Power side-channel leakage distinguishers on LESSv2.0

Exploiting sparse columns in Gaussian Elimination

Maciej Czuprynko[1], Rishub Nagpal[1], Tobias Schneider[2], and Sujoy Sinha Roy[1]

[1] ISEC, Graz University of Technology, Austria
{maciej.czuprynko,rishub.nagpal,sujoy.sinharoy}@tugraz.at
[2] NXP Semiconductors, Gratkorn, Austria
tobias.schneider@nxp.com

Abstract. We present the first passive side-channel distinguisher on LESSv2.0, a second-round candidate in NIST's call for additional post-quantum digital signature schemes. We target the Gaussian elimination at the core of LESS and and present a method to exploit algorithmic leakage arising from the manipulation of sparse versus dense columns.
We show that this leakage, while trivially available in non-constant-time implementations, also persists in constant-time implementations and can be exploited using a distinguisher. Concretely, the proposed attack relies only on distinguishing zero-valued computations from random ones that are repeatedly evaluated during the computation, leading to a large attack surface and making the attack robust to noise. Through simulation, we experimentally show that the required number of observed signatures lies between 300 and 2357 depending on the parameter set. This relatively large number is due to the targeted information being inherently noisy, leading to a correlation-based key recovery attack, even with noiseless leakage. Furthermore, we discuss three common countermeasures: first-order masking, shuffling and blinding.
Finally, we validate our approach on implementations with and without masking by showing the presence of leakage on a physical target.

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

1 Introduction

Many classical asymmetric cryptographic schemes derive their security from the hardness of integer factorization or discrete logarithms. Both problems are threatened by large-scale quantum computers, since Shor's algorithm [24] solves them efficiently. This has motivated sustained research in new algorithms that would resist such attacks thus introducing post-quantum cryptography. In this context, the National Institute of Standards and Technology (NIST) is in the process of standardizing post-quantum schemes. For digital signatures, three

schemes have been selected or are currently being standardized: Dilithium [13], Falcon [21], and SPHINCS+ [4]. Two of them, Dilithium and Falcon, rely on lattice-based assumptions; should progress be made against lattice problems, SPHINCS+ would remain as the only non-lattice alternative. To broaden the range of available assumptions, NIST therefore launched a separate call for additional digital signature schemes.

Among them, two code-based candidates submitted to this call advanced to the second round [18]. One is the Linear Equivalence Signature Scheme (LESS) [1]. Unlike most code-based constructions, which rely on the Syndrome Decoding Problem (SDP), LESS is built on the Linear Equivalence Problem (LEP), a classical problem in coding theory. This choice enables LESS to achieve substantially smaller signatures than CROSS, the other code-based signature scheme in the second round.

While theoretically secure, practical security also requires resistance against side-channel attacks. These types of attacks exploit unintended implementation leakage, for example through timing behavior or power consumption [14]. For SDP-based constructions, side-channel attacks have been studied, both for the KEMs considered in the NIST process [6,22,10] and for CROSS [23,11]. By contrast, LEP-based code-based signatures are more recent and have received less attention: to the best of our knowledge, the only physical attack previously studied against LESSv2.0 is the fault attack of [17]. A power-analysis attack against the first-round version of LESS was presented in [11]; however, it no longer applies to the current version, because LESSv2.0 introduces signature compression.

Our aim is to fill this knowledge gap by studying side-channel leakage in LESSv2.0. We target the Gaussian(-Jordan) elimination (GE) algorithm, which is the main computation in LESS. At a high level, all our proposed distiguishers target power consumption in order to recover information about a secret column permutation applied to a structured public matrix; the structure we exploit is the difference in power consumption when GE processes sparse columns, originating from the identity matrix, and dense columns, originating from a random matrix. We first show that non-constant-time implementations of GE leak exploitable information through a combination of timing and power consumption, which motivates the need for constant-time GE implementations [3,8] in LESS. For the attacks on constant-time implementations, the recovered information is inherently noisy in the cases we consider; the overall attack is therefore a correlation power analysis, even though the targeted information can be extracted using a distinguisher.

Contributions. Our contributions can be summarized as follows:

– **Attack on optimized constant-time GE.** We show that a constant-time GE implementation optimized for LESS [2] remains vulnerable to power-analysis attacks. The attack exploits secret-dependent sequencing introduced by the optimization, allowing an attacker to infer information about the secret permutation despite overall constant-time execution.

– **Attack on generic constant-time GE.** We present a zero-value attack that relies only on power leakage when handling columns of different types, without using any timing leakage. Furthermore, due to the large amount of leakage, we demonstrate that the targeted secret information can be recovered even in high-noise settings. Although we target the generic GE in this attack, the same attack can also be applied to the optimized GE.
– **Analysis of countermeasures.** We discuss the effectiveness of shuffling, masking [20], and blinding [3,26] as countermeasures for GE. We argue that shuffling is ineffective against our attack. Whereas, after correctly applying blinding, the GE is performed on a LEP instance. On the other hand, masking is less straightforward: using simulations, we show that although first-order masking increases the noise level as expected, the remaining leakage is still exploitable under relatively high noise.
– **Experimental validation.** We validate our attacks through simulations and show that the leakage source is present through practical experiments on a real embedded device.

2 Background

2.1 Notations

LESS performs arithmetic over the finite field $\mathbb{F}_q$, where q is prime. We write $\mathbb{F}_q^*$ for the multiplicative group $\mathbb{F}_q \setminus \{0\}$, and GL_k for the set of invertible matrices in $\mathbb{F}_q^{k\times k}$.

Matrices are denoted by bold uppercase letters, such as $\mathbf{A}$; vectors by bold lowercase letters, such as $\mathbf{a}$; and scalars by lowercase letters, such as a. For a matrix $\mathbf{A} \in \mathbb{F}_q^{k\times n}$, the entry in row i and column j is written as $\mathbf{A}[i,j]$. We denote the i-th row by $\mathbf{A}[i,:]$, or simply $\mathbf{A}[i]$, and the j-th column by $\mathbf{A}[:,j]$. For an index set $J \subseteq \{1,\dots,n\}$, the notation $\mathbf{A}_J$ denotes the submatrix formed by the columns of $\mathbf{A}$ indexed by J. Throughout the paper, indices start from 1.

When several matrices have the same role, superscripts are used to distinguish them; for example, $\mathbf{A}^{(i)}$ denotes the i-th matrix in a collection. The same convention is used for vectors and scalars. Multiplication between matrices, vectors, and scalars of compatible dimensions is written implicitly, except when "$\cdot$" is used for clarity. In all vector-matrix products, vectors are treated as row vectors. We use "$\sim$", "&", and "$\oplus$" for the bitwise NOT, AND, and XOR operations, respectively.

Permutations are denoted by Greek letters. Unless stated otherwise, their domain is $\{1,\dots,n\}$. The composition of two permutations is written with "$\circ$", so that

$$\tilde{\pi} \circ \pi(i) = \tilde{\pi}(\pi(i)).$$

For a set $J \subseteq \{1,\dots,n\}$, we define the image of J under π as

$$\pi(J) = \{\pi(i) \mid i \in J\}.$$

2.2 Linear codes

An $[n, k]$ linear code over the finite field $\mathbb{F}_q$ can be described as a vector subspace of dimension k within the space $\mathbb{F}_q^n$. Such a code may be specified by a generator matrix $\mathbf{G} \in \mathbb{F}_q^{k\times n}$ of full rank, whose rows span the code. Each element of the code, referred to as a codeword, is obtained by multiplying $\mathbf{G}$ with an information vector $\mathbf{u} \in \mathbb{F}_q^k$, yielding $\mathbf{c} = \mathbf{uG}$. This representation is not unique. Indeed, if $\mathbf{G}$ generates a code $\mathcal{C}$ and $\mathbf{S} \in \mathbb{F}_q^{k\times k}$ is an invertible matrix, then $\mathbf{SG}$ generates the same code $\mathcal{C}$, as it corresponds to a change of basis in the message space. To remove this ambiguity, it is common to fix a canonical representation. A standard choice is the systematic form, in which the generator matrix is written as $\mathbf{G} = (\mathbf{I}_k \mid \mathbf{V})$, where $\mathbf{I}_k$ denotes the $k \times k$ identity matrix and $\mathbf{V} \in \mathbb{F}_q^{k\times(n-k)}$ contains the remaining columns.

Linear equivalence. The linear equivalence problem is a special case of the code equivalence problem, which asks whether two linear codes are equivalent. In the linear equivalence setting, this relation is restricted to the existence of a monomial transformation between two $[n, k]$ linear codes. e denote the set of such transformations by M_n. Each element of M_n can be represented as an $n \times n$ matrix $\mathbf{Q} \in \mathrm{M}_n$ that decomposes as $\mathbf{Q} = \mathbf{DP}$, where $\mathbf{D}$ is an $n \times n$ diagonal matrix. The diagonal matrix can equivalently be represented by a vector $\mathbf{d}$ of non-zero elements in $\mathbb{F}_q^*$. The matrix $\mathbf{P}$ is the permutation matrix associated with a permutation π of the indices $\{1, \ldots, n\}$ and is defined by

$$\mathbf{P}[i, j] = \begin{cases} 1 & \text{if } \pi(i) = j, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

A monomial matrix is thus characterized by the pair $(\mathbf{d}, \pi)$. The underlying problem of LESS can now be stated as follows.

Definition 1 (Linear Equivalence Problem (LEP) [1]). *Given generator matrices $\mathbf{G}, \mathbf{G}' \in \mathbb{F}_q^{k\times n}$, find $\mathbf{Q} \in M_n$ and $\mathbf{S} \in GL_k$ such that $\mathbf{G}' = \mathbf{SGQ}$. Equivalently, find $\mathbf{Q} \in M_n$ such that RREF($\mathbf{G}'$) = RREF($\mathbf{GQ}$), where RREF denotes the process of reducing a matrix to its row-reduced echelon form (RREF).*

The RREF provides a unique representation of a generator matrix. It typically coincides with the systematic form, but the two differ when the first k columns of $\mathbf{G} \in \mathbb{F}_q^{k\times n}$ are not invertible. In such cases, the RREF may contain columns that do not belong to the identity matrix, intermixed with columns from the systematic form. To obtain this form in LESS, the columns are permuted so that the identity matrix lies on the left. We denote this operation by RREF^*.

