardless of whether the isolated column scaling or the in-place matrix multiplication variant is used.

Conditional on Stage 1 having succeeded so that the dense-column costs are assembled from the true coefficients, the Hungarian assignment returns the correct $\pi^{(\ell)}$ on 0.99 of attack traces. This result breaks down into two components. The dense-column assignment succeeds on all traces, while the identity-column assignment succeeds on 0.99 of traces. This was expected, since in the dense columns the elements are well distributed; therefore, even if the predictor makes mistakes on one row, it can compensate on another. In the identity columns, however, compensating for mistakes is harder, as each column contains only a single nonzero element. For reference, replacing the neural network with Gaussian templates conditioned on exact field value pairs, as described in Section 3.2, achieved 0.98 dense-column and 0.27 identity-column permutation recovery accuracy, yielding an overall accuracy of 0.27. This confirms that the bottleneck lies in the identity-column assignment.

Table 1 shows the row-normalized confusion matrices of the Hamming weight predictions for the Gaussian templates and the neural network. One can clearly see that both models make well-calibrated decisions. When they do confuse Hamming weights, they tend to do

so with neighboring values at a distance of one, which further supports the validity of the Hamming weight leakage model.

The neural network never confuses a nonzero entry with zero. The Gaussian templates, however, exhibit a non-negligible probability of predicting Hamming weight 0 for entries that are actually nonzero, most notably for Hamming weight 1 where 0.02 of the entries are misclassified as zero. Such a misclassification is fatal for identity columns, as each contains exactly one nonzero entry and a single error directly causes the column to be mistaken for a zero column, making a correct match impossible. This explains the substantially lower identity-column permutation recovery accuracy observed with Gaussian templates compared to the neural network.

Stage 3 is algebraic and comprises two distinct attack paths: a primary attack and a secondary attack. The primary attack requires only the dense coefficients and their corresponding permutations, and these values are recovered without error; however, the algebraic path requires the dense part of the generator matrix G_0 to be full rank, and in our results, out of 6000 matrices, 53 of them were not full rank, achieving a 99.1% success rate on the NIST Category 1 parameter set. The secondary attack instead requires the full permutation rather than only the dense permutations, and these values are recovered with a 0.99 accuracy rate; since the algebraic path did not introduce any additional error rate, the secondary path achieves a 99% success rate on the NIST Category 1 parameter set.

4.3 Ablation Study: Subtracted Term Information

Recall from Section 3.2 that, after identifying the swapped-in column, we compute the exact value of the subtracted term $G[\mathsf{row}, 0] \cdot G[0, :]$ and feed its Hamming weight to the classifier as side information when predicting the Hamming weight of the unknown minuend entry $G[\mathsf{row}, :]$. Here we ablate this design choice by retraining and re-evaluating both classifiers without access to the subtracted term Hamming weight, in order to isolate how much of the Stage-2 matching accuracy is actually attributable to this side information rather than to the trace segment alone.

Without the subtracted term information, the Gaussian template achieves only 82.23% dense-column and 1.27% identity-column matching accuracy, for an overall matching accuracy of 1.13%, confirming that the template relies heavily on this conditioning to marginalize correctly. The neural network, in contrast, achieves a slightly higher identity-column matching accuracy of 99.4% without the subtracted term information than the 99% obtained with it, though the gap is small. Comparing Table 1 against Table 2 directly, subtracted term information visibly sharpens both classifiers' confusion matrices, most notably tightening the Gaussian template's off-diagonal spread and reducing the neural network's confusion between adjacent Hamming weights, even though this sharpening does not translate into a meaningfully higher end-to-end identity-column matching accuracy for the neural network. This suggests that, given the already high baseline matching accuracy and the finite size of our evaluation set, a 0.4 percentage-point difference is within the range of statistical noise rather than a meaningful effect, and the two settings should be considered practically equivalent. We nonetheless report the subtracted term informed neural network for the results in Section 4.2, for consistency with the conditioning already used in Stage 1 and in the Gaussian-template variant of Stage 2. Although we have not tested this directly, we expect the subtracted term informed neural network to achieve better matching accuracy than the uninformed version in noisier acquisition conditions, since the additional information could help the network better distinguish signal from noise.

