# Masking Gaussian Elimination at Arbitrary Order

## with Application to Multivariate- and Code-Based PQC

Quinten Norga[1], Suparna Kundu[1], Uttam Kumar Ojha[2], Anindya Ganguly[3], Angshuman Karmakar[3], and Ingrid Verbauwhede[1]

[1] COSIC, KU Leuven, Belgium
{firstname}.{lastname}@esat.kuleuven.be
[2] Indian Statistical Institute Kolkata, India**
uttamkumarojha1729@gmail.com
[3] Indian Institute of Technology Kanpur, India
{anindyag, angshuman}@cse.iitk.ac.in

**Abstract.** Digital signature schemes based on multivariate- and code-based hard problems are promising alternatives for lattice-based signature schemes, due to their small signature size. Gaussian Elimination (GE) is a critical operation in the signing procedure of these schemes. In this paper, we provide a masking scheme for GE with back substitution to defend against first- and higher-order attacks. To the best of our knowledge, this work is the first to analyze and propose masking techniques for multivariate- or code-based DS algorithms.

We propose a masked algorithm for transforming a system of linear equations into row-echelon form. This is realized by introducing techniques for efficiently making leading (pivot) elements one while avoiding costly conversions between Boolean and multiplicative masking at all orders. We also propose a technique for efficient masked back substitution, which eventually enables a secure unmasking of the public output. All novel gadgets are proven secure in the $t$-probing model. Additionally, we evaluate the overhead of our countermeasure for several post-quantum candidates and their different security levels at first-, second-, and third-order, including UOV, MAYO, SNOVA, QR-UOV, and MQ-Sign. Notably, the operational cost of first-, second-, and third-order masked GE is 2.3× higher, and the randomness cost is 1.2× higher in MAYO compared to UOV for security levels III and V. In contrast, these costs are similar in UOV and MAYO for one version of level I. We also show detailed performance results for masked GE implementations for all three security versions of UOV on the Arm Cortex-M4 and compare them with unmasked results. Our masked implementation targeting UOV parameters has an overhead of factor 15.1×, 15.2×, and 15.4× compared to the unprotected implementation for NIST security level I, III, and V.

**Keywords:** Post-Quantum Cryptography · Masking · Gaussian Elimination · Digital Signatures · UOV

** Part of this work was completed while the author was at COSIC, KU Leuven.

## 1 Introduction

The National Institute of Standards and Technology (NIST) published the first set of Post-Quantum Cryptographic (PQC) standards in August 2024 [34,33,35]. Three of the four selected cryptographic schemes are based on hard lattice problems. To diversify their portfolio and avoid the dependency on a single hard problem, NIST announced another process [31] to standardize additional post-quantum Digital Signature (DS) schemes. The submitted schemes are designed from various hard problems, such as code-based, hash-based, and multivariate quadratic (MQ) system-based cryptography. Recently, NIST announced that 14 out of 40 initial candidates advanced to the second round [32]. Among the selected submissions, four signatures are from MQ-based cryptography and use the hash-and-sign paradigm. These schemes mainly rely on the computational hardness of solving multivariate quadratic systems, a problem known to be NP-complete [27]. The Unbalanced Oil and Vinegar (UOV) signature scheme is one of the oldest and well studied multivariate construction [28].

Gaussian Elimination (GE) is a key component of the signing procedure in many of the schemes selected for the second round of NIST PQC DS on-ramp [32], used for finding the unique solution of a system of linear equations. All MQ-based signature schemes, such as (i) UOV [8], (ii) MAYO [7], (iii) SNOVA [41], (iv) QR-UOV [20], (v) MQ-Sign [39], (vi) PROV [23], (vii) VOX [13], (viii) TUOV [19], (ix) VDOO [21], and (x) IPRainbow [9] rely on GE during signing. Recently it has also been used in some code-based (CB) signature schemes such as Wave [2]. As the secret key is used during the signing and the GE procedure, it is a potential target for side-channel attacks.

Side-Channel Analysis (SCA) attacks can have severe impacts on cryptographic implementations, and post-quantum schemes are equally vulnerable [40,26,24]. SCA attacks extract secret data of the mathematically secure cryptographic algorithm from the cryptographic device by observing the computation and its physical behavior. Such attacks can be prevented by ensuring that any computation in the cryptographic algorithm is independent of any secret variables.

Masking [10] is a provably secure widely used countermeasure of such attacks [38,18,29]. Various SCAs have been demonstrated on the UOV-based signature schemes in the literature [1,42,36]. However, there is almost no research on countermeasures for MQ and CB digital signature schemes, including UOV, to prevent potential SCAs. Even more so, no specialized gadgets for the GE operation, a critical and costly component during signing, have been proposed.

**Contributions.** We propose *first- and higher-order* masked algorithms for solving a system of linear equations using (masked) Gaussian elimination with back substitution (`SecRowEch` & `SecBackSub`), a critical component in MQ and CB signature schemes. We formally prove their security in the $t$-probing model and analyze the complexity. Our techniques and implementations are highly parametrizable and can be extended to other schemes that rely on GE to solve

a system of linear equations. We propose masked gadgets for the following sub-operations in Section 3:

- For efficiently solving the linear system, it needs to be in *row-echelon form*. We propose masked gadgets for making the pivot element non-zero, by securely adding different rows to the pivot-row if it is zero. The *conditional addition* does not reveal the pivot element itself by relying on a secure non-zero check.
- Subsequently, the pivot coefficient needs to be reduced to value one. As directly computing its inverse would require unmasking the pivot element, our approach is based on switching *masking representations* for computing its multiplicative inverse. Our approach is efficient as it exploits the multiplicative masking representation for share-wise multiplication.
- Finally, we devise an efficient masked gadget for back substituting the linear equations. We propose unmasking the final output *early* and *partitioned* to minimize the amount of masked operations. Intuitively, an output variable is unmasked once it is computed and its public representation used to compute other output variables.

We formally prove the $t$-order security of all proposed (sub-)gadgets in the $t$-probing model. We apply our techniques on several promising, UOV-based DS schemes (UOV, MAYO, SNOVA, QR-UOV & MQ-Sign) and compare their operation and randomness cost for masking the GE with back substitution, at first-, second-, and third-order. We analyze and show how their parameter choices impact the cost of masking the GE operation in Table 2 (see Section 4). We provide an arbitrary-order masked implementation (Arm M4 C code[4]) of our masked GE and evaluate the performance of our methods for UOV parameters on an Arm Cortex-M4 processor. Our first-order masked implementations of GE show an overhead of factor 15.1×, 15.2×, and 15.4× compared to the unmasked GE for UOV-I, UOV-III, and UOV-V, respectively. The most expensive steps for masking GE are ensuring the pivot element is non-zero and the full reduction to row-echelon form (see Section 5).

## 2 Preliminaries

### 2.1 Notation

We denote a finite field with $q$ elements by $\mathbb{F}_q$, with $q$ always a positive integer. $\mathbb{F}_q^*$ is used to present $\mathbb{F}_q \setminus \{0\}$. All the polynomials, vectors, and matrices are defined over $\mathbb{F}_q$ (or $\mathbb{F}_q^*$). We used lower-case letters to denote field elements/coefficients (e.g., $x$), bold lower-case letters to denote vectors (e.g., $\mathbf{b}$), and bold upper-case letters to denote matrices (e.g., $\mathbf{P}$). Please note that all the vectors are in column form and $\mathbf{P}^T$ represents the transposition of a matrix $\mathbf{P}$. The assignment of a variable is written as $:=$ and $x \leftarrow A$ denotes sampling an element uniformly

[4] `https://github.com/KULeuven-COSIC/Masking-Gaussian-Elimination`

random from set $A$ and assignment to variable $x$. All logarithms are in base 2. We denote the selection of coordinates in a vector and matrix as $\mathbf{b}_{[j]}$ and $\mathbf{P}_{[j,k]}$. The selection of a specific bit of a field element $x$ is denoted by $x^{[i]}$. A sequence of $n$ variables $(x_1, \cdots, x_n)$ (e.g., shares of variable $x$) is represented as $(x_i)_{1 \leq i \leq n}$ or in short as $(x_i)$ if the sequence length is obvious from context.

### 2.2 Gaussian Elimination

Gaussian elimination, $\boldsymbol{x} \leftarrow$ `Gaussian Elimination`$(\mathbf{A}, \mathbf{b})$, is an old technique to solve a linear system of the form $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A} \in \mathbb{F}_q^{m \times m}$ and vector $\mathbf{b} \in \mathbb{F}_q^m$ are given. Let us assume that $\mathbf{T} = [\mathbf{A} \mid \mathbf{b}]$. In Line 4 of Algorithm 1, it is attempted to make the pivot element $\mathbf{T}_{[j,j]}$ non-zero by adding the following rows if it is zero. In Line 8, it is checked if the pivot is still a non-zero pivot, and else the computation is aborted as the matrix $\mathbf{A}$ is not invertible in this case (Line 16). Subsequently, in Line 11, each current row element is multiplied with the inverse of the pivot element, making the pivot $\mathbf{T}_{[j,j]} = 1$. The loop in Line 13-14 subtracts a scalar multiple of row $j$ from all rows below the current pivot row, making the elements below the pivot zero. Finally, to find $\mathbf{x}$, Line 18 back-substitutes the variables into the system of equations.

**Algorithm 1:** `Gaussian Elimination` [8,12]

**Data:** Linear equation $\mathbf{Ax} = \mathbf{b}$, where matrix $\mathbf{A} \in \mathbb{F}_q^{m \times m}$ and vector $\mathbf{b} \in \mathbb{F}_q^m$
**Result:** Unique $\mathbf{x} \in \mathbb{F}_q^m$ such that $\mathbf{Ax} = \mathbf{b}$

```
1  T := [A | b]                                              /* T ∈ F_q^{m×(m+1)} */
2  for j = 1 upto m do
3      ## Try to make pivot T_[j,j] non-zero
4      for k = j + 1 upto m do
5          if T_[j,j] == 0 then
6              T_[j,j:m+1] = T_[j,j:m+1] + T_[k,j:m+1]
7      ## Check if pivot is non-zero
8      if T_[j,j] ≠ 0 then
9          ## Multiply row j with the inverse of its pivot
10         p := T_[j,j]^{-1}
11         T_[j,j:m+1] = p · T_[j,j:m+1]
12         ## Subtract scalar multiple of row j from the rows below
13         for k = j + 1 upto m do
14             T_[k,j:m+1] = T_[k,j:m+1] − T_[k,j] · T_[j,j:m+1]
15     else return ⊥
16 for j = m downto 2 do
17     for k = 1 upto j do
18         T_[k,m+1] = T_[k,m+1] + T_[j,k] · T_[j,m+1]
19 return x := T_[:,m+1]
```

### 2.3 Masking to Thwart SCA Attacks

Masking [10] is a well-known countermeasure against side-channel attacks. Here the sensitive variables $x$ are split into multiple, randomized shares $(x_1, \ldots, x_n)$. As a result, an attacker who does not have access to all shares $(x_i)$ does not learn anything about the secret variable $x$. The relation between $x$ and its shares $(x_1, \ldots, x_n)$ is some group operation that changes depending on the masking methods. The most utilized masking method in the literature is Boolean masking, where $x = x_1 + \cdots + x_n$ and the addition is a simple XOR ($\oplus$). In this work, we also use multiplicative masking, where $x = x_1 \otimes x_2 \cdots \otimes x_n$ and $\otimes$ denotes a multiplication between two elements of $\mathbb{F}_q$.