Canonical forms (CF). In the second round version of LESS, the canonical forms CF are introduced to reduce signature sizes. Let $\mathbf{A} \in \mathbb{F}_q^{k\times(n-k)}$, then the canonical form is a deterministic function $\mathrm{CF} : \mathbb{F}_q^{k\times(n-k)} \to \mathbb{F}_q^{k\times(n-k)}$ such

that, for any monomial matrices $\mathbf{Q}_r \in M_k$ and $\mathbf{Q}_c \in M_{n-k}$, either the following equality holds,

$$\mathrm{CF}(\mathbf{Q}_r \mathbf{A} \mathbf{Q}_c) = \mathrm{CF}(\mathbf{A}), \tag{2}$$

or both functions return a failure. Furthermore, there exist monomial matrices $\mathbf{Q}'_r \in M_k$ and $\mathbf{Q}'_c \in M_{n-k}$ such that $\mathrm{CF}(\mathbf{A}) = \mathbf{Q}'_r \mathbf{A} \mathbf{Q}'_c$. Intuitively, the canonical form provides a unique representation up to monomial transformations applied from the right and from the left. This leads to the following problem.

Definition 2 (Canonical Forms Linear Equivalence Problem (CF-LEP) [1]). *Given generator matrices $\mathbf{G}, \mathbf{G}' \in \mathbb{F}_q^{k \times n}$, find size-$k$ sets $J, J' \subseteq \{1, \ldots, n\}$ such that:*

$$\mathrm{CF}\left(\mathbf{A}\right) = \mathrm{CF}\left(\mathbf{A}'\right) \tag{3}$$

where

$$\left(\mathbf{I}_k \mid \mathbf{A}\right) = \mathrm{RREF}^*\left(\left(\mathbf{G}_J \mid \mathbf{G}_{\{1,\ldots,n\}\setminus J}\right)\right), \text{ and} \tag{4}$$

$$\left(\mathbf{I}_k \mid \mathbf{A}'\right) = \mathrm{RREF}^*\left(\left(\mathbf{G}'_{J'} \mid \mathbf{G}'_{\{1,\ldots,n\}\setminus J'}\right)\right). \tag{5}$$

This problem is equivalent to the previously defined LEP problem as shown by the authors of [9]. An interesting observation is that the sets that form a solution to the CF-LEP satisfy the relation $\pi(J) = J'$, where π is the permutation associated to a solution $\mathbf{Q} \in M_n$ of the LEP. Furthermore, if the matrix $\mathbf{G}$ is in systematic form, then one can always find a solution where $J = \{1, \ldots, k\}$. In that case, we have that $J' = \pi(\{1, \ldots, k\})$.

2.3 LESS signature scheme

The LESS signature scheme is based on an interactive Sigma protocol represented in Figure 1, from which a signature scheme is obtained by applying the Fiat-Shamir transformation. Intuitively, in the context of the Sigma protocol, the prover shares s generator matrices $\mathbf{G}^{(i)}, \forall i \in \{1, \ldots, s\}$, in systematic form that are related through monomial transformations only known to them. To prove this knowledge, the prover commits to a generator matrix (in systematic form) generated by applying a randomly sampled monomial to $\mathbf{G}^{(1)}$. The verifier then challenges them to provide a monomial transformation $\mathbf{Q}_{\mathrm{rsp}} \in M_n$ which when applied to one of the matrices in the public key, chosen by the verifier, results in the committed matrix. In practice, because of canonical forms, it is sufficient for the prover to provide the set $J_{\mathrm{rsp}} = \pi_{\mathrm{rsp}}^{-1}(\{1, \ldots, k\})$, where π_{rsp} is the permutation associated with $\mathbf{Q}_{\mathrm{rsp}}$. Furthermore, if the prover is asked to provide a monomial to apply to $\mathbf{G}^{(1)}$, they can instead provide a seed used to generate the randomness. Note that in all the other cases, the prover needs to keep track of the permutation applied during the computation of the RREF*, which we denote μ, for the matrix recomputed by the verifier to match the committed one.

However, the soundness of the proposed scheme is only $1/s$. As such, the scheme is repeated t times to ensure soundness at the desired security level.

Moreover, to reduce the signature size, it is desirable to bias the challenge towards requesting monomials from $\mathbf{G}^{(1)}$. For our attack, the key takeaway is that w iterations have a challenge in $\mathrm{ch} \in \{2, \ldots, s\}$ while the remaining $t - w$ iterations have $\mathrm{ch} \in \{1\}$. For a more detailed description, see [1].

Public Data	System parameters $q, n, k \in \mathbb{N}, s \in \mathbb{N} \backslash \{0, 1\}, \mathbf{G}^{(1)} = (\mathbf{I}_k \mid \mathbf{V})$, $\mathbf{V} \in \mathbb{F}_q^{k \times (n-k)}, \mathbf{Q}^{(1)} = \mathbf{I}_n$, the related permutation $\pi^{(1)}$, and hash function "Hash".
Private Key	$\mathbf{Q}^{(i)} \in M_n$ and the related permutation $\pi^{(i)}, \forall i \in \{2, \ldots, s\}$,
Public Key	$\mathbf{G}^{(i)} = \mathrm{RREF}(\mathbf{G}^{(1)}\mathbf{Q}^{(i)}), \forall i \in \{2, \ldots, s\}$.

PROVER		VERIFIER
do :		
$\tilde{\mathbf{Q}} \xleftarrow{\$} \mathrm{M}_n$		
$((\mathbf{I}_k \mid \mathbf{A}), \mu) \leftarrow \mathrm{RREF}^*(\mathbf{G}^{(1)}\tilde{\mathbf{Q}})$		
while $\mathrm{CF}(\mathbf{A}) = \perp$		
$\mathrm{cmt} \leftarrow \mathrm{Hash}(\mathrm{CF}(\mathbf{A}))$	$\xrightarrow{\mathrm{cmt}}$	
	$\xleftarrow{\mathrm{ch}}$	$\mathrm{ch} \xleftarrow{\$} \{1, \ldots, s\}$
Let $\tilde{\pi}$ be the permutation related to $\tilde{\mathbf{Q}}$		
$J_{\mathrm{rsp}} \leftarrow \pi^{(\mathrm{ch})} \circ \tilde{\pi}^{-1} \circ \mu^{-1}(\{1, \ldots, k\})$	$\xrightarrow{J_{\mathrm{rsp}}}$	
		$\mathbf{B} = (\mathbf{G}^{(\mathrm{ch})}_{J_{\mathrm{rsp}}} \mid \mathbf{G}^{(\mathrm{ch})}_{\{1,\ldots,n\} \backslash J_{\mathrm{rsp}}})$
		$(\mathbf{I}_k \mid \mathbf{A}) \leftarrow \mathrm{RREF}^*(\mathbf{B})$
		Accept **if** $\mathrm{Hash}(\mathrm{CF}(\mathbf{A})) = \mathrm{cmt}$

Fig. 1: Single iteration of the Sigma protocol at the basis of LESS.

2.4 Computing the RREF

In LESS, besides hashing, the most computationally expensive step is the RREF^* procedure, which, given a matrix $\mathbf{A} \in \mathbb{F}_q^{k \times n}$ as input, returns its systematic form $(\mathbf{I}_k \mid \mathbf{V})$. This procedure internally computes the RREF using Gaussian elimination over the finite field $\mathbb{F}_q$. As shown later in this work, the structure of RREF computation is also the primary source of exploitable side-channel leakage.

In the following, we describe the implementation of the RREF. The RREF of $\mathbf{A}$ is computed column by column, starting from the leftmost or 1st column, to construct an identity submatrix. For each pivot index $i \in \{1, \ldots, k\}$, the Gaussian elimination proceeds as follows:

1. **Pivot search and row swap.** Let $i_c = i$, search for a non-zero element in the i_c-th column among the rows $\{i, \ldots, k\}$. If such an element is found at row j, then rows i and j are swapped to place the pivot at position (i, i_c). If no such element exists, repeat this step with i_c incremented by 1.

2. **Pivot normalization.** The pivot row is multiplied by the inverse of its pivot element so that the pivot becomes equal to 1. That is, $\mathbf{A}[i] = \mathbf{A}[i, i_c]^{-1}\mathbf{A}[i]$.
3. **Column elimination.** The pivot row is used to eliminate all remaining non-zero entries in the current column. For each row $j \in \{1, \ldots, k\} \setminus \{i\}$, eliminate the entry in row j by $\mathbf{A}[j] = \mathbf{A}[j] - \mathbf{A}[j, i_c]\,\mathbf{A}[i]$.

Each iteration of the Gaussian elimination constructs one column of the identity matrix. In a standard Gaussian elimination, all three steps are applied uniformly to every column.

Pivot-reuse optimization in LESS. In the Sigma protocol of LESS, as seen in Figure 1, the input to the RREF or Gaussian elimination is a generator matrix obtained by applying a monomial transformation to a matrix previously reduced to systematic form. As a result, many columns already contain exactly one non-zero entry and correspond to columns taken from the identity in the previous systematic form. These columns are referred to as *sparse columns*. In contrast, the remaining columns are referred to as *dense columns* as they contain many non-zero entries.

This observation enables pivot-reuse optimization [2]. When a column is known to be sparse and corresponds to a previous pivot, the normalization and elimination steps of Gaussian elimination are skipped. One caveat observed in [2] is that this "reuse", if performed for all sparse columns, leads to timing variations dependent on the applied ephemeral monomial. To mitigate timing leakage caused by this optimization, the implementation restricts the maximum number of reused pivots to a parameter l so that the probability of observable timing variations remains cryptographically negligible.