Table 2: Row-normalized confusion matrices of Hamming weight predictions without subtracted term information, for Gaussian templates (left) and the neural network (right). Rows correspond to actual (A) and columns to predicted (P) Hamming weights.

(a) Gaussian templates.

A\P	0	1	2	3	4	5	6
0	0.99	0.00	0.00	0.00	0.00	0.00	0.00
1	0.04	0.15	0.43	0.31	0.06	0.01	0.00
2	0.01	0.04	0.34	0.45	0.14	0.03	0.00
3	0.00	0.01	0.15	0.47	0.31	0.06	0.00
4	0.00	0.00	0.06	0.30	0.49	0.14	0.01
5	0.00	0.00	0.03	0.13	0.46	0.36	0.01
6	0.00	0.00	0.01	0.07	0.28	0.55	0.07

(b) Neural network.

A\P	0	1	2	3	4	5	6
0	0.98	0.02	0.00	0.00	0.00	0.00	0.00
1	0.00	0.75	0.22	0.02	0.00	0.00	0.00
2	0.00	0.09	0.66	0.21	0.03	0.01	0.00
3	0.00	0.01	0.09	0.66	0.20	0.03	0.01
4	0.00	0.00	0.02	0.07	0.71	0.17	0.02
5	0.00	0.00	0.00	0.01	0.07	0.75	0.17
6	0.00	0.00	0.00	0.00	0.00	0.07	0.93

4.4 Scaling to NIST Category 3 and 5.

The higher parameter sets use larger matrices, but the attack structure is unchanged. In Stage 1, since each dense-column coefficient is recovered independently with 100% accuracy by aggregating leakage across all k rows of that column, increasing k (more rows) only strengthens the per-coefficient signal, while increasing $n - k$ (more coefficients to recover) adds no difficulty since the coefficients are found with 100% accuracy. We therefore expect Stage 1 to maintain 100% coefficient recovery across all parameter sets.

In Stage 2, the Hungarian assignment must match a larger number of columns, so some decrease in permutation recovery accuracy is expected as we scale to Category 3 ($k = 200$, $n = 400$) and Category 5 ($k = 274$, $n = 548$). However, we expect this degradation to be small. Since a sharp accuracy drop with column count would already be apparent at $n = 252$ columns for Category 1, yet we achieve a 99% success rate for the identity permutation and 100% for the dense permutation. The fact that accuracy remains high at this scale suggests the further increase in n is unlikely to cause a steep decline.

Stage 3 is purely algebraic, and both paths carry an additional mathematical requirement beyond the outputs of Stages 1 and 2, each of which holds with overwhelming probability regardless of the parameter set. Finally, regarding computational cost, every step of the attack, the side-channel recovery in Stages 1 and 2, the candidate search for the swapped-in column, and the linear-algebraic completion of Stage 3, has a cost that grows only polynomially in the parameter set size. The attack therefore remains entirely practical across all parameter sets, and this scaling is shared by both completion paths.

5 Countermeasures

Our results naturally raise the question of how LESS v2.0 can be protected against the attack we describe. Since the three stages of our attack are strictly sequential, each stage requiring the full output of all preceding stages, it suffices to defend Stage 1 alone to render the entire attack infeasible.

Shuffling the column order.

A natural first candidate is to randomize the order in which columns are processed inside `generator_monomial_mul`. At first glance, this appears to break Stage 1, because the attacker no longer knows which trace segment corresponds to which column of G_0. However, this countermeasure does not close the attack path. The same technique used in Stage 2 to recover the monomial permutation from the RREF function can be applied here to

recover the shuffled column order, restoring the correspondence between trace segments and columns. Column-order shuffling, therefore provides no meaningful security improvement against our attack.

This countermeasure was also proposed in [HKLM25], targeting a power side-channel attack on the same function but aimed at the permutation rather than the coefficients. Nonetheless, the same column-matching technique can be applied in their case as well, and therefore this countermeasure does not provide security against their attack either.

Shuffling the row order.

A stronger approach is to randomize the order in which the k rows are processed within each column. Notice that if a single global row permutation is applied to all columns, this does not constitute a sufficient defense either, by the same argument as column-order shuffling. The key insight is therefore to draw an independent row permutation for each column separately. Under this scheme, an attacker can no longer determine which row index is being multiplied at any given point in the trace, as the randomization order cannot be recovered. Consequently, the public matrix entry $G_0[i][j]$ is no longer available at inference time, and the Gaussian template must predict the secret coefficient without conditioning on it.