To argue about the security level of a masked implementation and its attackers, Ishai et al. [25] introduced the $t$-probing model. It assumes an adversary can probe up to $t$ variables during a cryptographic computation. An implementation is $t$-probing secure if any $t$ intermediate values leak no information about the unshared secret and thus can be simulated without the knowledge of this secret. In order to simplify the theoretical security analysis of a larger masked algorithm, they can be broken down into smaller functions (i.e., *gadgets*). To prove the probing security of the composition of multiple gadgets, several security notions were introduced in [3]. Below, we recall these security notions as presented in [38].

**Definition 1 ($t$-(Strong-)Non-Interference ($t$-(S)NI) security [3]).** *A gadget with one output sharing and $m_i$ input shares is t-Non-Interference (t-NI) (resp. t-Strong Non-Interference (t-SNI)) secure if any set of at most $t_1$ probes on its internal wires and $t_2$ probes on wires from its output sharings such that $t_1 + t_2 \leq t$ can be simulated with $t_1 + t_2$ (resp. $t_1$) shares of each of its $m_i$ input sharings.*

We recall two extensions of the above-mentioned security notions. Firstly, for the secure unmasking of a shared variable, we rely on the free-$t$-SNI notion, as introduced in [16]. It is a stronger notion than $t$-SNI: all output shares, except one, can be perfectly simulated (using fresh randomness). As a result, all output shares of a gadget can be simulated using the encoded (e.g. unmasked) value of the output of that gadget.

**Definition 2 (free-$t$-Strong-Non-Interference (free-$t$-SNI) security [16]).** *A gadget with output sharing $(b_i)$ and n input shares $(m_i)$ is free-t-SNI secure if for any set of at most $t_1$ probes on its internal wires such that $t_1 \leq t$, there exists a subset $I$ of input indices with $|I| \leq t_1$, such that the $t_1$ intermediate variables and the output shares $b_{|I}$ can be perfectly simulated from $m_{|I}$, while for any $O \subsetneq [1, n] \setminus I$ the output variables in $b_{|O}$ are uniformly and independently distributed, conditioned on the probed variables and $c_{|I}$.*

Secondly, we rely on the extended $t$-NIo notion from [4], which allows for public outputs. Here, certain intermediate values in an algorithm are made public and

thus are accessible to attackers. As a result, the simulator can also access the distribution of these intermediate values, to ensure successful simulation of the full gadget.

**Definition 3 ($t$-Non-Interference with public outputs ($t$-NIo) security [4]).** *A gadget with public output $b$ and $m_i$ input sharings is $t$-Non-Interference with public outputs ($t$-NIo) secure if any set of at most $t_1$ probes on its internal wires such that $t_1 \leq t$ can be simulated with $t_1$ shares of each of its $m_i$ input sharings and $b$.*

In Section 3, we formally prove our algorithms to be $t$-NI or $t$-SNI secure with $n = t + 1$ shares via simulation.

## 3 Masked Gaussian Elimination with Back Substitution

We now describe a method for solving a system of linear equations using GE and back substitution in a masked manner. Our approach is generic and can be applied at first- and higher-order. The main algorithms are the conversion of a matrix to its row-echelon form (`SecRowEch`, Algorithm 6) and the back substitution (`SecBackSub`, Algorithm 7).

First, in Sections 3.1 - 3.5, we introduce several novel masked gadgets that are used as subroutines in the main algorithms, including:

- `SecCondAdd`: conditional addition of two Boolean shared vectors (rows). This allows us to securely add two rows together, only if the pivot element of the first row is zero, without directly revealing any information about the pivot.
- `SecScalarMult`: multiplication of a (Boolean) shared vector with a multiplicative shared scalar. This allows us to multiply a row with the masked pivot of a different row, without unmasking and revealing the scalar value.
- `B2Minv`: Boolean to multiplicative inverse mask conversion, which allows us to make a pivot element one by multiplying the row with its inverse, so efficient back substitution can subsequently be performed.

All components are put together to achieve fully masked GE with back substitution in Sections 3.6 - 3.7. Table 1 gives an overview of all the used gadgets in this work, including gadgets from previous works. We also include a short description of the computed functionality and the assumed security properties.

**Methodology.** All novel gadgets are described by a $t$-order algorithm ($n = t + 1$ shares) and accompanied with a detailed description. We also prove the $t$-(S)NI security in the probing model of all algorithms/gadgets. The proofs are simulation-based: we show how probes on intermediate and output variables in the algorithms can be perfectly simulated with only a limited number of input shares. For algorithms that are composed of smaller gadgets, we rely on the $t$-(S)NI properties of the sub-gadgets. By iterating over all possible intermediate

Table 1: Overview of used gadgets in this work, with $n = t + 1$ shares.

| Algorithm | Description | Security | Reference |
|---|---|---|---|
| `Refresh` | Refresh of Boolean masking | $t$-NI | [3,6] & Alg. 8 |
| `StrongRefresh` | Strong refresh of Boolean masking | $t$-SNI | [3] & Alg. 9 |
| `FullAdd` | Refresh and combine Boolean shares | $t$-NI | [15,4] & Alg. 10 |
| `B2M` | Boolean to multiplicative mask conversion | $t$-SNI | [22,30] & Alg. 12 |
| `B2Minv` | Boolean to multiplicative inverse conversion | $t$-SNI | Algorithm 5 |
| `SecMult` | Multiplication of Boolean shares | $t$-SNI | [25,3] |
| `SecNonzero` | Nonzero check of Boolean shares | $t$-SNI | [11] & Alg.11 |
| `SecCondAdd` | Secure conditional addition | $t$-SNI | Algorithm 2 |
| `SecScalarMult` | Masked scalar multiplication | $t$-SNI | Algorithm 3 |
| `SecMultSub` | Masked multiplication and subtraction | $t$-NI | Algorithm 4 |
| `SecRowEch` | Matrix conversion to row echelon form | $t$-NIo | Algorithm 6 |
| `SecBackSub` | Masked back substitution with public output | $t$-NIo | Algorithm 7 |

(and output) variables of each sub-gadget, starting at the output and moving to the input of the algorithm, all required probes for simulation are summed. Crucially, the set of probes required from the input shares of a $t$-SNI gadget is independent from the amount of probes on its output shares.

### 3.1 Masked Conditional Addition

The conditional addition of two Boolean shared row vectors $(\mathbf{x}_i)$ and $(\mathbf{y}_i)$ is described in Algorithm 2. Depending on the condition, represented by a Boolean shared bit $(b_i)$, the result $\mathbf{s} = \mathbf{x} + \mathbf{y}$ or $\mathbf{s} = \mathbf{x}$ is computed (Line 3) and returned. For each coefficient in the vector, the shared term that is added to $(\mathbf{x}_{[j]_i})$ is computed using the `SecAND` gadget (Line 2). This gadget can be seen as a specific invocation of the `SecMult` gadget for GF(2) (bit-wise logical AND).

**Algorithm 2:** `SecCondAdd`

**Data:** 1. A Boolean sharing $(\mathbf{x}_i)$ of a (row) vector $\mathbf{x} \in \mathbb{F}_q^l$.
2. A Boolean sharing $(\mathbf{y}_i)$ of a (row) vector $\mathbf{y} \in \mathbb{F}_q^l$.
3. A Boolean sharing $(b_i)$ of a coefficient (bit) $b$.

**Result:** A Boolean sharing $(\mathbf{s}_i)$ of the vector $\mathbf{s} = \mathbf{x} + b \cdot \mathbf{y} \in \mathbb{F}_q^l$

1 **for** $j = 1$ *upto* $l$ **do**
2 $\quad(\mathbf{a}_{[j]_i}) := \texttt{SecAND}((\mathbf{y}_{[j]_i}), (b_i^{[w:1]}))$ `/* extend` $b$ `to` $w = \lceil \log(q) \rceil$ `bits */`
3 $\quad(\mathbf{s}_{[j]_i}) := (\mathbf{x}_{[j]_i} + \mathbf{a}_{[j]_i})$
4 $\quad(\mathbf{s}_{[j]_i}) = \texttt{StrongRefresh}((\mathbf{s}_{[j]_i}))$
5 **return** $(\mathbf{s}_i)$

#### 3.1.1 Complexity

Here, we discuss the run-time complexity (number of operations) and randomness complexity of the `SecCondAdd` operation. We follow the approach proposed

in [14,38]. We denote the run-time and randomness complexity of an operation `Operation` by $T_{\texttt{Operation}}$ and $R_{\texttt{Operation}}$, respectively. We also assume that the run-time cost of random number generation is unit time and operands are $w = \lceil \log(q) \rceil$ bits wide.

$$\begin{aligned} T_{\texttt{SecCondAdd}}(n,l) &= l \cdot (T_{\texttt{SecAND}}(n) + n + T_{\texttt{StrongRefresh}}(n)) \\ &= l \cdot (\frac{7n^2 - 5n}{2} + n + \frac{3n^2 - 3n}{2}) = (5n^2 - 3n)l\,, \\ R_{\texttt{SecCondAdd}}(n,l,w) &= l \cdot (R_{\texttt{SecAND}}(n,w) + 0 + R_{\texttt{StrongRefresh}}(n,w)) \\ &= l \cdot ((\frac{n^2 - n}{2} \cdot w) + (\frac{n^2 - n}{2} \cdot w)) = (n^2 - n)lw\,. \end{aligned}$$

### 3.1.2 Security

We now show Algorithm 2 to be $t$-SNI secure with $n = t+1$ shares. This means it provides resistance against an adversary with $t$ probes and allows the algorithm to be used in larger compositions.