Column corruption. During Gaussian elimination, a sparse column may become dense and thus no longer reusable; such a column is said to be "corrupted". The corruption occurs when the pivot row contains the unique non-zero entry of a sparse column. While performing the column elimination step, the non-zero entry is multiplied by the reduced column i_c and added to the sparse column. In the case that the i_c column is dense, most added values are non-zero, which in turn makes the sparse column dense after the column elimination.

To reduce and reliably detect corruption, LESS enforces that the unique non-zero entries of reusable sparse columns lie on the diagonal. In this form, corruption can only occur after a row swap. If no swap occurs, the pivot row cannot contain such entries by construction; otherwise, the affected columns correspond exactly to the swapped row indices. This form is obtained using Algorithm 1, which reorders rows to place these entries on the diagonal.

RREF with pivot reuse. The RREF with pivot reuse is implemented following the standard Gaussian elimination explained in Section 2.4 with the modifications described in this section. The procedure is summarized in Algorithm 2.

Algorithm 1 PositionRows [2]

Input: $\mathbf{G} \in \mathbb{F}_q^{k \times n}$; $\mathbf{p} \in \{0,1\}^n$ where $\mathbf{p}[j] = 1$ indicates that column j is a pivot column from a previous RREF.
Output: $\mathbf{G}$ with the diagonal alignment: for every $j \in \{1, \ldots, k\}$ with $\mathbf{p}[j] = 1$, the unique non-zero entry of column j is at position (j, j).

for $j \leftarrow 1$ **to** k **do**
  if $\mathbf{p}[j] = 1$ **then**
    Find $r \in \{1, \ldots, k\}$ such that $\mathbf{G}[r, j] \neq 0$ ▷ unique if column j is from a pivot
    if $r \neq j$ **then**
      SwapRows($\mathbf{G}[r]$, $\mathbf{G}[j]$) ▷ move the non-zero entry of column j to (j, j)
return $\mathbf{G}$

Algorithm 2 RREFPivotReuse

Input: $\mathbf{G} \in \mathbb{F}_q^{k \times n}$; $\mathbf{p} \in \{0,1\}^n$ marks reusable (sparse) pivot columns; $l \in \mathbb{N}$ max reuse.
Output: $\mathbf{G}$ in RREF.

if $l > 0$ **then**
  $\mathbf{G} \leftarrow$ PositionRows($\mathbf{G}, \mathbf{p}$) ▷ Align reusable sparse columns on the diagonal
$rc \leftarrow 0$ ▷ Number of reused pivots
for $r = 1$ **to** k **do**
  $j \leftarrow r$
  while $j < n$ **do**
    Find $i \geq r$ such that $\mathbf{G}[i, j] \neq 0$ ▷ Pivot search in column j
    if i found **then**
      break
    $j \leftarrow j + 1$ ▷ Column j does not contain a pivot, increment j
  if $i \neq r$ **then**
    $\mathbf{p}[j] \leftarrow 0$ ▷ Row swap corrupts diagonal-aligned sparse column
    SwapRows($\mathbf{G}[r], \mathbf{G}[i]$)
  if $\mathbf{p}[j] \neq 1$ **and** $rc \geq l$ **then**
    $\mathbf{G}[r] \leftarrow \mathbf{G}[r, j]^{-1}\mathbf{G}[r]$ ▷ Normalize pivot row
    for $u = 1$ **to** k **do** ▷ Eliminate pivot column
      if $u \neq r$ **then**
        $\mathbf{G}[u] \leftarrow \mathbf{G}[u] - \mathbf{G}[u, j]\,\mathbf{G}[r]$
  else
    $rc \leftarrow rc + 1$ ▷ Reuse pivot, skip normalization and elimination
return $\mathbf{G}$

Constant-time Gaussian elimination. As previously described, Algorithm 2 is not constant-time since it requires performing a pivot search. In the context of other schemes, the works [3,8] propose a constant-time implementation of the Gaussian elimination. While in the previous works the algorithm is made constant-time as a precaution, we show in Section 3.2 that in LESS it is necessary since it leads to practical attacks. As such, we adapt the implementation in [8] to enable the reuse of the pivots. The pseudo-code is presented in Algorithm 3. In essence, to obtain a non-zero pivot on a given row, the rows below it are added until the pivot is non-zero. The adaptation needed in our case concerns how other columns are considered corrupted. In the previous case, only one element could be affected. However, in this case, if a row is added to the pivot row, the column at the same index is also corrupted.

Constant-time PositionRows. In the case of Algorithm 1, we do not provide a constant-time implementation. The reason for this choice is that we present in Section 3.2 an attack exploiting the optimization, regardless of whether such an algorithm is actually implemented.

Algorithm 3 CT-RREFPivotReuse (constant-time) [3,8]

Input: $\mathbf{G} \in \mathbb{F}_q^{k \times n}$; $\mathbf{p} \in \{0,1\}^n$ marks reusable pivot columns; $l \in \mathbb{N}$ max reuse.
Output: $\mathbf{G}$ in RREF.

if $l > 0$ **then**
  $\mathbf{G} \leftarrow$ CT-PositionRows($\mathbf{G}, \mathbf{p}$) ▷ Align reusable sparse columns on the diagonal
$rc \leftarrow 0$ ▷ Number of reused pivots
for $r = 1$ **to** k **do**
  $j \leftarrow r$
  while $j < n$ **do** ▷ Pivot search in column j
    for $i = r + 1$ **to** k **do**
      $mask \leftarrow 1$ **if** $\mathbf{G}[r, j] = 0$; **else** 0 ▷ Has to be implemented in constant-time
      $\mathbf{G}[r] \leftarrow \mathbf{G}[r] + mask \cdot \mathbf{G}[i]$
      $\mathbf{p}[i] \leftarrow mask$ & $\mathbf{p}[i]$ ▷ Row addition invalidates reuse flag
    if $\mathbf{G}[r, j] \neq 0$ **then** ▷ This check does not leak secret information
      break
    $j \leftarrow j + 1$ ▷ Column j does not contain a pivot, increment j
  if $\mathbf{p}[j] \neq 1$ **or** $rc \geq l$ **then**
    $\mathbf{G}[r] \leftarrow \mathbf{G}[r, j]^{-1}\mathbf{G}[r]$ ▷ Normalize pivot row
    for $u = 1$ **to** k **do** ▷ Eliminate pivot column
      if $u \neq r$ **then**
        $\mathbf{G}[u] \leftarrow \mathbf{G}[u] - \mathbf{G}[u, j]\mathbf{G}[r]$
  else
    $rc \leftarrow rc + 1$ ▷ Reuse pivot, skip normalization and elimination
return $\mathbf{G}$

3 Attack Description

To simplify the description of the attack, we focus on the information recovered in a single iteration of the identification protocol in Figure 1. Furthermore, we consider there to be a single long-term secret that has to be recovered, i.e., $s = 2$. In the general case, the impact of having more long-term secrets is an increase in

the number of iterations required to observe, since each secret must be recovered independently. It is worth noting that, in the case ch = 1, the long-term secret is not involved in the computation at any point of the algorithm, and thus no information can be obtained from it. Thus, for the remainder of this section, we assume the attack targets the permutation $\pi^{(2)}$, or just π, and fix the challenge as ch = 2.

In Section 3.1, we first present a sketch of the attack and we introduce the information we aim to extract via side-channels. In Section 3.2, we describe concrete leakage points in the RREF computation. Finally, in Section 3.3, we present a method to recover the long-term secret.

3.1 Sketch of the attack

Attack target. In one iteration of the identification protocol in Figure 1, the prover samples an ephemeral monomial $\tilde{\mathbf{Q}}$ with the associated permutation $\tilde{\pi}$ and computes a commitment from $\mathrm{RREF}^*(\mathbf{G}^{(1)}\tilde{\mathbf{Q}})$. Importantly, this RREF computation depends only on the ephemeral monomial $\tilde{\mathbf{Q}}$ (and thus on the permutation $\tilde{\pi}$) applied to the public matrix $\mathbf{G}^{(1)}$. Under the assumption that the challenge is always ch = 2, the prover then outputs the response set $J_{\mathrm{rsp}} = \pi \circ \tilde{\pi}^{-1} \circ \mu^{-1}(\{1, \ldots, k\})$, which becomes a public part of the transcript. In this expression, the long-term signing key π remains hidden due to the composition with the secret ephemeral permutations $\tilde{\pi}^{-1}$. On the other hand, the permutation μ is not secret since it is used only to put the output of the RREF in systematic form. For the sake of brevity, in the remaining parts of the paper, we assume that μ is the identity and we do not discuss how to take its effect into account in the general case.

Given this context, intuitively, if one recovers information about $\tilde{\pi}^{-1}$ (or about $\tilde{\pi}^{-1}(\{1, \ldots, k\})$), then one obtains information about the input of π that gives the transmitted set J_{rsp}. In turn, having information about both the input and output gives some knowledge about the long-term secret π. To extract this information, during our side-channel attacks, we target the computations performed during the RREF that leak parts of the permutation $\tilde{\pi}$, and that can be used to gain information about $\tilde{\pi}^{-1}$.