It is worth noting that although this countermeasure defends against our attack, it does not protect against the attack presented in [HKLM25]. In their attack, the targeted variable is a permutation entry that remains constant across all rows within a given column. Therefore, shuffling the row processing order leaves their targeted variable unaffected, as it depends on the column being processed rather than the row currently being processed. Consequently, an implementation that aims to be secure against both attacks must independently randomize both the column processing order and the row processing order.

Experimental evaluation.

We implemented this row-shuffling countermeasure and evaluated it under the same experimental setup as Stage 1 (same Gaussian template model, same number of captured traces). The only difference is that, since the row order is unknown at inference time, the template must be evaluated without conditioning on the public matrix entry $G_0[i][j]$. Under these conditions, the Gaussian template recovers each individual coefficient with an accuracy of only 0.27. Because all $n - k = 126$ dense coefficients must be recovered simultaneously and correctly for Stage 1 to succeed, the probability of recovering all $n - k = 126$ coefficients correctly simultaneously is $(0.27)^{126} \approx 2.24 \times 10^{-72}$, which is negligible. We therefore conclude that independently shuffling the row processing order within each column is an effective countermeasure against our attack.

Performance overhead.

We evaluated the performance overhead of the row-shuffling countermeasure on the same ARM Cortex-M4 target used in our attack experiments. As noted in Section 4.1, the full $k \times n$ matrix exceeds the available SRAM of the target device. We therefore evaluate the countermeasure using the isolated column-scaling variant, which processes one column of k elements at a time rather than loading the entire generator matrix simultaneously. This is the same setup used in the isolated column-scaling experiment of Section 4.1.

The total execution time of the protected implementation is 10,357.5 μs, representing a 33.6× overhead relative to the unprotected baseline of 327.8 μs. However, this overhead is dominated by the cost of generating random values required to produce independent row permutations via Fisher–Yates shuffling. When the random generation cost is excluded

and only the matrix multiplication itself is measured, the protected implementation takes 415.7 μs, which corresponds to a modest 1.26× overhead over the unprotected baseline. This suggests that the intrinsic cost of the shuffling logic is low, and that the dominant overhead stems from the random number generator.

We note that the security of this countermeasure depends on the row permutations being generated without observable side-channel leakage. Standard implementations of the Fisher-Yates shuffle are not constant-time and can leak the generated permutation through power analysis [VCMKS12]. A deployment-ready protected variant would therefore require a side-channel-secured shuffle-generation routine, the implementation cost of which is not reflected in our timing measurements.

6 Conclusion

We present a single-trace side-channel attack on the key generation of LESS v2.0. A single power trace of one key generation on an ARM Cortex-M4 is sufficient to recover the full secret monomial matrices for the NIST Category 1 parameter set ($k = 126$, $n = 252$); we evaluate the attack over 6000 independent keys. The attack combines side-channel leakage from the matrix multiplication and the RREF routine with a final algebraic stage that recovers the remaining secret coefficients without any additional side-channel effort. The final algebraic stage admits two independent paths: a primary path that uses only the dense coefficients and their permutation positions and succeeds on on 99.1% of the traces, and an alternative path that uses all recovered values and succeeds on 99% of the traces. We further discuss how the attack scales to the NIST Category 3 and Category 5 parameter sets in terms of attack cost and accuracy. Finally, we analyze countermeasures and show that independently shuffling the row processing order within each column of the matrix multiplication function renders the attack infeasible. Although LESS was not advanced to the third round of the NIST Additional Signature Scheme process, the methodology developed in this work may be of independent interest for the implementation security of related schemes, specifically those performing secret-dependent permutations. As future work, we plan to investigate whether the attack template developed here transfers to other code-based signature schemes.

References

[Bar86] Paul Barrett. Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 311–323. Springer, 1986.