**Lemma 1.** *The gadget* `SecCondAdd` *(Algorithm 2) is $t$-SNI secure.*

*Proof.* We first show that a single iteration $j$ is $t$-SNI secure, which is shown in an abstract diagram in Figure 1. Apart from the gadgets listed in Table 1, we model the share-wise addition on Line 3 as $t$-NI ($G_2$). An adversary can probe each gadget ($G_i$) internally or at its output. The number of internal and output probes for each gadget is denoted as $t_{G_i}$ and $o_{G_i}$, respectively. The total number of probes $t_{A_2}$ and output shares $|O|$ of (an iteration of) Algorithm 2 are:

$$t_{A_2} = \sum_{i=1}^{3} t_{G_i} + \sum_{i=1}^{2} o_{G_i}, \quad |O| = o_{G_3}\,.$$

We show that all internal and output probes can be perfectly simulated with $\leq t_{A_2}$ input shares. Firstly, to simulate the internal and output probes on gadget $G_3$, only $t_{G_3}$ shares of the output of $G_2$ are required. This is a direct result of the $t$-SNI property of $G_3$: the simulation of a $t$-SNI gadget can be performed independent of the number of probed output shares, stopping the propagation of output probes to the input. Secondly, the simulation of the probes on gadgets $G_1$-$G_2$ requires $t_{G_1} + t_{G_2} + o_{G_2}$ shares of inputs $(\mathbf{x}_i)$, $(\mathbf{y}_i)$ and $(b_i)$, as $G_2$ is $t$-NI secure. Due to the $t$-SNI property of gadget $G_1$, $t_{G_1}$ input shares are required to simulate $t_{G_1}$ intermediate probes and $o_{G_1}$ output shares. Finally, we sum up the required shares of the inputs for the simulation of all gadgets $|I|$. As $|I| = t_{G_1} + t_{G_2} + o_{G_2} + t_{G_3} \leq t_{A_2}$ and independent from $|O|$, iteration $j$ of Algorithm 2 is $t$-SNI.

Now we remark that each iteration $j$ is independent and can be executed in parallel, meaning that an adversary who places $t$ probes across all iterations can simulate them with no more number of input shares. All gadgets can thus be summarized into a single gadget (across iterations), meaning the entire loop is $t$-SNI. □

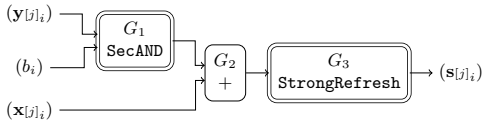


Fig. 1: An abstract diagram of an iteration $j$ in `SecCondAdd` (Algorithm 2). The $t$-NI gadgets are depicted with a single border, the $t$-SNI gadgets with a double border.

### 3.2 Masked Scalar Multiplication

The `SecScalarMult` gadget is described in Algorithm 3. The operation is used to multiply one row-vector ($\mathbf{x}$) of a matrix with a non-zero scalar value ($p$), its pivot-element. It computes the multiplication $y = p \cdot x$, with one Boolean shared operand ($\mathbf{x}_i$) and a multiplicative shared operand ($p_i$). The end-result is also Boolean shared, as is preferred for succeeding operations.

Our approach does not rely on first converting both operands to the same sharing type. The conversion of ($\mathbf{x}_i$) to the multiplicative domain (`B2M`) would allow for simple share-wise computation of the multiplication, but would require another conversion back (`M2B`). The conversion of $p_i$ to the Boolean domain (`M2B`) would result in the requirement for `SecMult` for the multiplication, which has a higher cost than `SecScalarMult`.

**Algorithm 3:** `SecScalarMult`

**Data:** 1. A Boolean sharing $(\mathbf{x}_i)$ of vector $x \in \mathbb{F}_q^l$.
2. A multiplicative sharing $(p_i)$ of a coefficient $p \in \mathbb{F}_q$.
**Result:** A Boolean sharing $(\mathbf{y}_i)$ of the vector $y = p \cdot x \in \mathbb{F}_q^l$

```
1 (y_i) := (x_i)
2 for j = 1 upto n do
3     for k = 1 upto l do
4         (y_[k]_i) = (p_j · y_[k]_i)
5         (y_[k]_i) = Refresh((y_[k]_i))
6 return (y_i)
```

We propose to perform the multiplication on one Boolean shared and multiplicative shared operand. The multiplicative shares are multiplied with all shares of the first operand, one share at a time (Line 2-5). To ensure no shares of $(p_i)$ are re-combined during the computation, a mask refreshing is used after each multiplication (Line 5).

### 3.2.1 Complexity

The run-time and randomness complexity of `SecScalarMult` are:

$$T_{\texttt{SecScalarMult}}(n,l) = n \cdot l \cdot (n + T_{\texttt{Refresh}}(n)) = (5n^2 - 3n)l\,,$$
$$R_{\texttt{SecScalarMult}}(n,l,w) = n \cdot l \cdot (0 + R_{\texttt{Refresh}}(n,w)) = (n^2 - n)lw\,.$$

### 3.2.2 Security

We now show that `SecScalarMult` is $t$-SNI secure with $n = t + 1$ shares. This means it provides resistance against an adversary with $t$ probes and allows the algorithm to be used in larger compositions.

**Lemma 2.** *The gadget* `SecScalarMult` *(Algorithm 3) is $t$-SNI secure.*

*Proof.* Before proving that the entire (outer) loop is $t$-SNI secure (Line 2-5), we show that the inner loop (Line 3-5) is $t$-NI secure, and its operations can be modeled as single $t$-NI gadgets. This follows directly from each iteration being independent and assumed to be executed in parallel (on individual coefficients). As a result, we can summarize the operations in the inner loop into single $t$-NI gadgets $G_1$ (share-wise multiplication) and $G_2$ (`Refresh`), which operate on the entire vector.

Figure 2 depicts an abstract diagram of a single iteration $j$ in Algorithm 3. Since there are $n$ iterations, an adversary cannot place probes in at least one iteration, which we refer to as $j^*$. We first consider the case where an adversary places all probes after or at the output of iteration $j = j^*$. As shown in [6], any set of output shares of `Refresh` with size $\leq n - 1$ is uniformly distributed (Lemma 1). Thus, all probes can be perfectly simulated with fresh randomness, including the outputs of the entire gadget.

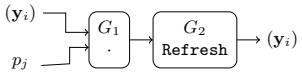


Fig. 2: An abstract diagram of an iteration $j$ in `SecScalarMult` (Algorithm 3). The $t$-NI gadgets are depicted with a single border.

We now show that when an adversary places $t$ probes before $j^*$, these can be simulated with no more number of shares of input $(\mathbf{x}_i)$ and $(p_i)$. If the share-wise multiplication ($G_1$) and/or refresh ($G_2$) is probed, all probes (across iterations) can be simulated with the same (or fewer) number of input shares of $(\mathbf{x}_i)$. Now we determine the amount of shares of $(p_i)$ required to simulate all possible probes. If all $t$ probes are placed in a single iteration $j = j^{'}$, only $p_{j^{'}}$ is required for successful simulation. If $G_2$ (`Refresh`) in successive iterations $j = j^{'} - 1$ and $j = j^{'}$ are probed, only $p_{j^{'}}$ is required for simulation. And finally, if probes

are placed in non-consecutive iterations of $G_2$, no shares of $(p_i)$ are required. The simulation is sound as the outputs of the `Refresh` gadget are uniformly random, and any such combination of probes can be perfectly simulated with fresh randomness. We now remark that the size of the required set of shares of $(p_i)$ never exceeds the number of placed probes. In conclusion, as the output of the entire loop can be perfectly simulated without any input shares and the required set of input shares of all intermediate probes is less or equal to the amount of placed probes, the entire algorithm is $t$-SNI. □

### 3.3 Masked Multiplication and Subtraction

The `SecMultSub` gadget (Algorithm 4) is used to first multiply a Boolean shared (row-)vectors $(\mathbf{x}_i)$ with a Boolean shared coefficient $(c_i)$. For this, we rely on the `SecMult` gadget (Line 2). We do not convert to the multiplicative domain, as the coefficients can be zero and thus would require handling the zero-problem as discussed in [22,30]. The result of the multiplication is subsequently subtracted from a second Boolean shared vector $(\mathbf{y}_i)$ in Line 3.

**Algorithm 4:** `SecMultSub`

**Data:** 1. Two Boolean sharings $(\mathbf{x}_i), (\mathbf{y}_i)$ of vector $x, y \in \mathbb{F}_q^l$.
2. A Boolean sharing $(c_i)$ of a coefficient $c \in \mathbb{F}_q$.
**Result:** A Boolean sharing $(\mathbf{z}_i)$ of the vector $z = x * c + y \in \mathbb{F}_q^l$

1 **for** $j = 1$ *upto* $l$ **do**
2 $\quad (\mathbf{t}_{[j]_i}) :=$ `SecMult`$((\mathbf{x}_{[j]_i}), (c_i))$ `/* t_i ∈ F_q^l */`
3 $\quad (\mathbf{z}_{[j]_i}) := (\mathbf{y}_{[j]_i} - \mathbf{t}_{[j]_i})$
4 **return** $(\mathbf{z}_i)$

#### 3.3.1 Complexity

The run-time and randomness complexity of `SecMultAdd` are:

$$T_{\texttt{SecMultAdd}}(n, l) = l \cdot (T_{\texttt{SecMult}}(n) + n) = \frac{7n^2 - 3n}{2} \cdot l\,,$$
$$R_{\texttt{SecMultAdd}}(n, l, w) = l \cdot (R_{\texttt{SecMult}}(n, w) + 0) = \frac{n^2 - n}{2} \cdot lw\,.$$

#### 3.3.2 Security

We now show that Algorithm 4 is $t$-NI secure with $n = t+1$ shares. This means it provides resistance against an adversary with $t$ probes and allows the algorithm to be used in larger compositions.

**Lemma 3.** *The gadget* `SecMultSub` *(Algorithm 4) is t-NI secure.*

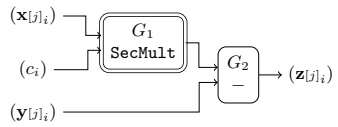


Fig. 3: An abstract diagram of an iteration $j$ in `SecMultSub` (Algorithm 4). The $t$-NI gadgets are depicted with a single border and the $t$-SNI gadgets with a double border.