Extracted information through the SCA. By targeting $\mathrm{RREF}^*(\mathbf{G}^{(1)}\tilde{\mathbf{Q}})$, we are able to recover two types of information about the ephemeral permutation $\tilde{\pi}$:

1. Specific relations of the form $\tilde{\pi}(i) = j$ for some $i, j \in \{1, \ldots, k\}$, meaning that the i-th pivot position in the systematic form is moved to column j by the ephemeral permutation. Equivalently, this reveals that $\tilde{\pi}^{-1}(j) = i$.
   Since the public response set is $J_{\mathrm{rsp}} = \pi \circ \tilde{\pi}^{-1}(\{1, \ldots, k\})$, it follows that the secret value $\pi(i)$ must belong to J_{rsp}. As the cardinality $|J_{\mathrm{rsp}}| = k < n$, this observation restricts $\pi(i)$ from n possible positions to one of k publicly known positions, yielding a concrete constraint on the secret permutation π.
2. The set size $|\tilde{\pi}(\{1, \ldots, k\}) \cap \{1, \ldots, k\}|$, where $\cap$ is the set intersection operator. This reveals how many elements in the set $\{1, \ldots, k\}$ remain within

itself after applying the ephemeral permutation $\tilde{\pi}$. An important observation is that this count is also equal to the number of elements from the set $\{1, \ldots, k\}$ that are sent back to itself through the inverse permutation $\tilde{\pi}^{-1}$. In other words, the equality

$$|\tilde{\pi}(\{1, \ldots, k\}) \cap \{1, \ldots, k\}| = \left|\tilde{\pi}^{-1}(\{1, \ldots, k\}) \cap \{1, \ldots, k\}\right|$$

holds.
Given the public response $J_{\text{rsp}} = \pi \circ \tilde{\pi}^{-1}(\{1, \ldots, k\})$, this leakage implies that exactly $\left|\tilde{\pi}^{-1}(\{1, \ldots, k\}) \cap \{1, \ldots, k\}\right|$ elements of the set $\pi(\{1, \ldots, k\})$ lie inside the public set J_{rsp}. The attacker is thus able to learn how many secret indices $\pi(i)$, with $i \in \{1, \ldots, k\}$, are contained in the publicly revealed response set.

Recovering the secret monomial Q. In Section 3.3, we show that both leakage points let us recover the action of permutation π on the set $\{1, \ldots, k\}$, i.e. $\pi(\{1, \ldots, k\})$. As described in Section 2.2, this constitutes a solution to the CF-LEP, given the generator matrices $\mathbf{G}^{(1)}$ and $\mathbf{G}^{(2)}$ from the public key. Because the CF-LEP and LEP are equivalent, we can then use the CF-LEP solution to recover a monomial $\mathbf{Q}'$. Although $\mathbf{Q}'$ may not be identical to the original $\mathbf{Q}$, it still leads to the same public key, and thus is equivalent. The complete algorithm for this is outlined in [9, Algorithm 4].

Note on the previous attack attempt. The authors of [11] presented an attack on LESSv1.2 that also used correlation power analysis. In that work, they could predict a scaling coefficient in the ephemeral monomials from valid responses and a guess on the secret key. However, with the introduction of canonical forms in the second round of the competition (Section 2.2), it was shown that this information does not need to be transmitted in the response, thereby removing the attack vector used in the previous attack.

On the other hand, the attack proposed in this paper targets leakage that reveals information about the permutation associated with the secret ephemeral monomials used during the commitment phase.

3.2 Practical targets for the side-channel attack

Timing variations in the Gaussian elimination. During the computation of $\text{RREF}(\mathbf{G}^{(\mathbf{1})}\tilde{\mathbf{Q}})$, the potential non-constant-time behavior occurs during the search for a non-zero element in Algorithm 1 (Algorithm 1) and in Algorithm 2 (Algorithm 2). In particular, a naive implementation that scans and aborts early upon finding the first non-zero element leaks the row position $r \in \{1, \ldots, k\}$ of the non-zero element. The main idea of the attack is that, if the column is sparse, this information can be used to recover $\tilde{\pi}(r) = j$ (Case 1), where $j \in \{1, \ldots, k\}$ is the iteration number in either algorithm. Indeed, assuming that no row swaps were performed before the iteration j, the index r corresponds to the original position of the sparse column in the identity matrix, thus we learn that $\tilde{\pi}(r) = j$.

In the general case, one must keep track of row swaps to undo them and recover the original position.

As described, the timing-based SCA applies in a straightforward manner when targeting Algorithm 1 since only sparse columns are processed. In the case of Algorithm 2 and assuming no pivot is reused, one generally recovers less information because some sparse columns become corrupted (Section 2.4). On the other hand, when pivot column reuse is enabled, the non-zero elements of the sparse columns lie on the diagonal, and thus the information that can be recovered in Algorithm 2 is greatly reduced. There can still be some non-constant-time behavior, such as learning that there is a zero on the diagonal in a dense column. However, to the best of our knowledge, it is not known how to exploit this information, and we leave it as future work.

Power analysis attacks on the Gaussian elimination. In this section we describe two side-channel attacks on constant-time RREF: one attack targets the pivot-reuse optimization, while the other attack is general to any implementation of the RREF.

Exploiting leakage from pivot reuse. Although the RREF in Algorithm 3 is computed in constant time, power side-channel leakage allows an attacker to distinguish whether the condition at Step 14 of Algorithm 3 is satisfied. In the power trace, this corresponds to the presence or absence of a column reduction during an outer-loop iteration.

By observing when pivot reuse occurs, we can identify the last iteration at which a reduction is skipped. The observation provides an upper bound on the number of sparse columns processed so far. Moreover, the position of this last skip is statistically correlated with the number of dense columns appearing among the pivot positions, and thus inversely correlated with the number of sparse columns. Consequently, pivot-reuse leakage reveals information about $|\tilde{\pi}(\{1,\ldots,k\}) \cap \{1,\ldots,k\}|$, which is shown to be related to the long-term secret in Section 3.1, Case 2. Additionally, the practicality of the attack is shown in Section 5.1.

Exploiting the difference between sparse and dense columns. When the pivot-reuse optimization is disabled by setting $l = 0$ in Algorithm 3, the leakage arising from skipped reductions no longer occurs. Nevertheless, the constant-time Gaussian elimination still contains two exploitable side-channel leakages:

1. Step 9 of Algorithm 3. There, depending on the value of the variable *mask*, either a row consisting entirely of zeros is added or a row filled with at least $n - k$ random elements from $\mathbb{F}_q$[1]. These row additions are repeated until a non-zero pivot is obtained. For dense columns, the pivot is typically non-zero immediately, thus few or no nonzero additions are performed. In contrast,

[1] Around k elements are zero, which corresponds to the number of sparse columns in the original matrix.

when a sparse column is reduced, several random $\mathbb{F}_q$ additions are usually required.
2. Step 18 of Algorithm 3. For each row that does not contain the pivot, the pivot row $\mathbf{G}[r]$, scaled by $\mathbf{G}[u, j]$, is subtracted. Eliminating a sparse j-th column involves essentially only subtractions by zero, whereas eliminating a dense column involves many subtractions by random elements of $\mathbb{F}_q$.

In both cases, to distinguish a dense column from a sparse one, the same type of leakage is observed: many rows are scaled by either a zero or a non-zero element, and then added to another row. Between the two cases, the major difference is the amount of computations that leak this information. In the first case, the distinguishable behavior is observed only until a pivot is found. Moreover, after each RREF iteration, one fewer row is searched. Conversely, during column elimination, the computation is performed unconditionally for each row that does not contain the pivot. We thus always observe $(k-1)$ row operations.

Over the whole RREF computation, the information recovered is the number of sparse columns among the first k positions, which is described by Case 2 in Section 3.1. However, during the RREF computation, some sparse columns may become corrupted (see Section 2.4), causing the attacker to misclassify them and produce a noisy estimate. Despite this underestimation, secret reconstruction remains feasible as is shown in Section 3.3. Note recovering the that individual index permutations is also possible, leading to Case 1. This approach requires row-wise classification, which is less practical for side-channel analysis. Thus, we focus on the more robust leakage.

3.3 Secret recovery

In the previous sections, we showed that it is possible to recover some information about the long-term secret permutation π. However, a single iteration of the ID protocol (Figure 1) is insufficient to obtain it fully. Therefore, we must observe multiple RREF computations and aggregate the leaked information to get π. Concretely, as explained in Section 2.2, learning the action of the permutation on the set $\{1, \ldots, k\}$ is sufficient to solve the CF-LEP. Case 1 from Section 3.1, where we recover mappings of π of a single index, is straightforward. Therefore, for the remaining part of this section, we cover Case 2 from Section 3.1.

To make the recovery idea clearer, it is useful to encode the solution of $\pi(\{1, \ldots, k\})$ as a binary vector $\mathbf{x} \in \{0, 1\}^n$ of Hamming weight k defined as:

$$\mathbf{x}[j] = \begin{cases} 1 & j \in \pi(\{1, \ldots, k\}), \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

Solving for the unknown $\mathbf{x}$ is thus equivalent to recovering $\pi(\{1, \ldots, k\})$. In addition, we denote $J_{\text{rsp}}^{(i)}$ as the public response revealed in the i-th execution of the LESS ID protocol. Similarly, the response $J_{\text{rsp}}^{(i)}$ can also be encoded as a binary vector $\mathbf{r}^{(i)} \in \{0, 1\}^n$ of Hamming weight k.

Side-channel analysis of the Gaussian elimination (described in the previous section) yields a value $\hat{c}^{(i)} \leq k$, which estimates the number of sparse columns appearing in the response of the i-th round of the LESS ID protocol. This quantity represents the number of indices j such that $\mathbf{x}[j] = 1 \wedge \mathbf{r}^{(i)}[j] = 1$, i.e. $\hat{c}^{(i)} \approx c^{(i)} = \mathbf{r}^{(i)}\mathbf{x}^\top$.

In an idealized setting where the exact value $\hat{c}^{(i)} = c^{(i)}$ is available for each round, collecting such equations over m protocol executions defines the system of linear equations:

$$\mathbf{R}\mathbf{x}^\top = \mathbf{c}, \tag{7}$$

where $\mathbf{R}[i] = \mathbf{r}^{(i)}$ and $\mathbf{c}[i] = c^{(i)}$. By observing $m = n$ iterations of the ID protocol where the vectors $\mathbf{r}^{(i)}$ are linearly independent, the system can be solved for the unknown vector $\mathbf{x} \in \{0,1\}^n$. Hence, the secret image set $\pi(\{1,\ldots,k\})$ is also determined.