[BBB+25a] Marco Baldi, Alessandro Barenghi, Lukas Beckwith, Tung Chou, Jean-François Biasse, Alexander Esser, Kris Gaj, Kaveh Mohajerani, Gianluca Pelosi, Edoardo Persichetti, Markku-Juhani O. Saarinen, Paolo Santini, Richard Wallace, and Felix Zweydinger. Less: Linear equivalence signature scheme (version 2.0). https://www.less-project.com/LESS-2025-02-07.pdf, February 2025. Accessed: 2026-04-18.

[BBB+25b] Marco Baldi, Alessandro Barenghi, Sebastian Bitzer, Patrick Karl, Felice Manganiello, Alessio Pavoni, Gerardo Pelosi, Paolo Santini, Jonas Schupp, Freeman Slaughter, et al. Codes and restricted objects signature scheme. *Submission to the NIST Post-Quantum Cryptography Standardization Process. Algorithm Specifications and Supporting Documentation. Version*, 2025.

[BBN+22] Alessandro Barenghi, Jean-François Biasse, Tran Ngo, Edoardo Persichetti, and Paolo Santini. Advanced signature functionalities from the code equivalence problem. *International Journal of Computer Mathematics: Computer Systems Theory*, 7(2):112–128, 2022.

[BBPS21] Alessandro Barenghi, Jean-François Biasse, Edoardo Persichetti, and Paolo Santini. Less-fm: fine-tuning signatures from the code equivalence problem. In *International Conference on Post-Quantum Cryptography*, pages 23–43. Springer, 2021.

[BBPS22] Alessandro Barenghi, Jean-François Biasse, Edoardo Persichetti, and Paolo Santini. On the computational hardness of the code equivalence problem in cryptography. *Cryptology ePrint Archive*, 2022.

[BHK+19] Daniel J Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The sphincs+ signature framework. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pages 2129–2146, 2019.

[BWMG23] Luke Beckwith, Robert Wallace, Kamyar Mohajerani, and Kris Gaj. A high-performance hardware implementation of the less digital signature scheme. In *International Conference on Post-Quantum Cryptography*, pages 57–90. Springer, 2023.

[CFG+10] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Horizontal correlation analysis on exponentiation. In *International Conference on Information and Communications Security*, pages 46–61. Springer, 2010.

[CMR25] Maciej Czuprynko, Anisha Mukherjee, and Sujoy Sinha Roy. Correlation power analysis of less and cross. In *International Conference on Cryptology in Africa*, pages 270–295. Springer, 2025.

[CPS25] Tung Chou, Edoardo Persichetti, and Paolo Santini. On linear equivalence, canonical forms, and digital signatures. *Designs, Codes and Cryptography*, 93(7):2415–2457, 2025.

[DKL+18] Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-dilithium: A lattice-based digital signature scheme. *IACR transactions on cryptographic hardware and embedded systems*, pages 238–268, 2018.

[FHK+18] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, Zhenfei Zhang, et al. Falcon: Fast-fourier lattice-based compact signatures over ntru. *Submission to the NIST's post-quantum cryptography standardization process*, 36(5):1–75, 2018.

[HKLM25] Dina Hesse, Elisabeth Krahmer, Yi-Fu Lai, and Jonas Meers. A little LESS secure - side-channel attacks exploiting randomness leakage. Cryptology ePrint Archive, Paper 2025/913, 2025.

[JMV01] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1(1):36–63, 2001.

[Kuh55] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[Moo26] Dustin Moody. *Status Report on the Second Round of the Additional Digital Signature Schemes for the NIST Post: Quantum Cryptography Standardization Process*. 2026.

[MOW17] David McCann, Elisabeth Oswald, and Carolyn Whitnall. Towards practical tools for side channel aware software engineering:'grey box'modelling for instruction leakages. In *26th USENIX security symposium (USENIX Security 17)*, pages 199–216, 2017.

[nis] PQC Digital Signature Second Round Announcement | CSRC — csrc.nist.gov. https://csrc.nist.gov/News/2024/pqc-digital-signature-second-round-announcement. [Accessed 18-04-2026].

[Nis92] Corporate Nist. The digital signature standard. *Communications of the ACM*, 35(7):36–40, 1992.

[PS23] Edoardo Persichetti and Paolo Santini. A new formulation of the linear equivalence problem and shorter less signatures. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 351–378. Springer, 2023.

[RSA78] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[Sho94] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

[Sho99] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[VCMKS12] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, pages 740–757, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.