*Proof.* Figure 3 depicts a single iteration $j$ in `SecMultSub`. We model the `SecMult` gadget as $G_1$ and the share-wise subtraction as $t$-NI gadget $G_2$. An adversary can probe intermediate values of both gadgets $t_{G_i}$ and the ouput of $G_1$ ($o_{G_1}$). The total number of adversary probes in an iteration of Algorithm 4 is

$$t_{A_4} = \sum_{i=1}^{2} t_{G_i} + o_{G_1} \,.$$

We now show that all probes in iteration $j$ can be simulated with no more number of shares of the inputs ($|I|$) of the iteration: $|I| \leq t_{A_4}$. If that is the case, and an adversary can place $t$ probes across different (independent) iterations, those can still be simulated with no more number of input shares. Due to the $t$-NI property of $G_2$ and $t$-SNI security of $G_1$, it follows directly that $|I| = t_{G_1} + t_{G_2}$, and Lemma 4 is proven. As $t$-SNI security allows to simulate the intermediate and output probes of a gadget with a number of input shares, independent from the amount of probed output shares. Finally, as each iteration $j$ is independent and can be executed in parallel, we can summarize the gadgets in each iteration as a single gadget across all iterations. As a result, the entire loop is $t$-NI. □

### 3.4 Boolean to Multiplicative Inverse Conversion

We now introduce a method for converting from a Boolean to a multiplicative inverse masked representation (Algorithm 5). In order to make a pivot element in a matrix one, as required for it to be in the row-echelon form, we multiply it with its inverse. As the computation of an inverse prefers multiplicative masking, the Boolean shared input ($x_i$) is first converted to a multiplicative sharing ($m_i$) (Line 1). We rely on the Boolean to multiplicative conversion proposed in [22], which is recalled in Section 3.5. Finally, in Line 2, a share-wise inversion is performed.

#### 3.4.1 Complexity

The run-time and randomness complexity of `B2Minv` are:

$$T_{\texttt{B2Minv}}(n) = \frac{5n^2 - 5n + 4}{2} \,, \quad R_{\texttt{B2Minv}}(n, w) = \frac{n^2 - n}{2} \cdot w \,.$$

**Algorithm 5:** `B2Minv`

**Data:** A Boolean sharing $(x_i)$ of a coefficient $x \in \mathbb{F}_q^*$

**Result:** A multiplicative inverse sharing $(p_i)$ such that $x^{-1} = \prod_{i=1}^{n} p_i$

1 $(m_i) :=$ `B2M`$((x_i))$
2 $(p_i) := (m_i^{-1})$ `/* multiplicative inverse */`
3 **return** $(p_i)$

### 3.4.2 Security

We now show that Algorithm 5 is $t$-SNI secure with $n = t + 1$ shares. As a result, it provides resistance against an adversary with $t$ probes and allows the algorithm to be used in larger compositions.

**Lemma 4.** *The gadget* `B2Minv` *(Algorithm 5) is $t$-SNI secure.*

*Proof.* `B2M` is a $t$-SNI Boolean to multiplicative conversion, as in [30] (Appendix A.2). The multiplicative inversion on Line 2 can be modeled as a $t$-NI gadget, as it is performed share-wise. Logically, it follows that Algorithm 5 is $t$-SNI secure. □

## 3.5 Auxiliary Gadgets

Before discussing our approach to masking the Gaussian elimination and back substitution, we first recall several auxiliary gadgets (see Appendix A). We refer to their original work for details on complexity and security.

`Refresh` & `StrongRefresh`. Both types of mask refresh gadgets are used throughout this work and recalled in Algorithm 8 & 9. Both were introduced in [3] (Algorithm 4a & 4b) and proven to be $t$-NI and $t$-SNI in [6] and [3], respectively.

`FullAdd`. We use Algorithm 10 for the secure share recombination. It consists of two steps: the *strong* mask refreshing and the share combination (unmasking). In the context of secure unmasking, a strong refresh refers to a free-$t$-SNI mask refreshing. It is shown in [17] that the `StrongRefresh` gadget satisfies the free-$t$-SNI notion. Thanks to the free-SNI notion, all outputs $(y_i)$ are simulatable if the simulator is given the unmasked value $y$. As a result, we can recombine the output shares of the gadget (Line 2) while ensuring all intermediate variables can be perfectly simulated.

In contrast, without a free-$t$-SNI gadget in Line 1, the simulation would not be sound. Placing an intermediate probe in the unmasking would require all its input shares for simulation. A $t$-NI refresh means that to simulate all of its output shares, one would require all input shares, which doesn't allow us to prove the probing security of the full circuit. The free-$t$-SNI refresh allows us to simulate all of its output shares (and all intermediate variables of the subsequent unmasking) using all but one of its inputs and the encoded value $y$, which is made public.

As shown in [18,17], the `FullAdd` gadget satisfies the $t$-NIo definition when the output $y$ is made public. Or, to prove the probing security of a composed circuit, the full gadget can be modeled as $t$-NI if the simulator has knowledge of the encoded output $y$.

`SecNonzero`**.** The gadget is recalled in Algorithm 11, as introduced in [11]. We refer to the original work for the proof of its $t$-SNI security and Appendix B for the complexity analysis. The algorithm checks if a Boolean shared operand $(x_i)$ is non-zero, if unmasked, and returns the result as a single Boolean shared bit $(b_i)$.

`B2M`**.** Finally, we also recall the `B2M` conversion as proposed in [22] (`AMtoMM`) in Algorithm 12. Its $t$-SNI security is proven in [30] (Appendix A.2). Intuitively, the algorithm sequentially replaces each Boolean share $x_i$ by a multiplicative one $(z_i)$.

### 3.6 Masked Row Echelon Conversion

A critical step in multivariate-based post-quantum signature schemes, including UOV, is solving a system of linear equations. In this (and the next) section, we propose a method for solving a Boolean shared set of linear equations $((\mathbf{T}_i) = [\mathbf{A}_i \mid \mathbf{b}_i])$ using masked Gaussian elimination. Our strategy consists of reducing a shared matrix, containing the set of equations, to its (masked) row echelon form using (`SecRowEch`). Subsequently, solving the system the system requires performing masked back substitution (`SecBackSub`).

An important step in the computation is the (repeated) checking if the matrix $\mathbf{T}$ is invertible, as this leads to a unique solution. We propose an efficient approach, which relies on verifying if pivot elements are (non-)zero in a masked manner. Note that we do not leak (unmask) the pivot element itself, but securely compute and reveal if it is zero (or not). Leaking that the matrix is not invertible is not an issue for security, as the matrix is discarded and the algorithm is re-started in this case. We now discuss the four steps of gadget `SecRowEch` in detail.

**Step 1: try to make pivot $\mathbf{T}_{[j,j]}$ non-zero.** As each pivot element (and the rest of the row) is multiplied with its inverse, in order to make it one (row echelon form), it needs to be non-zero. If the selected pivot element is zero (`SecNonzero` & `SecNOT`[5]), one of $r$ rows below is added to the 'pivot-row' $j$ using `SecCondAdd` to make it non-zero, all in the masked domain (Line 7). One needs to iterate over all rows to ensure there are no timing side-channel leakages.

**Step 2: check if pivot $\mathbf{T}_{[j,j]}$ is non-zero.** A masked non-zero check is performed on the pivot element (Line 9), resulting in a Boolean masked bit $(t_i)$. This value is securely unmasked (`FullAdd`) and made public in Line 10: if the pivot is still zero the computation is terminated $(c_j == 0)$.

[5] $(y_i) =$`SecNOT`$((x_i)) = \neg x_1 + \cdots + x_n$

**Algorithm 6:** SecRowEch

**Data:** 1. A Boolean sharing $(\mathbf{A}_i)$ of matrix $\mathbf{A} \in \mathbb{F}_q^{m \times m}$
2. A Boolean sharing $(\mathbf{b}_i)$ of the vector $\mathbf{b} \in \mathbb{F}_q^m$
**Result:** Masked conversion to row echelon form or $\perp$

```
1  (T_i) := [A_i | b_i]                                   /* T_i ∈ F_q^{m×(m+1)} */
2  for j = 1 upto m do
3      ## Try to make pivot (T_[j,j]) non-zero
4      for k = j + 1 upto m do
5          (z_i) := SecNonzero((T_[j,j]_i))
6          (z_i) = SecNOT((z_i))
7          (T_[j,j:m+1]_i) = SecCondAdd((T_[j,j:m+1]_i), (T_[k,j:m+1]_i), (z_i))
8      ## Check if pivot is non-zero
9      (t_i) := SecNonzero((T_[j,j]_i))
10     c_[j] := FullAdd((t_i))
11     if c_[j] ≠ 0 then
12         ## Multiply row j with the inverse of its pivot
13         (p_i) := B2Minv((T_[j,j]_i))
14         (T_[j,j:m+1]_i) = SecScalarMult((T_[j,j:m+1]_i), (p_i))
15         ## Subtract scalar multiple of row j from the rows below
16         for k = j + 1 upto m do
17             (s_i) := StrongRefresh((T_[k,j]_i))
18             (T_[k,j:m+1]_i) = SecMultSub((T_[j,j:m+1]_i), (T_[k,j:m+1]_i), (s_i))
19     else return ⊥
20 return ((A_i), (b_i))
```

**Step 3: make pivot $\mathbf{T}_{[j,j]} = 1$.** If the pivot element is non-zero, all elements of its row $(\mathbf{T}_{[j,:]_i})$ are multiplied with the inverse of the pivot in a masked fashion. We rely on the `B2Minv` gadget: the pivot element is first converted from its Boolean shared form to a multiplicative sharing, which allows us to compute its multiplicative inverse easily (Line 13). As a result, the `SecScalarMult` gadget operates on a Boolean shared variable and multiplicative shared scalar (Line 14). We note that we do not need to represent a zero coefficient in the multiplicative domain (zero-problem, see [22,30]) as the computation is aborted before in that case.

**Step 4: make elements below pivot $\mathbf{T}_{[j,j]}$ zero.** Finally, for each of the $k = m - j$ rows below the 'pivot-row' (Line 16-18), the element below the pivot is made zero (column $j$). Using the `SecMultSub` gadget in Line 18, the pivot-row $j$ is subtracted $\mathbf{T}_{[k,j]}$ times from each row $k$ below the pivot, in the masked domain.