In practice, because of column corruption, we observe only an estimate $\hat{c}^{(i)}$, which is a lower bound on the true value $c^{(i)}$. Figure 2 illustrates this effect by comparing the distributions of the true values and the ones that can be recovered through side-channel leakage, showing that their overlap is small. This lower bound is not sufficiently restrictive to uniquely identify the correct solution using direct equation solving.

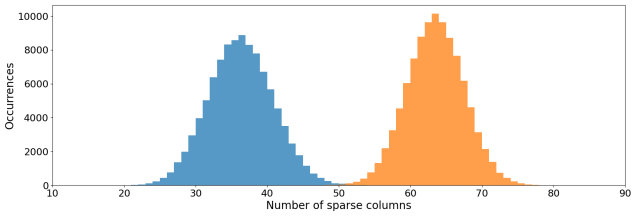


Fig. 2: Distributions of the number of sparse columns used during a Gaussian elimination over 100,000 ID protocol executions. The parameters are $n = 252, k = 126$, corresponding to NIST-I security level. The blue distribution (left) represents the information that can be observed using side-channel leakage. The orange distribution (right) represents the actual counts.

Moreover, this distribution does not account for misclassifications caused by noise during SCA. Hence, rather than attempting to solve a system of noisy equations, we exploit the statistical dependence between the public responses and the observed leakage. From Equation 7, each index j such that $\mathbf{x}[j] = 1$, contributes additively to the true count $c^{(i)}$ whenever $\mathbf{r}^{(i)}[j] = 1$. The column $\mathbf{R}[:, j]$ and the vector $\mathbf{c}$ are thus correlated. The opposite is true for j' such that $\mathbf{x}[j'] = 0$; the column $\mathbf{R}[:, j']$ is negatively correlated with $\mathbf{c}$. This is the case because the sum of sparse and dense columns is constant.

In the noisy scenario, although individual contributions may be missed in $\hat{c}^{(i)}$, the presence of a sparse column still increases the expected value of $\hat{c}^{(i)}$. To exploit this effect, we compute, for each column index j, a correlation score between the binary column $\mathbf{R}[:, j]$ and the observed leakage values $\hat{\mathbf{c}}$, where $\hat{\mathbf{c}}[i] = \hat{c}^{(i)}$. By ranking all column positions according to their correlation scores and selecting the k highest-ranked positions, we recover the positions where $\mathbf{x}[j] = 1$, and thus the image set $\pi(\{1, \ldots, k\})$ of the secret permutation. The experimental results in Section 5 confirm that this correlation-based approach reliably recovers the secret permutation in practice.

4 Countermeasures

In this section, we discuss different existing countermeasures against the proposed attack. First, we argue that our attack is not affected by shuffling. Then, we propose a blinding countermeasure based on the work of [26] and argue that the targeted leakage is not present in the GE since after blinding we end up in the LEP setting. Finally, we apply a masking countermeasure adapted from the work in [20], and we present a second-order zero-value attack on the first-order masking.

4.1 Shuffling

Shuffling is a known countermeasure that consists in hiding secret-dependent leakage by randomizing the execution of a computation. In the RREF, one can, for instance, shuffle the row access order, rendering attacks that rely on knowing their indices ineffective. This corresponds to the attacks that recover the information from Case 1 in Section 3.1. On the other hand, hiding the leakage of Case 2 in Section 3.1 is considerably more difficult. Indeed, since the attacks that recover this information do not rely on the position of the computation but on the arithmetic nature of a column, shuffling does not prevent the leakage while performing the RREF.

4.2 Blinding

An alternative idea is to use blinding by applying a fresh non-singular matrix $\mathbf{S}$ to the generator matrix $\mathbf{G} = (\mathrm{I}_k|\mathbf{V}) \in \mathbb{F}_q^{k \times n}$. This type of countermeasure was already proposed in multiple code-based schemes, for instance, in the McEliece scheme [15,3,26]. No additional changes to the computations are needed since, for any non-singular matrix $\mathbf{S} \in \mathrm{GL}_k$, the RREF satisfies:

$$\mathrm{RREF}(\mathbf{SG}) = \mathrm{RREF}(\mathbf{G}). \tag{8}$$

After the application of the matrix $\mathbf{S}$, which can be efficiently generated using the method presented in [12], the identity matrix present in the systematic form becomes randomized. Our attacks therefore become infeasible since there is no longer any distinct difference between the columns of the generator matrix.

Going further, it can also be used to protect the RREF, since after applying the monomial matrix and $\mathbf{S}$, we end up in the LEP setting. In that case, the Gaussian elimination no longer performs operations on sensitive information. However, this approach assumes that the previous steps are done securely. In particular, the non-singular matrix is prone to horizontal attack during the multiplication with the known matrix $\mathbf{G}$ which is a similar attack scenario to [23,11].

4.3 Masking

Masking Gaussian elimination has already been proposed by the authors of [20], with application to schemes that use finite fields with characteristic 2. Most of the operations can be adapted to any field in a straightforward manner. The exception is the zero check of an element. In that case, one needs to use arithmetic-to-Boolean conversions to perform the computation. To compare an arithmetically masked value with 0, we make use of the approach introduced in [7]. The description of the resulting gadget SecIsZero can be found in Algorithm 4. A masked implementation of the Gaussian elimination without pivot-reuse is provided in Algorithm 5.

Algorithm 4 SecIsZero [7,20]

Input: An arithmetic sharing $(x_i^{A_q})$ of an element $x \in \mathbb{F}_q$ and $l = \lceil \log_2 q \rceil$.
Output: An arithmetic sharing $(z_i^{A_q})$ of $z \in \mathbb{F}_q$ such that $z = 1 \mod q$ if $x = 0$ and $z = 0$ otherwise.
1: $(t_i^{B,l}) \leftarrow \text{A2B}((x_i^{A_q}))$
2: $(a_i^{B,1}) \leftarrow \text{SecNonZero}(t^{B,l})$
3: $(b_i^{B,1}) \leftarrow (\sim a_1^{B,1}, a_2^{B,1}, \ldots, a_{d+1}^{B,1})$ ▷ Equivalent to SecNot from [20]
4: $(z_i^{A_q}) \leftarrow \text{B2A}_{q-bit}((b_i^{B,1}))$
5: **return** $(z_i^{A_q})$

We use the notation (x_i) to denote a sequence of $d+1$ elements $(x_1, \ldots, x_{d+1})$. A Boolean sharing of a variable x, denoted by $(x_i^{B,l})$, is a sequence of numbers from the set $\{0, \ldots, 2^l - 1\}$ such that $x = \bigoplus_{i=1}^{d+1} x_i^{B,l}$. Similarly, an arithmetic sharing x^{A_q} is a sequence of elements from $\mathbb{F}_q$ such that $x = \sum_{i=1}^{d+1} x_i^{A_q}$. A multiplicative sharing x^{M_q} is a sequence of elements from $\mathbb{F}_q^*$ such that $x = \prod_{i=1}^{d+1} x_i^{M_q}$. This notation also extends to matrices and vectors.

Attacking the masked implementation. We adapt the attacks proposed for the unmasked version in Section 3.2 to the masked one. Let $q = 127$ be the 7-bit prime used in all instances of LESS. While first-order masking guarantees that each share is individually uniformly random, it does not prevent joint leakage from revealing low-entropy predicates. In particular, we show that the joint distribution of the Hamming weights of the two shares depends strongly on whether $x = 0$ or $x \neq 0$:

Algorithm 5 SecRREF [20]

Input: An arithmetic sharing $(\mathbf{G}_i^{A_q})$ of a matrix $\mathbf{G} \in \mathbb{F}_q^{k \times n}$
Output: An arithmetic sharing $(\mathbf{G}_i^{A_q})$ of $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ in RREF.

for $r = 1$ to k **do**
  $j \leftarrow 0$
  while $j < n$ **do** ▷ Pivot search in column j
    for $i = r + 1$ to k **do**
      $(z_i^{A_q}) \leftarrow \text{SecIsZero}((\mathbf{G}^{A_q}[r, j]_i))$
      $(\mathbf{G}^{A_q}[r]_i) \leftarrow \text{SecCondAdd}((\mathbf{G}^{A_q}[r]_i), (\mathbf{G}^{A_q}[i]_i), (z_i^{A_q}))$
    $(t_i^{A_q}) \leftarrow \text{SecIsZero}((\mathbf{G}^{A_q}[r, j]_i))$
    $\mathbf{c}[j] \leftarrow \text{FullAdd}((t_i^{A_q}))$
    if $\mathbf{c}[j] = 0$ **then**
      break
    $j \leftarrow j + 1$ ▷ Column j does not contain a pivot, increment j
  $(prod_i^{M_q}) \leftarrow \text{A2Minv}((\mathbf{G}^{A_q}[r, j]_i))$
  $(\mathbf{G}^{A_q}[r]_i) \leftarrow \text{SecScalarMult}((\mathbf{G}^{A_q}[r]_i), (prod_i^{M_q}))$ ▷ Normalize pivot row
  for $u = 1$ to k **do** ▷ Eliminate pivot column
    if $u \neq r$ **then**
      $(f_i^{A_q}) \leftarrow \text{StrongRefresh}((\mathbf{G}^{A_q}[u, j]_i))$
      $(\mathbf{G}^{A_q}[u]_i) \leftarrow \text{SecMultSub}((\mathbf{G}^{A_q}[r]_i), (\mathbf{G}^{A_q}[u]_i), (f_i^{A_q}))$
return $\mathbf{G}^{A_q}$

- If $x = 0$, then $x_1^{A_q} = -x_0^{A_q} \bmod 127$. For any $a \in \mathbb{F}_{127}$, the additive inverse satisfies $-a \equiv ((\sim a) \bmod 128) \pmod{127}$, which induces a strong correlation between the bit representations of the two shares. As a result, $\text{HW}(x_0^{A_q}) + \text{HW}(x_1^{A_q}) = 7$, except for the rare case $x_0^{A_q} = x_1^{A_q} = 0$.
- If $x \neq 0$ and is distributed uniformly at random in $\mathbb{F}_{127}$, then both shares behave approximately as independent uniformly random 7-bit values. As such, their Hamming weights follow distributions close to $B(7, 1/2)$, and their sum is well approximated by $B(14, 1/2)$.