### 3.6.1 Complexity

The run-time and randomness complexity of `SecRowEch` are:

$$
\begin{aligned}
T_{\texttt{SecRowEch}}(n,m) =& \frac{m^2-m}{2} \cdot (((5n^2+2n-1) + \lceil \log(w+1) \rceil \cdot (5n^2-n+2)) + 1) \\
&+ \frac{2m^3+3m^2+m}{6} \cdot (5n^2-3n) + m \cdot ((5n^2+2n-1) \\
&+ \lceil \log(w+1) \rceil \cdot (5n^2-n+2)) + m \cdot \frac{3n^2-n-2}{2} + m \\
&+ m \cdot \frac{5n^2-5n+4}{2} + \frac{m^2+3m}{2} \cdot (5n^2-3n) + \frac{m^2-m}{2} \cdot \\
&\frac{3n^2-3n}{2} + \frac{2m^3+3m^2+m}{6} \cdot \frac{7n^2-3n}{2}, \\
R_{\texttt{SecRowEch}}(n,m,w) =& \frac{m^2-m}{2} \cdot \frac{\lceil \log(w+1) \rceil^2 - \lceil \log(w+1) \rceil}{2} \cdot (n^2-n) \\
&+ \frac{2m^3+3m^2+m}{6} \cdot (n^2-n)w + m \cdot \frac{\lceil \log(w+1) \rceil^2 - \lceil \log(w+1) \rceil}{2} \cdot \\
&(n^2-n) + m \cdot \frac{(n^2-n)w}{2} + m \cdot \frac{n^2-n}{2} \cdot w + \frac{m^2+3m}{2} \cdot \\
&(n^2-n)w + \frac{m^2-m}{2} \cdot (\frac{n^2-n}{2} \cdot w) + \frac{2m^3+3m^2+m}{6} \cdot \\
&\frac{n^2-n}{2} \cdot w \,.
\end{aligned}
$$

More details on the computation are provided in Appendix C.

### 3.6.2 Security

We now argue about the first- and high-order security of Algorithm 6 by proving it to be $t$-NIo secure with $n = t + 1$ shares and public output $c$. This means it provides resistance against an adversary with $t$ probes and allows the algorithm to be used in larger compositions.

**Lemma 5.** *The gadget* `SecRowEch` *(Algorithm 6) is $t$-NIo secure with public output* $\mathbf{c}$.

*Proof.* We prove the security of Steps 1 through 4 from their composition of smaller gadgets in Appendix D. Step 1 and 3 can be modeled as $t$-SNI gadgets, while Step 2 is $t$-NIo with public output $\mathbf{c}_{[j]}$ and Step 4 is $t$-NI (Figure 4). We now prove the larger composition (Algorithm 6) to be $t$-NIo. An abstract diagram of an iteration $j$ is shown in Figure 4, constructed from Step 1 - 4. We now show that a single (outer) loop is $t$-NI secure if the value $\mathbf{c}_{[j]}$ is given to the simulator. This is a direct result of each step achieving at least $t$-NI security and being composed as in Figure 4 and all public outputs being securely recombined. This means all probes in a single iteration $j$ can be simulated with no more number of shares of $(\mathbf{T}_i)$. It is clear that if an adversary can place $t$ probes across different iterations, these can also be simulated with no more number of input shares if $\mathbf{c}$ is given to the simulator. As a result, the entire gadget `SecRowEch` is $t$-NIo secure when $\mathbf{c}$ is public. □

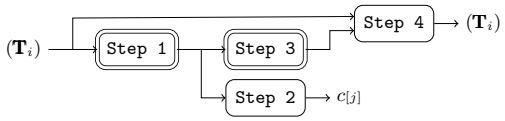


Fig. 4: An abstract diagram of a single iteration $j$ of `SecRowEch` (Algorithm 6). The $t$-NI gadgets are depicted with a single border and the $t$-SNI gadgets with a double border.

### 3.7 Masked Back Substitution with Public Output

The second and final step in solving a system of linear equations ($\mathbf{Ax} = \mathbf{b}$), is performing back substitution on a system (e.g., matrix) in row-echelon form ($[\mathbf{A}_i \mid \mathbf{b}_i]$) and is Boolean shared. We propose to compute the unique solution $\mathbf{x}$ as a public output, with a secure recombination of the shares.

**Algorithm 7:** `SecBackSub`

**Data:** 1. A Boolean sharing $(\mathbf{A}_i)$ of matrix $\mathbf{A} \in \mathbb{F}_q^{m \times m}$
2. A Boolean sharing $(\mathbf{b}_i)$ of the vector $\mathbf{b} \in \mathbb{F}_q^m$.
**Result:** Unique, public solution $\mathbf{x} \in \mathbb{F}_q^m$ such that $\mathbf{Ax} = \mathbf{b}$

```
for j = m downto 2 do
    x[j] = FullAdd((b[j]_i))
    for k = 1 upto j − 1 do
        (b[k]_i) := (b[k]_i + x[j] · A[k,j]_i)
x[1] = FullAdd((b[1]_i))
return x
```

More precisely, the system is solved by moving from the final row ($m$) to the first one, as in typical back substitution. Firstly, the solution of the current row $\mathbf{x}_{[j]} = (\mathbf{b}_{[j]_i})$ is securely unmasked. Secondly, all the elements above in column $j$ of $(\mathbf{A}_i)$ are made zero, by (securely) multiplying all its elements $(\mathbf{A}_{[1:j-1,j]_i})$ with the solution $\mathbf{x}_{[j]}$. As the multiplier is unmasked, the operation can be performed share-wise. For each row, that result is subtracted from the element in vector $\mathbf{b}$, share by share. Finally, this process is repeated for all rows except the first one, which can be directly solved and unmasked.

### 3.7.1 Complexity

The run-time and randomness complexity of `SecBackSub` are:

$$\begin{aligned} T_{\texttt{SecBackSub}}(n,m) &= (m-1)\cdot T_{\texttt{FullAdd}}(n) + (\frac{m(m-1)}{2}\cdot 2n) + T_{\texttt{FullAdd}}(n) \\ &= \frac{3}{2}n^2m - \frac{3}{2}mn - m + m^2n\,, \\ R_{\texttt{SecBackSub}}(n,m,w) &= (m-1)\cdot R_{\texttt{FullAdd}}(n,w) + 0 + R_{\texttt{FullAdd}}(n,w) \\ &= \frac{(n^2-n)mw}{2}\,. \end{aligned}$$

### 3.7.2 Security

We now argue about the first- and high-order security of Algorithm 7 by proving it to be $t$-NIo secure with $n = t+1$ shares and public output $\mathbf{x}$. This means it provides resistance against an adversary with $t$ probes and allows the algorithm to be used in larger compositions.

**Lemma 6.** *The gadget* `SecBackSub` *(Algorithm 7) is $t$-NIo secure with public output $\mathbf{x}$.*

*Proof.* We first show that a single iteration $j$ is $t$-NIo secure with output $\mathbf{x}_{[j]}$, of which an abstract diagram is shown in Figure 5. We model the extraction of element $j$ and column $j$ from vector $(\mathbf{b}_i)$ and matrix $(\mathbf{A}_i)$ as $t$-NI gadgets $G_1$-$G_2$ and $G_3$, respectively. We also model the loop of share-wise multiplications and additions in Line 3-4 as a single $t$-NI gadget, which can be trivially shown as the operations are performed share by share and each iteration is independent. As a result, the iterations are assumed to be executed in parallel, and we summarize them into a single gadget $G_5$. An adversary can probe the intermediate values $t_{G_i}$ and output shares $o_{G_i}$ of each gadget $G_i$, except the outputs of the entire gadget. The total number of probes in Algorithm 7 is defined as:

$$t_{A_7} = \sum_{i=1}^{5} t_{G_i} + \sum_{i=1}^{4} o_{G_i}$$

To prove Lemma 6, we show that the internal and output probes of each gadget in Algorithm 7 can be perfectly simulated with $\leq t_{A_7}$ input shares. As $t$-NI security implies that the simulation of internal and output probes of a gadget $G_i$ requires a corresponding number of shares of its input. From this, it is clear that on a matrix/vector-level, the probes cannot be perfectly simulated as there are duplicate entries in the sum ($2 \cdot t_{G_5}$ of $(\mathbf{b}_i)$). To overcome this issue, we model gadget $G_5$ to operate on individual entries in row/column vectors rather than on the complete variables. As a result, to simulate $t_{G_5}$ intermediate values requires $t_{G_5}$ shares of element $j$ and elements $[1 : j-1]$. As the coefficients inside a column/row are independent, the simulation succeeds. Following the flow from the output to the inputs, all probes required for simulation are summed up. We

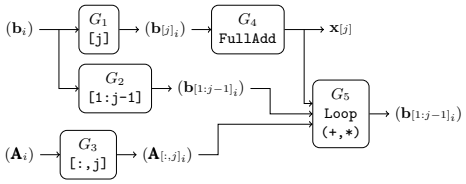


Fig. 5: An abstract diagram of a single iteration $j$ of `SecBackSub` (Algorithm 7). The $t$-NI gadgets are depicted with a single border, the $t$-SNI gadgets with a double border.

determine the set of shares of the inputs $|I|$, required for simulating the entire algorithm. As $|I| = t_{G_1} + o_{G_1} + t_{G_2} + o_{G_2} + t_{G_3} + o_{G_3} + t_{G_4} + o_{G_4} + t_{G_5} \leq t_{A_7}$, the iteration $j$ is $t$-NI. We have shown that all $t$ probes placed by an adversary in a single iteration $j$ can be simulated with no more number of shares of inputs $(\mathbf{A}_i)$ and $(\mathbf{b}_i)$. As a result, all probes across different iterations can also be simulated with no more number of input shares. □

## 4 Application to Different Digital Signature Schemes

This section discusses the operational and randomness cost of the masked Gaussian elimination on multivariate- and code-based digital signatures. In total, we observed that at least the following ten UOV-based digital signatures use some variant of Gaussian elimination: UOV, MAYO, SNOVA, QR-UOV, MQ-Sign, PROV, VOX, TUOV, VDOO, IPRainbow, and WAVE. In between these schemes, UOV, MAYO, SNOVA, and QR-UOV have advanced to the second round of the NIST additional digital signature standardization procedure [32], and MQ-Sign is a second-round candidate for the Korean PQC standardization procedure [37]. Therefore, we have restricted our cost analysis to these five multivariate-based digital signature candidates in this work. As seen from the complexity analysis in Section 3, the cost of the masked Gaussian elimination mainly depends on the dimension of the matrix $\mathbf{A}$ $(m)$, the width of the modulus $q$ $(w)$, and the number of shares $(n)$.

We present a cost analysis of the five selected schemes in Table 2. We have used the parameters of the corresponding scheme in the run-time and randomness complexity equations to calculate the expected arithmetic operational and randomness cost of the masked Gaussian elimination for each scheme, respectively. It can be seen from the table that the effect of the matrix dimension $m$ and the share count $n$ is much more potent in operation and randomness costs than the width of the modulus $q$. For instance, in the two versions of UOV for the NIST security level I, the operational cost of masked GE in UOV-Ip ($m = 44$ & $q = 256$) is 2.9× less compared to UOV-Is ($m = 64$ & $q = 16$) for first-, second-, and third-order masking. The randomness cost of GE in UOV-Ip is 1.5× less compared to UOV-Is for first, second, and third-order masking. Due to similar reasons, the operational cost of first-, second-, and third-order masked

Table 2: Cost estimation of masked Gaussian elimination to different digital signature schemes. Here, $m$ represents the dimension of the matrix $\mathbf{A}$, $q$ represents the modulus, and $n$ represents the number of masking shares.

| Scheme | NIST Security Level | Scheme Parameters | | Operations (x1000) | | | Randomness (KB) | | |
|---|---|---|---|---|---|---|---|---|---|
| | | $q$ | $m$ | $n=2$ | $n=3$ | $n=4$ | $n=2$ | $n=3$ | $n=4$ |
| UOV [8] | I | 256 | 44 | 105 | 260 | 482 | 742 | 2226 | 4452 |
| | | 16 | 64 | 300 | 747 | 1392 | 1112 | 3336 | 6671 |
| | III | 256 | 72 | 428 | 1065 | 1986 | 3146 | 9437 | 18873 |
| | V | 256 | 96 | 985 | 2459 | 4590 | 7360 | 22079 | 44158 |
| MAYO [7] | I | 16 | 64 | 300 | 747 | 1392 | 1112 | 3336 | 6671 |
| | III | 16 | 96 | 973 | 2434 | 4546 | 3680 | 11040 | 22079 |
| | V | 16 | 128 | 2263 | 5670 | 10597 | 8638 | 25914 | 51827 |
| QR-UOV [20] | I | 7 | 100 | 1084 | 2717 | 5076 | 3102 | 9306 | 18612 |
| | | 31 | 60 | 249 | 619 | 1155 | 1147 | 3441 | 6881 |
| | | 31 | 70 | 388 | 968 | 1806 | 1806 | 5417 | 10834 |
| | | 127 | 54 | 184 | 457 | 852 | 1175 | 3524 | 7048 |
| | III | 7 | 140 | 2922 | 7333 | 13712 | 8431 | 25292 | 50584 |
| | | 31 | 87 | 730 | 1825 | 3407 | 3432 | 10295 | 20590 |
| | | 31 | 100 | 1097 | 2744 | 5125 | 5184 | 15550 | 31100 |
| | | 127 | 78 | 531 | 1327 | 2476 | 3472 | 10415 | 20829 |
| | V | 7 | 190 | 7217 | 18131 | 33918 | 20942 | 62825 | 125650 |
| | | 31 | 114 | 1610 | 4032 | 7533 | 7646 | 22937 | 45873 |
| | | 31 | 120 | 1872 | 4689 | 8761 | 8904 | 26710 | 53419 |
| | | 127 | 105 | 1265 | 3166 | 5914 | 8373 | 25119 | 50237 |
| SNOVA [41] | I | 16 | 68 | 357 | 890 | 1660 | 1329 | 3987 | 7974 |
| | | 16 | 72 | 421 | 1051 | 1961 | 1573 | 4719 | 9437 |
| | | 16 | 80 | 572 | 1428 | 2666 | 2147 | 6439 | 12877 |
| | III | 16 | 100 | 1097 | 2744 | 5125 | 2147 | 6439 | 12877 |
| | | 16 | 99 | 1065 | 2664 | 4976 | 4031 | 12093 | 24186 |
| | | 16 | 128 | 2263 | 5670 | 10597 | 8638 | 25914 | 51827 |
| | V | 16 | 132 | 2477 | 6209 | 11604 | 9465 | 28395 | 56789 |
| | | 16 | 135 | 2647 | 6634 | 12400 | 10119 | 30356 | 60712 |
| | | 16 | 160 | 4369 | 10959 | 20489 | 16773 | 50317 | 100634 |
| MQ-Sign [39] | I | 256 | 46 | 119 | 294 | 547 | 845 | 2534 | 5068 |
| | III | 256 | 72 | 428 | 1065 | 1986 | 3146 | 9437 | 18873 |
| | V | 256 | 96 | 985 | 2459 | 4590 | 7360 | 22079 | 44158 |

GE is 2.3× higher, and the randomness cost is 1.2× higher in MAYO compared to UOV for III and V, both security levels.

As the modulus $q$ in QR-UOV is prime, we considered the next closest power-of-two integer as the modulus during the complexity cost analysis of masked Gaussian elimination on QR-UOV. However, the actual cost is likely to be higher because a masked prime modulus reduction is often more expensive than a masked power-of-two modulus reduction. We can also observe from Table 2 that the operational and randomness costs of masked GE in QR-UOV and SNOVA even vary hugely for the same security level. In fact, two NIST security level III variants ((i) $m = 87$ & $q = 31$, and (ii) $m = 78$ & $q = 127$) of QR-UOV are cheaper to protect than a NIST security level I variant ($m = 100$ & $q = 7$) of QR-UOV. We conclude that the main contributors to operational and randomness cost in masked GE with back substitution are matrix dimension $m$ and masking order $t = n - 1$.

## 5 Implementation Results

In this section, we discuss our software (C, M4) implementation of the gadgets introduced in Section 3 and evaluate their performance for different UOV parameter sets and sharing degrees. The performance results are obtained from running our code[6], which we make publicly available, on the STM32Discovery board, containing an STM32 Arm Cortex-M4 microcontroller. We use the on-chip TRNG for on-the-fly randomness generation, which is included in the total cycle count. Below, we discuss and show results for practically relevant security orders ($n = 2, 3, 4$), but our implementation could be scaled up to arbitrary order. We emphasize that our implementation is only a proof-of-concept. Mitigation of micro-architectural leakages and optimized masked implementations are beyond the scope of this work and left as future work.

Table 3: Detailed performance overview (cycle counts ×1000) of masked Gaussian Elimination ($n = 2, 3, 4$) for UOV-I, -III, -V on Arm Cortex-M4.

| Scheme | Operation | Umasked | Masked | | |
|---|---|---|---|---|---|
| | | | 1st-order | 2nd-order | 3rd-order |
| UOV-I (m=44 q=256) | GE (Total) | 1034 | **15649 (15.1×)** | **39587 (38.3×)** | **69186 (66.9×)** |
| | SecRowEch | 999 | 15575 (15.6×) | 39469 (39.5×) | 69021 (69.1×) |
| | Step 1 | 80 | 1998 (25×) | 5642 (70.5×) | 9901 (123.8×) |
| | Step 2 | | 39 | 108 | 193 |
| | Step 3 | | 598 | 1335 | 2426 |
| | Step 4 | | 12938 | 32383 | 56498 |
| | SecBackSub | 32 | 74 (2.3×) | 118 (3.7×) | 164 (5.1×) |
| UOV-III | GE (Total) | 4117 | **62680 (15.2×)** | **157901 (38.4×)** | **275768 (67×)** |
| UOV-V | GE (Total) | 9336 | **143361 (15.4×)** | **360590 (38.6×)** | **1025942 (109.9×)** |

### 5.1 Case Study: UOV

Table 3 shows the performance results (clock cycles) of our masked implementation for all three UOV security levels and scaled to first-, second- and third-masking order. We include the performance numbers of the unmasked UOV reference code [8], of the supported parameter sets. We repeat that the randomness generation overhead is included in the benchmarking results. The full masked GE operation results in about 15.1/15.2/15.4× overhead for 2 shares at NIST security level I/III/V. As expected, the performance overhead increases significantly for higher-order protected implementations. Our second-order masked implementation has overhead factor 38.3/38.4/38.6× for UOV-I, -III and -V, respectively. The third-order implementation has an overhead of 66.9-109.9× for different NIST security levels. The main contributor to the cycle count is the conversion to row-echelon form, more specifically, Steps 1 and 4 ($\sim 95\%$). At all protection orders, the repeated execution of `SecMultSub` and `SecCondAdd` are the main bottlenecks and good targets for future optimizations.

[6] arm-none-eabi-gcc v10.3.1

We observe that the masked implementation of Step 1, which makes the pivot element non-zero, has an overhead of $\sim 25/70/124\times$ for 2/3/4 shares. The authors of UOV propose an optimization for the iteration of conditional additions (Algorithm 1), which we also implemented. Instead of iterating over all rows below the current pivot row (up to $m$), the UOV submission describes iterating over (and conditionally adding) only a few rows below the pivot row. The exact amount depends on the parameter selection but ensures a sufficiently low probability of the pivot element being non-zero. We encourage other UOV-like candidates to explore similar optimizations, which will lead to more efficient (masked) implementations of GE with back substitution.

## 6 Conclusions

In this work, we presented first- and higher-order masked algorithms for Gaussian elimination with back substitution: `SecRowEch` & `SecBackSub`. We analyze several novel multivariate- and code-based PQC schemes and show that GE is a critical operation for solving a system of linear equations and requires side-channel protection. Our `SecCondAdd` gadget allows us to make a pivot element nonzero by conditionally adding other rows in the matrix without revealing if it is zero. We rely on the `SecScalarMult` gadget to efficiently multiply a matrix row with the (masked) inverse of its pivot element to make the pivot element one. Our approach only requires a single mask conversion. For the same masking order, the matrix dimension $m$ is the main contributor to operation and randomness cost in masked GE. The MAYO scheme has a larger $m$ compared to UOV, and as a result, the GE is $2.3\times$ and $1.2\times$ more expensive to mask. Future work includes analysis of reduced iteration counts in the GE, as proposed in the UOV specification, for other PQC DS candidates. We implement our algorithms in C for arbitrary protection orders and parameter sets of different PQC schemes. We also evaluate its performance on Arm Cortex-M4 platforms. In future work, a complete masking and hardening against physical attacks for all mentioned schemes can be constructed using the methods proposed in this work.

**Acknowledgements.** This work was partially supported by Horizon 2020 ERC Advanced Grant (101020005 Belfort), Horizon Europe (101070008 ORSHIN), CyberSecurity Research Flanders with reference number VOEWICS02, BE QCI: Belgian-QCI (3E230370) (see beqci.eu), and Intel Corporation. Anindya Ganguly is supported by TCS research fellowship. The work of Angshuman Karmakar is supported by the Research-I foundation from Infosys, the Initiation grant from IIT Kanpur, and the Google India research fellowship.