This results in a strong statistical distinguisher between the cases $x = 0$ and $x \neq 0$. In contrast to classical higher-order DPA, the adversary does not aim to recover a high-entropy secret value, but merely to classify a single-bit predicate. This distinction is efficiently captured by computing the standard deviation of the combined leakage of the two shares, rather than their mean. The experimental results in the next section confirm that this distinguisher converges with few traces and remains effective under moderate noise. This demonstrates that first-order masking alone does not prevent leakage of zero tests in Gaussian elimination, due to the algorithmic amplification of a low-entropy predicate that is repeatedly evaluated during pivot search and elimination.

5 Practical results

In this section, we describe the practical results of the proposed attacks. For the attack that relies on averaging the power leakage, we consider only the one that uses the reduction step. We chose this one since both attacks are similar, the experimental setup is easier and the leakage is stronger. The first section

presents the simulated number of required Gaussian eliminations to successfully recover the secret key. Then, we demonstrate the presence of the leakage sources of the attacks on a microcontroller with a Cortex-M4 processor. Note that the full end-to-end attack is not performed.

Experimental setup. The side-channel target used in our experiments is an STM32F415 microcontroller setup on a CW308 UFO board. The target is running at 25MHz. The power measurement is performed using a Picoscope 6404C with a sampling frequency of 625 MSamples/s. To reduce the number of data points, a decimation by averaging of a factor of 25 is performed using the provided software development kit.

5.1 Simulation results

Exploiting pivot reuse. As described in Section 3.2, this attack relies on the ability to distinguish between the constant-time pivot search step and the column reduction step that follows to recover the index of the last reused pivot. Since both operations take considerable time and the operations performed differ, this is possible. We visualize this in Figure 3.

Given this, we performed Gaussian elimination on matrices defined by the NIST security levels and recorded the index of the last time a pivot was reused. After subtracting the recovered indices from k, we performed the key retrieval as described in Section 3.3. The number of Gaussian eliminations that need to be observed before recovering the secret key is shown in Table 1.

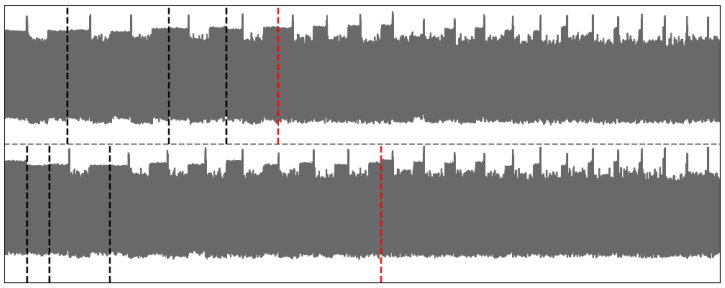


Fig. 3: Two power traces of Gaussian elimination for $(n, k) = (50, 25)$, and with a limit of 4 reused pivots. For clarity, the trace is averaged 10 times. Each pivot reuse is marked by a vertical line. The last one is highlighted in red.

Exploiting the difference between sparse and dense columns. In order to quantify the impact of noise on the success of the attack, we performed a

Security level	NIST-I	NIST-III	NIST-V
Required eliminations	7300	7700	13400

Table 1: Number of Gaussian eliminations observed before retrieving one secret.

simulation of the retrieval of the sparse column count as described in Sections 3.2 and Sections 4.3 for the unmasked and masked cases. We then computed the Pearson correlation coefficient between the retrieved counts and the correct ones.

Concretely, we assume that the noise added to the Hamming weight of the result of the multiplication is independent and follows a normal distribution centered at 0 with a standard deviation σ. As such, assuming the pivot is non-zero, the average that we compute when reducing the u-th column of the matrix $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ is:

$$\text{avg} = \sum_{i=1 \wedge i \neq u}^{k} \sum_{j=1}^{n} \frac{\text{HW}(\mathbf{G}[i,i] \cdot \mathbf{G}[i,j]) + \mathcal{N}(0,\sigma)}{(k-1) \cdot n} \tag{9}$$

Using the assumption that the noise is independent, we can simplify it to[2]:

$$\text{avg} = \sum_{i=1 \wedge i \neq u}^{k} \sum_{j=1}^{n} \frac{\text{HW}(\mathbf{G}[i,i] \cdot \mathbf{G}[i,j])}{(k-1) \cdot n} + \mathcal{N}\left(0, \frac{\sigma}{\sqrt{(k-1) \cdot n}}\right) \tag{10}$$

We compute the left-hand sum experimentally and then add noise afterward, allowing us to reuse the simulated results for different noise levels and reducing the required randomness generation. To distinguish between a reduction of a sparse column and a dense column, we try multiple thresholds (chosen between the minimum and maximum of the computed averages) to separate the obtained means. For each threshold, we then compute the sum of sparse columns in each Gaussian elimination. The best threshold is chosen by computing the correlation between the sums and the true number of sparse columns. We use the sums obtained this way to perform the secret recovery as described in Section 3.2. In practice, an attacker would try each threshold to recover the secret. However, we do not do it to save on computation time. The computed correlation and the required number of observations are reported for varying standard deviations of the added noise in Figure 4.

In the masked case, we use a similar approach, but we compute the standard deviation rather than the average. Let $(\mathbf{m}_0^{A_q}, \mathbf{m}_1^{A_q})$ be the first-order arithmetic masking of a vector containing the results of each multiplication $\mathbf{G}[i,i] \cdot \mathbf{G}[i,j]$ for $i \in \{1,\dots,k\} \backslash \{u\}$ and $j \in \{1,\dots,n\}$. Let $\mathbf{y}_0, \mathbf{y}_1$ be the noise associated to each value following the normal distribution $\mathcal{N}(0,\sigma)$. In that case, using the

[2] Note that with increasing indexes l, we gain the information that the first l columns are sparse. Thus, adding them only increases the noise. To keep the analysis simpler, we always consider every column.

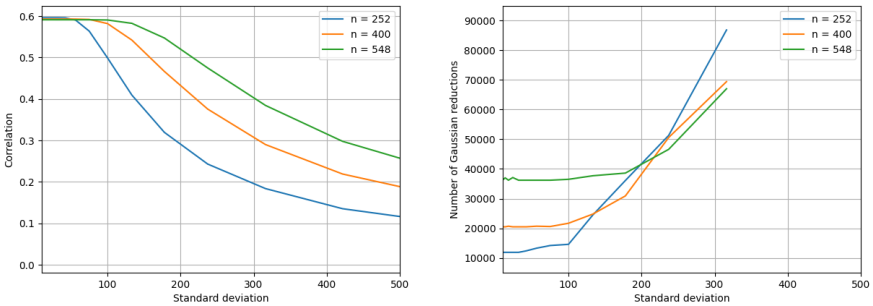


Fig. 4: Unprotected constant-time Gaussian elimination. Correlation between the recovered and the correct sums of sparse columns (left) and the required number of observed Gaussian reductions (right). The results are presented for increasing standard deviations of Gaussian noise. In each figure, the curves are for NIST security levels I, III, and V with the corresponding matrix column numbers $n =$ 252, 400, and 548. The results are an average of 10 runs with different keys.

assumption that the noise is independent of the leaked Hamming weight, we are able to write:

$$\mathrm{Var}(\mathrm{HW}(\mathbf{m}_0^{A_q}) + \mathbf{y}_0 + \mathrm{HW}(\mathbf{m}_1^{A_q}) + \mathbf{y}_1) \tag{11}$$

$$= \mathrm{Var}(\mathrm{HW}(\mathbf{m}_0^{A_q}) + \mathrm{HW}(\mathbf{m}_1^{A_q})) + \mathrm{Var}(\mathbf{y}_0 + \mathbf{y}_1) \tag{12}$$

$$= \mathrm{Var}(\mathrm{HW}(\mathbf{m}_0^{A_q}) + \mathrm{HW}(\mathbf{m}_1^{A_q})) + \frac{(\sqrt{2}\cdot\sigma)^2}{(k-1)\cdot n - 1}\chi^2((k-1)\cdot n - 1) \tag{13}$$

where, $\chi^2(k)$ is the chi-square distribution with k degrees of freedom. The standard deviation can then be obtained by taking the square root of the variance. The last equality comes from the fact that the sum of two samples from the distribution $\mathcal{N}(0, \sigma)$ follows the distribution $\mathcal{N}(0, \sqrt{2}\cdot\sigma)$. Furthermore, the variance of $(k-1)\cdot n$ samples from a normal distribution $\mathcal{N}(0, \sqrt{2}\cdot\sigma)$ has the distribution $\frac{(\sqrt{2}\cdot\sigma)^2}{(k-1)\cdot n-1}\chi^2((k-1)\cdot n - 1)$ [16, p. 418]. The rest of the procedure is the same as in the unmasked case. We report the results for the first-order masked implementation with varying noise in Figure 5.

Results interpretation. From Fig. 4 and Fig. 5, we observe that the zero-value attack performs well at relatively high noise levels, even with applied masking. The robustness of the attack to noise can be attributed to the large amount of computations that can be used to retrieve one bit of secret data, letting us decrease the noise significantly. Indeed, the averaging reduces the noise by a factor of `{177,282,387}` for NIST security levels `{I,III,V}`. The effect of the increased matrix dimensions is shown in Fig. 4 and Fig. 5, where the drop in correlation is delayed for larger matrices.