## References

1. Aulbach, T., Campos, F., Krämer, J., Samardjiska, S., Stöttinger, M.: Separating oil and vinegar with a single trace side-channel assisted kipnis-shamir attack on UOV. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2023**(3), 221–245 (2023). https://doi.org/10.46586/TCHES.V2023.I3.221-245

2. Banegas, G., Carrier, K., Chailloux, A., Couvreur, A., Debris-Alazard, T., Gaborit, P., Karpman, P., Loyer, J., Niederhagen, R., Sendrier, N., Smith, B., Tillich, J.: Wave specification document (2023), https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/wave-spec-web.pdf
3. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P.A., Grégoire, B., Strub, P.Y., Zucchini, R.: Strong non-interference and type-directed higher-order masking. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 116–129. ACM Press (Oct 2016). https://doi.org/10.1145/2976749.2978427
4. Barthe, G., Belaïd, S., Espitau, T., Fouque, P.A., Grégoire, B., Rossi, M., Tibouchi, M.: Masking the GLP lattice-based signature scheme at any order. In: Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part II 37. pp. 354–384. Springer (2018)
5. Barthe, G., Fong, N., Gaboardi, M., Grégoire, B., Hsu, J., Strub, P.Y.: Advanced probabilistic couplings for differential privacy. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 55–67. ACM Press (Oct 2016). https://doi.org/10.1145/2976749.2978391
6. Bettale, L., Coron, J.S., Zeitoun, R.: Improved high-order conversion from boolean to arithmetic masking. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 22–45 (05 2018). https://doi.org/10.46586/tches.v2018.i2.22-45
7. Beullens, W., Campos, F., Celi, S., Hess, B., Kannwischer, M.J.: Mayo specification document (2023), https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/mayo-spec-web.pdf
8. Beullens, W., Chen, M.S., Ding, J., Gong, B., Kannwischer, M.J., Patarin, J., Peng, B.Y., Schmidt, D., Shih, C.J., Tao, C., Yang, B.Y.: Uov: Unbalanced oil and vinegar algorithm specifications and supporting documentation version 1.0 (2023), https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/UOV-spec-web.pdf
9. Cartor, R., Cartor, M., Lewis, M., Smith-Tone, D.: Iprainbow. In: Cheon, J.H., Johansson, T. (eds.) Post-Quantum Cryptography - 13th International Workshop, PQCrypto 2022, Virtual Event, September 28-30, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13512, pp. 170–184. Springer (2022). https://doi.org/10.1007/978-3-031-17234-2_9
10. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards Sound Approaches to Counteract Power-Analysis Attacks. In: Wiener, M. (ed.) Advances in Cryptology — CRYPTO' 99. pp. 398–412. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
11. Chen, K.Y., Chen, J.P.: Masking floating-point number multiplication and addition of falcon: First-and higher-order implementations and evaluations. IACR Transactions on Cryptographic Hardware and Embedded Systems **2024**(2), 276–303 (2024)
12. Chou, T., Kannwischer, M.J., Yang, B.: Rainbow on cortex-m4. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(4), 650–675 (2021). https://doi.org/10.46586/TCHES.V2021.I4.650-675
13. Cogliati, B., Faugère, J., Fouque, P., Goubin, L., Larrieu, R., Macario-Rat, G., Minaud, B.: Vox specification document (2023), https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/vox-spec-web.pdf

14. Coron, J.S.: High-order conversion from Boolean to arithmetic masking. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 93–114. Springer, Cham (Sep 2017). https://doi.org/10.1007/978-3-319-66787-4$_5$
15. Coron, J.S., Großschädl, J., Vadnala, P.K.: Secure conversion between boolean and arithmetic masking of any order. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 188–205. Springer (2014)
16. Coron, J.S., Spignoli, L.: Secure wire shuffling in the probing model. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology – CRYPTO 2021. pp. 215–244. Springer International Publishing, Cham (2021)
17. Coron, J.S., Gérard, F., Montoya, S., Zeitoun, R.: High-order polynomial comparison and masking lattice-based encryption. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 153–192 (11 2022). https://doi.org/10.46586/tches.v2023.i1.153-192
18. Coron, J.S., Gérard, F., Trannoy, M., Zeitoun, R.: Improved gadgets for the high-order masking of dilithium. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2023**(4), 110—145 (Aug 2023). https://doi.org/10.46586/tches.v2023.i4.110-145
19. Ding, J., Gong, B., Guo, H., He, X., Jin, Y., Pan, Y., Schmidt, D., Tao, C., Xie, D., Yang, B.Y., Zhao, Z.: Tuov: Specification document v1.0 (2023), `https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/TUOV-spec-web.pdf`
20. Furue, H., Ikematsu, Y., Hoshino, F., Takagi, T., Yasuda, K., Miyazawa, T., Saito, T., Nagai, A.: QR-UOV specification document (2023), `https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/qruov-spec-web.pdf`
21. Ganguly, A., Karmakar, A., Saxena, N.: VDOO: A short, fast, post-quantum multivariate digital signature scheme. In: Chattopadhyay, A., Bhasin, S., Picek, S., Rebeiro, C. (eds.) Progress in Cryptology - INDOCRYPT 2023 - 24th International Conference on Cryptology in India, Goa, India, December 10-13, 2023, Proceedings, Part II. Lecture Notes in Computer Science, vol. 14460, pp. 197–222. Springer (2023). https://doi.org/10.1007/978-3-031-56235-8_10
22. Genelle, L., Prouff, E., Quisquater, M.: Thwarting higher-order side channel analysis with additive and multiplicative maskings. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 240–255. Springer, Berlin, Heidelberg (Sep / Oct 2011). https://doi.org/10.1007/978-3-642-23951-9$_1$6
23. Goubin, L., Cogliati, B., Faugère, J., Fouque, P.A., Larrieu, R., Macario-Rat, G., Minaud, B., Patarin, J.: Prov specification document (2023), `https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/prov-spec-web.pdf`
24. Guerreau, M., Martinelli, A., Ricosset, T., Rossi, M.: The hidden parallelepiped is back again: Power analysis attacks on falcon. IACR Trans. Cryptogr. Hardw. Embed. Syst.s **2022**(3), 141–164 (Jun 2022). https://doi.org/10.46586/tches.v2022.i3.141-164
25. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: Securing hardware against probing attacks. In: Advances in Cryptology-CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings 23. pp. 463–481. Springer (2003)
26. Ito, A., Ueno, R., Homma, N.: On the success rate of side-channel attacks on masked implementations: Information-theoretical bounds and their practical usage. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. p. 1521–1535. CCS '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3548606.3560579

27. Johnson, D.S., Garey, M.R.: Computers and Intractability: A Guide to the Theory of NP-completeness. WH Freeman (1979)
28. Kipnis, A., Patarin, J., Goubin, L.: Unbalanced oil and vinegar signature schemes. In: Stern, J. (ed.) Advances in Cryptology — EUROCRYPT '99. pp. 206–222. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
29. Kundu, S., D'Anvers, J., Beirendonck, M.V., Karmakar, A., Verbauwhede, I.: Higher-order masked saber. In: Galdi, C., Jarecki, S. (eds.) Security and Cryptography for Networks - 13th International Conference, SCN 2022, Amalfi, Italy, September 12-14, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13409, pp. 93–116. Springer (2022). https://doi.org/10.1007/978-3-031-14791-3_5
30. Mathieu-Mahias, A., Quisquater, M.: Mixing additive and multiplicative masking for probing secure polynomial evaluation methods. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(1), 175–208 (2018). https://doi.org/10.13154/tches.v2018.i1.175-208
31. NIST: Call for additional digital signature schemes for the post-quantum cryptography standardization process (2022), `https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf`, accessed: 2024-10-30
32. NIST: Ir 8528: Status report on the first round of the additional digital signature schemes for the nist post-quantum cryptography standardization process. Tech. rep., U.S. Department of Commerce, Washington, D.C. (2024). https://doi.org/10.6028/NIST.IR.8528
33. NIST: Module-lattice-based digital signature standard. Tech. rep., U.S. Department of Commerce, Washington, D.C. (2024). https://doi.org/10.6028/NIST.FIPS.204
34. NIST: Module-lattice-based key-encapsulation mechanism standard. Tech. rep., U.S. Department of Commerce, Washington, D.C. (2024). https://doi.org/10.6028/NIST.FIPS.203
35. NIST: Stateless hash-based digital signature standard. Tech. rep., U.S. Department of Commerce, Washington, D.C. (2024). https://doi.org/10.6028/NIST.FIPS.205
36. Park, A., Shim, K.A., Koo, N., Han, D.G.: Side-channel attacks on post-quantum signature schemes based on multivariate quadratic equations. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(3), 500–523 (Aug 2018). https://doi.org/10.13154/tches.v2018.i3.500-523
37. Quantum-Resistant Cryptography Research Group: Kpqc competition round 2 (2024), `https://www.kpqc.or.kr/competition_02.html`, accessed: 2024-10-30
38. Schneider, T., Paglialonga, C., Oder, T., Güneysu, T.: Efficiently masking binomial sampling at arbitrary orders for lattice-based crypto. In: Public-Key Cryptography – PKC 2019: 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography, Beijing, China, April 14-17, 2019, Proceedings, Part II. p. 534–564. Springer-Verlag, Berlin, Heidelberg (2019). https://doi.org/10.1007/978-3-030-17259-6_18
39. Shim, K.A., Kim, J., An, Y.: Mq-sign: A new post-quantum signature scheme based on multivariate quadratic equations: Shorter and faster (2023), `https://www.kpqc.or.kr/images/pdf/MQ-Sign.pdf`
40. Ulitzsch, V.Q., Marzougui, S., Tibouchi, M., Seifert, J.P.: Profiling side-channel attacks on Dilithium. In: Smith, B., Wu, H. (eds.) Selected Areas in Cryptography. pp. 3–32. Springer International Publishing, Cham (2024)
41. Wang, L.C., Chou, C.Y., Ding, J., Kuan, Y.L., Li, M.S., Tseng, B.S., Tseng, P.E., Wang, C.C.: Snova specification document (2023), `https:`

//csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/SNOVA-spec-web.pdf
42. Yi, H., Nie, Z.: Side-channel security analysis of UOV signature for cloud-based internet of things. Future Gener. Comput. Syst. **86**, 704–708 (2018). https://doi.org/10.1016/J.FUTURE.2018.04.083