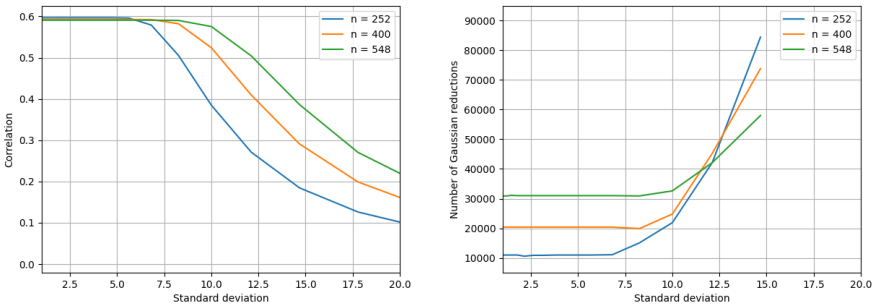


Fig. 5: Masked constant-time Gaussian elimination. Correlation between the recovered and the correct sums of sparse columns (left) and the required number of observed Gaussian reductions (right). The results are presented for increasing standard deviations of Gaussian noise. In each figure, the curves are for NIST security levels I, III, and V with the corresponding matrix column numbers 252, 400, and 548.

Note that, even in the absence of noise, a correlation of only ≈ 0.6 is observed with the correct number of sparse columns in the first k positions. This is due to the corruption of sparse columns while performing the Gaussian elimination (see Section 2.4), which introduces an inherent noise in the information that can be observed.

From the number of iterations to the number of signatures. In the previous sections, the attack difficulty is characterized by the number of Gaussian reductions that need to be observed. A more practical metric would be the number of signatures that need to be recorded. To estimate this number, there are two aspects that need to be considered: each signature contains exactly w Gaussian eliminations that can be exploited, and $s-1$ secret keys that have to be retrieved. The effect of the first parameter is to reduce the number of required signatures. We approximate this decrease by dividing by w the required number of GE to observe. On the other hand, the number of keys increases the number of required signatures. We approximate the average increase by noting that observing $s-1$ distinct groups is a generalization of the coupon-collecting problem. Let m be the required number of Gaussian eliminations that need to be observed to recover one secret key, then the asymptotic behavior of the expectation with respect to m is $m \cdot (s-1)$ [19]. The combination of both effects results in the following formula:

$$m \cdot \frac{(s-1)}{w}. \tag{14}$$

In Table 2, we use the formula to estimate the number of required signatures for both attack scenarios mentioned in this section.

NIST Cat.	Parameter Set	Code Params n	k	q	Prot. Params t	w	s	Required signatures pivot re-use	zero value
1	LESS-252-192	252	126	127	192	36	2	202.8	318.1
	LESS-252-68				68	42	4	521.4	817.9
	LESS-252-45				45	34	8	1502.9	2357.4
3	LESS-400-220	400	200	127	220	68	2	113.2	300.7
	LESS-400-102				102	61	4	378.7	1005.7
5	LESS-548-345	548	274	127	345	75	2	178.7	448.0
	LESS-548-137				137	79	4	508.9	1275.9

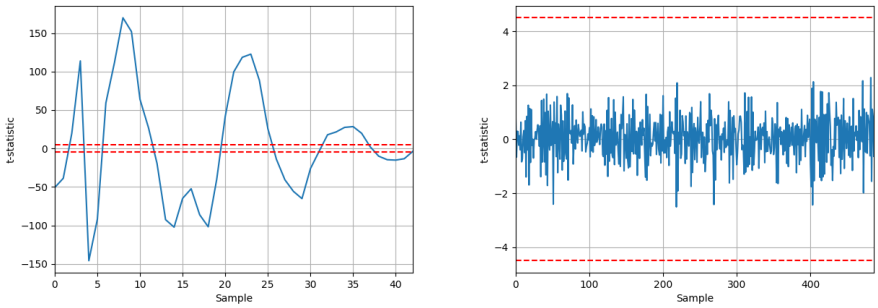
Table 2: Attack complexity estimations for all LESS parameter sets.

5.2 Experimental results

We implemented the Gaussian elimination in plain C. We based our code on the second round reference implementation of LESS[3]. However, one important modification is the generation of permutations. The reference implementation provides a biased sampling algorithm. Therefore, we use the Fisher-Yates shuffling to sample random permutations. For the masking gadgets, we implement the targeted operations in assembly to avoid transitive leakage. We used a defensive approach proposed in [5]. In particular, we focus on the SecMultSub operation, which corresponds to the reduction step of the Gaussian elimination.

Evaluating the leakage. To evaluate the leakage in the power trace, we implement a single multiplication followed by a subtraction, which corresponds to the computation that we target. To determine the points of interest (PoIs) and to validate that our implementation of the SecMultSub gadget is secure, we use Welch's first- and second-order t-tests. It is particularly suited for this scenario since the first-order test compares the difference of means between two groups, while the second-order test compares the standard deviation between two groups. If we set the first group to a multiplication by zero, with the other two values random, and a random multiplication, we end up close to our attack scenario. The first-order results are presented in Figure 6. The second-order results are presented in Figure 7 as a heat map. There, the point indexed by the x-axis is combined by addition with the point indexed on the y-axis. Given the PoIs, we select the one with the highest result. In the case of the unmasked implementation, we sort the power consumption into two groups and we use a linear normalization so that the mean of the group corresponding to a zero is also zero while the other group has a mean of 3.5 (approximately the mean of the Hamming weight of a random element of $\mathbb{F}_{127}$). The standard deviation of the noise is then computed as that of the first group. For the masked case, the power consumption is sorted by the Hamming weight of the product into 7 groups. A similar linear normalization is performed by subtracting the mean of the group with a Hamming weight of 0 and dividing by the mean of the group

[3] `https://github.com/less-sig/LESS`, commit c575939



(a) Unmasked (b) First-order masked

Fig. 6: First-order t-test results of the operation $r = c - a \cdot b$, where $a, b, c, r \in \mathbb{F}_q$ for 100,000 traces. In both groups, each variable is random except a, which is fixed to 0 in one group and to a random value in another.

with a Hamming weight of 6. There, the standard deviation is the average of the standard deviations of each group. For the unprotected implementation, the lowest recorded standard deviation is 4.5. On the other hand, for the masked implementation, the lowest value is 0.8.

Results interpretation. From the noise measurements, one can observe that the leakage in the masked implementation is less noisy than in the unmasked implementation ($\sigma = 0.8$ vs $\sigma = 4.5$). The reduced noise likely stems from implementation differences. In the masked case, the leakage increases towards the end of the computation, which corresponds to copying the multiplication result from the stack to a variable in the SecMult gadget. Conversely, in the unmasked implementation, the product remains in the registers before being subtracted from another value.

Despite this lower noise level, the masked implementation remains more secure overall. This is confirmed by the simulation results in Fig. 4 and Fig. 5, which show that the attack is harder to mount against the masked implementation at equivalent noise levels, as expected. Higher-order attacks face an additional obstacle: recombining trace points across shares might itself be non-trivial in more complex devices or with additional side-channel countermeasures.

Finally, the measured noise levels $\sigma = 0.8$ vs $\sigma = 4.5$ correspond to the region where the attack is most effective, as seen in the simulation results in Section 5.1. This suggests that distinguishing between sparse and dense columns exactly should be possible.

Feasibility estimation of the attack. Due to the considerable execution time, we do not perform the full attack. Instead, we capture 10 traces of both the masked and unmasked RREF iterations and estimate how well sparse and

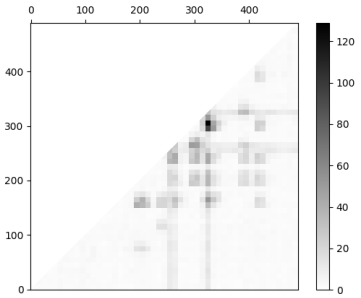


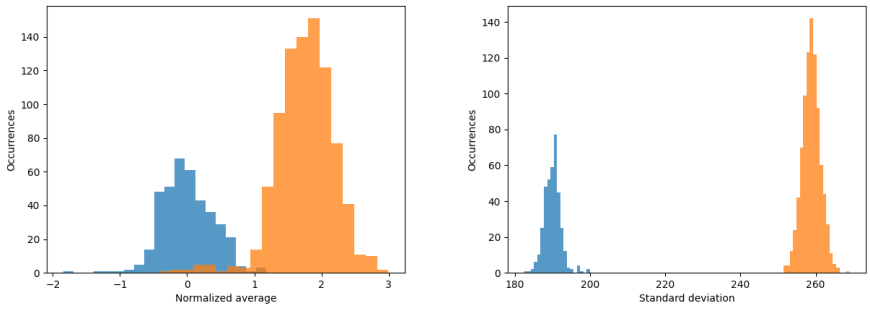
Fig. 7: Bivariate t-test results of SecMultSub. The x- and y-axis correspond to the indices of the points that are added together. A 10×10 max pooling is applied.

dense columns can be distinguished. We focus on the LESS-252 parameter set, as it represents the case with the least leakage.

The trace capture is performed separately for each of the 125 reduction steps. We first apply a high-pass filter to reduce flicker noise. The filtered traces are then split into individual operations, each consisting of a multiplication followed by a subtraction. From these operations, we compute the distribution of the average leakage in the unmasked case and the distribution of the standard deviation in the masked case, both at the best PoIs. The aggregated results are shown in Figure 8; each figure contains 1250 data points, corresponding to the 10 RREF computations.

In the masked case, the two distributions are clearly distinguishable, which corresponds to a noise with normalized standard deviation $\sigma \leq 5$. In contrast, the distributions overlap in the unmasked case. To quantify the corresponding noise standard deviation, we normalize the unmasked distributions so that the average power consumption is 0 for sparse columns and 1.75 for dense columns. Note that we use 1.75 rather than 3.5 because, during each reduction, at least half of the columns are sparse. After normalization, the standard deviation of the power consumption is 0.39 for sparse columns and 0.43 for dense columns. To estimate the noise standard deviation, we assume that the power consumption of sparse columns has standard deviation 0. Under this assumption, the observed standard deviation of sparse-column power consumption is entirely due to noise. We thus multiply 0.39 by the averaging factor $\sqrt{n \cdot (k-1)}$. We estimate that the noise present in a single computation in the graph is $\sigma = 68.6$.

Results interpretation. In the masked implementation, the sparse and dense cases produce clearly separated distributions, consistent with the observed noise level. As a result, the column structure can be accurately recovered, except for the columns that are corrupted by the Gaussian elimination. In contrast, the unmasked implementation exhibits a much higher standard deviation than



(a) Unmasked (b) Masked

Fig. 8: Distribution of the (normalized) average power consumption (left) and the distribution of the standard deviation (right) computed from selected points of interest. The values are colored based on whether they correspond to a reduction by a sparse (blue) or a dense (orange) column.

expected: 68.6 instead of 4.5. This suggests that averaging does not reduce the noise as predicted by the model. One possible explanation is that the model assumes independent noise. In practice, however, the noise may be correlated across computations, especially since these computations occur close together.

Although the noise limits the effectiveness of the simple averaging approach, the traces still contain additional exploitable leakage. As discussed in Section 5.2, multiple points in the trace leak secret information. More advanced side-channel techniques could exploit this leakage more effectively. For example, one could use template attacks, such as Linear Discriminant Analysis [25]. In the specific case of LESS, an attacker does not need a device under their control to construct the templates. Indeed, for some Gaussian elimination steps, the inputs are revealed as part of the signature. These steps can therefore be used for template building: since the attacker has access to the inputs, they know which computation is being performed.

Limitations. While the full end-to-end key recovery attack was not performed, the experimental results demonstrate that the leakage is distinguishable in both masked and unmasked cases. Specifically, sparse and dense columns can be differentiated from the traces. We therefore expect that key recovery would succeed with additional traces. Based on observed noise levels and simulation results, we expect to require approximately 13,000 and 11,000 RREF observations for the unmasked and masked cases respectively. Importantly, masked traces having clearer distinguishability compared to the unmasked ones do not indicate that the masked implementation is less secure. As discussed in Section 5.2, the reduced noise stems most likely from implementation-specific factors, not fundamental weaknesses in the masking scheme.

6 Conclusion and Future Work

In this work, we analyzed side-channel leakage in the Gaussian elimination procedure used by LESSv2.0. We showed that the distinction between sparse columns, originating from the systematic identity part of the generator matrix, and dense columns, originating from the random part, can leak information about the ephemeral permutation used during signing. Combined with public responses, this leakage can be used to recover information about the long-term secret permutation.

In constant-time implementations, power consumption can be used to recover secret information through the pivot-reuse behavior and the differences between operations involving zero values and random field elements. For the latter attack, we estimated through simulations the complexity of the attack to range from 300 to 2357 signatures, depending on the parameter set. Furthermore, due to the large leakage surface, the attack remains effective up to relatively high simulated noise levels: around $\sigma \approx 100$ in the unmasked setting and around $\sigma \approx 10$ in the masked setting.

We also evaluated representative operations on a Cortex-M4 target. The measurements confirm that the targeted zero-value leakage is present and that sparse and dense reductions can be distinguished in practical traces. However, due to the high runtime of the algorithm, we did not perform a complete end-to-end key-recovery attack on a full signer.

For future work, exploring the blinding countermeasure to avoid protecting the Gaussian elimination itself is an interesting direction to explore. In particular, comparing the cost of securely generating and applying the random non-singular matrix to masking should be explored, since the results of this work suggest that higher-order masking might be required. Furthermore, the attack implementation does not fully utilize the power traces, suggesting that the noise levels at which the attack remains effective might be higher. Finally, analyzing and protecting the full scheme are other goals to pursue, since in this work we investigate only the Gaussian elimination.

Acknowledgments. This work was partially supported through the FWF grant PAT6402023, and the State Government of Styria, Austria, through the Department Zukunftsfonds Steiermark. Further, this work was partially supported by the SINFONIA project. The SINFONIA project has received funding from the Recovery and Resilience Facility (RRF) as the centrepiece of NextGenerationEU via the Austrian Research Promotion Agency (FFG) and Austria Wirtschaftsservice Gesellschaft mbH (aws) in the frame of the IPCEI ME/CT – Important Project of Common European Interest on Microelectronic and Communication Technologies under FFG project No 917423 and AWS project No P2431566.

References

1. Baldi, M., Barenghi, A., Beckwith, L., Biasse, J.F., Esser, A., Gaj, K., Mohajerani, K., Pelosi, G., Persichetti, E., Saarinen, M.J.O., Santini, P., Wallace, R.: LESS :

Linear equivalence signature scheme, version 2.0. National Institute for Standards and Technology (2025), https://www.less-project.com/LESS-2025-02-07.pdf
2. Beckwith, L., Esser, A., Persichetti, E., Santini, P., Zweydinger, F.: LESS is even more: Optimizing digital signatures from code equivalence. Cryptology ePrint Archive, Paper 2025/1424 (2025), https://eprint.iacr.org/2025/1424
3. Bernstein, D.J., Chou, T., Schwabe, P.: McBits: fast constant-time code-based cryptography. Cryptology ePrint Archive, Paper 2015/610 (2015), https://eprint.iacr.org/2015/610
4. Bernstein, D.J., Hülsing, A., Kölbl, S., Niederhagen, R., Rijneveld, J., Schwabe, P.: The SPHINCS+ signature framework. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. p. 2129–2146. CCS '19, Association for Computing Machinery, New York, NY, USA (2019), https://doi.org/10.1145/3319535.3363229
5. Bronchain, O., Cassiers, G.: Bitslicing arithmetic/boolean masking conversions for fun and profit with application to lattice-based KEMs. Cryptology ePrint Archive, Paper 2022/158 (2022), https://eprint.iacr.org/2022/158
6. Cayrel, P.L., Fall, M., Mesmoudi, L., Gueye, C.T.: Survey on side-channel attacks on code-based key encapsulation mechanism. Moroccan Journal of Algebra and Geometry with Applications (2024)
7. Chen, K., Chen, J.: Masking floating-point number multiplication and addition of falcon first- and higher-order implementations and evaluations. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2024**(2), 276–303 (2024). https://doi.org/10.46586/TCHES.V2024.I2.276-303, https://doi.org/10.46586/tches.v2024.i2.276-303
8. Chou, T., Kannwischer, M.J., Yang, B.Y.: Rainbow on cortex-m4. Cryptology ePrint Archive, Paper 2021/532 (2021). https://doi.org/10.46586/tches.v2021.i4.650-675, https://eprint.iacr.org/2021/532
9. Chou, T., Persichetti, E., Santini, P.: On linear equivalence, canonical forms, and digital signatures. Cryptology ePrint Archive, Paper 2023/1533 (2023), https://eprint.iacr.org/2023/1533
10. Colombier, B., Grosso, V., Cayrel, P.L., Drăgoi, V.F.: Horizontal correlation attack on classic McEliece. Cryptology ePrint Archive, Paper 2023/546 (2023), https://eprint.iacr.org/2023/546
11. Czuprynko, M., Mukherjee, A., Roy, S.S.: Correlation power analysis of LESS and CROSS. Cryptology ePrint Archive, Paper 2025/839 (2025), https://eprint.iacr.org/2025/839
12. Division, C.S., Randall, D.: Efficient generation of random nonsingular matrices (Nov 1991), http://digicoll.lib.berkeley.edu/record/138424
13. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-dilithium: A lattice-based digital signature scheme. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(1), 238–268 (2018), https://doi.org/10.13154/tches.v2018.i1.238-268
14. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) Advances in Cryptology — CRYPTO' 99. pp. 388–397. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
15. McEliece, R.J.: A public-key cryptosystem based on algebraic coding theory. The Deep Space Network Progress Report42-44 (1978)
16. Mendenhall, W., Beaver, R.J., Beaver, B.M.: Introduction to Probability and Statistics. Brooks/Cole, 13 edn. (2009)

17. Mondal, P., Adhikary, S., Kundu, S., Karmakar, A.: ZKFault: Fault attack analysis on zero-knowledge based post-quantum digital signature schemes. Cryptology ePrint Archive, Paper 2024/1422 (2024), https://eprint.iacr.org/2024/1422
18. National Institute for Standards and Technology (NIST): Post-quantum cryptography: Additional digital signature schemes, https://csrc.nist.gov/projects/pqc-dig-sig/round-2-additional-signatures (Sep 2025)
19. Newman, D.J.: The double dixie cup problem. The American Mathematical Monthly **67**(1), 58–61 (1960)
20. Norga, Q., Kundu, S., Ojha, U.K., Ganguly, A., Karmakar, A., Verbauwhede, I.: Masking gaussian elimination at arbitrary order, with application to multivariate- and code-based PQC. Cryptology ePrint Archive, Paper 2024/1777 (2024), https://eprint.iacr.org/2024/1777
21. Prest, T., Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: Falcon. Tech. rep., National Institute of Standards and Technology (2022), available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions
22. Schamberger, T., Renner, J., Sigl, G., Wachter-Zeh, A.: A power side-channel attack on the CCA2-secure HQC KEM. Cryptology ePrint Archive, Paper 2020/910 (2020), https://eprint.iacr.org/2020/910
23. Schupp, J., Sigl, G.: A horizontal attack on the codes and restricted objects signature scheme (CROSS). Cryptology ePrint Archive, Paper 2025/116 (2025), https://eprint.iacr.org/2025/116
24. Shor, P.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th Annual Symposium on Foundations of Computer Science. pp. 124–134 (1994). https://doi.org/10.1109/SFCS.1994.365700
25. Standaert, F.X., Archambeau, C.: Using subspace-based template attacks to compare and combine power and electromagnetic information leakages. In: Oswald, E., Rohatgi, P. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2008. pp. 411–425. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
26. Urian, R., Schermann, R.: Classic McEliece key generation on RAM constrained devices. Cryptology ePrint Archive, Paper 2022/1613 (2022), https://eprint.iacr.org/2022/1613