# A Auxiliary Algorithms

**Algorithm 8:** `Refresh`, from [5]

**Data:** A Boolean sharing $(x_i)$ of $x \in \mathbb{F}_q$

**Result:** A Boolean sharing $(y_i)$ of $y \in \mathbb{F}_q$ such that $\sum_{i=1}^{n} y_i = \sum_{i=1}^{n} x_i$

$(y_i) := (x_i)$
**for** $i = 2$ *upto* $n$ **do**
  $r \leftarrow \mathbb{F}_q$
  $y_1 = y_1 + r$
  $y_i = y_i - r$
**return** $(y_i)$

**Algorithm 9:** `StrongRefresh`, from [5]

**Data:** A Boolean sharing $(x_i)$ of $x \in \mathbb{F}_q$

**Result:** A Boolean sharing $(y_i)$ of $y \in \mathbb{F}_q$ such that $\sum_{i=1}^{n} y_i = \sum_{i=1}^{n} x_i$

$(y_i) := (x_i)$
**for** $i = 1$ *upto* $n$ **do**
  **for** $j = i + 1$ *upto* $n$ **do**
    $r \leftarrow \mathbb{F}_q$
    $y_i = y_i + r$
    $y_j = y_j - r$
**return** $(y_i)$

**Algorithm 10:** `FullAdd`, from [15,4]

**Data:** A Boolean sharing $(y_i)$

**Result:** Unmasked value $y$ such that $y = \sum_{i=1}^{n} y_i$

$(a_i) :=$ `StrongRefresh`$((y_i))$ `/* free-`$t$`-SNI */`
$y := a_1 + \cdots + a_n$
**return** $y$

**Algorithm 11:** `SecNonzero`, from [11]

**Data:** 1. A Boolean sharing $(x_i)$ of a coefficient $x \in \mathbb{F}_q$.
2. Parameter $w = \lceil \log(q) \rceil$

**Result:** One-bit Boolean sharing $(b_i)$ such that $\sum_{i=1}^{n} b_i = 0 \Leftrightarrow \sum_{i=1}^{n} x_i = 0$

$(t_i) := (x_i)$
len $:= w/2$
**while** $len \geq 1$ **do**
  $(l_i) :=$ `StrongRefresh`$((t_i^{[2\text{len:len}]}))$
  $(r_i) := (t_i^{[\text{len:1}]})$
  $(t_i) =$ `SecOR`$((l_i), (r_i))$ `/* [11] */`
  len $=$ len $\gg 1$
**return** $(t_i^{[1]})$

**Algorithm 12:** `B2M`, from [22]

**Data:** A Boolean sharing $(x_i)$ of a coefficient $x \in \mathbb{F}_q$

**Result:** A multiplicative sharing $(m_i)$ such that $\sum_{i=1}^{n} x_i = \prod_{i=1}^{n} m_i$

$m_1 := x_1$
**for** $j = 2$ *upto* $n$ **do**
  $m_j \leftarrow \mathbb{F}_q$
  $m_1 = m_1 * m_j$
  **for** $k = 2$ *upto* $n - j + 1$ **do**
    $r \leftarrow \mathbb{F}_q$
    $x_k = m_j * x_k$
    `## Refresh additive share`
    $x_k = x_k + r$
    $m_1 = m_1 + x_k$
    $x_k = r$
  $x_{n-j+2} = x_{n-j+2} * m_j$
  $m_1 = m_1 + x_{n-j+2}$
  $m_j = m_j^{-1}$
**return** $(m_i)$

## B Complexity Analysis of SecNonzero

The run-time and randomness complexity of SecNonzero (Alg. 11) are:

$$\begin{aligned}
T_{\texttt{SecNonzero}}(n) &= n+1+w\cdot(T_{\texttt{StrongRefresh}}(n)+n+T_{\texttt{SecOR}}(n)+1)\\
&= n+1+(\lceil\log(w+1)\rceil-1)\cdot(\frac{3n^2-3n}{2}+n+(2n+T_{\texttt{SecAnd}}(n)+1)+1)\\
&= n+1+(\lceil\log(w+1)\rceil-1)\cdot(\frac{3n^2-3n}{2}+n+(2n+\frac{7n^2-5n}{2}+1)+1)\\
&= (5n^2+2n-1)+\lceil\log(w+1)\rceil\cdot(5n^2-n+2)\,,
\end{aligned}$$

$$\begin{aligned}
R_{\texttt{SecNonzero}}(n,w) &= \sum_{len=1}^{(\lceil\log(w+1)\rceil-1)}\cdot(R_{\texttt{StrongRefresh}}(n,len)+R_{\texttt{SecOR}}(n,len))\\
&= \sum_{len=1}^{(\lceil\log(w+1)\rceil-1)}\cdot((\frac{n^2-n}{2}\cdot len)+R_{\texttt{SecAnd}}(n,len))\\
&= \sum_{len=1}^{(\lceil\log(w+1)\rceil-1)}\cdot(\frac{n^2-n}{2}+\frac{n^2-n}{2})\cdot len\\
&= \frac{\lceil\log(w+1)\rceil^2-\lceil\log(w+1)\rceil}{2}\cdot(n^2-n)\,.
\end{aligned}$$

## C Complexity Analysis of SecRowEch

The run-time and randomness complexity of SecRowEch (Alg. 6) are:

$$\begin{aligned}
T_{\texttt{SecRowEch}}(n,m) =& \frac{m^2-m}{2}\cdot(T_{\texttt{SecNonzero}}(n)+T_{\texttt{SecNOT}}(n))+\frac{2m^3+3m^2+m}{6}\cdot\\
& T_{\texttt{SecCondAdd}}(n,1)+m\cdot T_{\texttt{SecNonzero}}(n)+m\cdot T_{\texttt{FullAdd}}(n)+m\\
& +m\cdot T_{\texttt{B2Minv}}(n)+\frac{m^2+3m}{2}\cdot T_{\texttt{SecScalarMult}}(n,1)+\frac{m^2-m}{2}\cdot\\
& T_{\texttt{StrongRefresh}}(n)+\frac{2m^3+3m^2+m}{6}\cdot T_{\texttt{SecMultAdd}}(n,1)\\
=& \frac{m^2-m}{2}\cdot(((5n^2+2n-1)+\lceil\log(w+1)\rceil\cdot(5n^2-n+2))+1)\\
& +\frac{2m^3+3m^2+m}{6}\cdot(5n^2-3n)+m\cdot((5n^2+2n-1)+\\
& \lceil\log(w+1)\rceil\cdot(5n^2-n+2))+m\cdot\frac{3n^2-n-2}{2}+m\\
& +m\cdot\frac{5n^2-5n+4}{2}+\frac{m^2+3m}{2}\cdot(5n^2-3n)+\frac{m^2-m}{2}\\
& \cdot\frac{3n^2-3n}{2}+\frac{2m^3+3m^2+m}{6}\cdot\frac{7n^2-3n}{2}\,,
\end{aligned}$$

$$
\begin{aligned}
R_{\texttt{SecRowEch}}(n,m,w) =& \frac{m^2-m}{2} \cdot (R_{\texttt{SecNonzero}}(n,w) + R_{\texttt{SecNOT}}(n,w)) \\
&+ \frac{2m^3+3m^2+m}{6} \cdot R_{\texttt{SecCondAdd}}(n,1,w) + m \cdot \\
&R_{\texttt{SecNonzero}}(n,w) + m \cdot R_{\texttt{FullAdd}}(n,w) + m \cdot R_{\texttt{B2Minv}}(n,w) \\
&+ \frac{m^2+3m}{2} \cdot R_{\texttt{SecScalarMult}}(n,1,w) + \frac{m^2-m}{2} \cdot \\
&R_{\texttt{StrongRefresh}}(n,w) + \frac{2m^3+3m^2+m}{6} \cdot R_{\texttt{SecMultAdd}}(n,1,w) \\
=& \frac{m^2-m}{2} \cdot \frac{\lceil \log(w+1) \rceil^2 - \lceil \log(w+1) \rceil}{2} \cdot (n^2-n) \\
&+ \frac{2m^3+3m^2+m}{6} \cdot (n^2-n)w + m \cdot \frac{\lceil \log(w+1) \rceil^2 - \lceil \log(w+1) \rceil}{2} \cdot \\
&(n^2-n) + m \cdot \frac{(n^2-n)w}{2} + m \cdot \frac{n^2-n}{2} \cdot w + \frac{m^2+3m}{2} \cdot \\
&(n^2-n)w + \frac{m^2-m}{2} \cdot (\frac{n^2-n}{2} \cdot w) + \frac{2m^3+3m^2+m}{6} \cdot \\
&\frac{n^2-n}{2} \cdot w \,.
\end{aligned}
$$

# D Security Proofs of Step 1-4 of Algorithm 6

***Step 1 ($G_1$ - $G_6$, $t$-SNI, Fig. 6)***: we first show that a single iteration of the loop (Line 4-7) is $t$-SNI secure. We model the extraction of rows $j$ and $k$ (and elements within that row) from matrix $\mathbf{T}$ as $t$-NI gadgets $G_1$ - $G_3$. This is trivial to show as selected rows from the matrix pass through and the rest is discarded. The `SecNonzero` and `SecNOT` operations are modeled as $t$-SNI and $t$-NI gadgets $G_4$ and $G_5$, respectively. The `SecCondAdd` operation is modeled as $t$-SNI gadget $G_6$, operating on rows $j$ and $k$. An adversary can probe each gadget $G_i$ internally $t_{G_i}$ and at the output $o_{G_i}$. The total number of probes in Step 1 is defined as $t_{S_1}$ and the output shares as $|O|$, with

$$
t_{S_1} = \sum_{i=1}^{6} t_{G_i} + \sum_{i=1}^{5} o_{G_i}, \quad |O| = o_{G_6}
$$

We now show that the internal and output probes of each gadget in Step 1 can be perfectly simulated with $\leq t_{S_1}$ input shares. To simulate the internal and output probes of $G_6$, $t_{G_6}$ shares of each of its inputs are required. Gadget $G_4$ is $t$-SNI and stops the propagation of probes to the input: only $t_{G_4}$ shares of the output of $G_3$ are required. The internal probes and output shares of gadgets $G_1$ and $G_2$ can be simulated with a corresponding number of shares of the input ($\mathbf{T}_i$), which is a problem as the required input shares is ($t_{G_1}$ + $o_{G_1} + t_{G_2} + o_{G_2} + t_{G_3} + o_{G_3} + t_{G_4} + 2 \cdot t_{G_6}$). It is clear that on a variable-level the probes cannot be perfectly simulated, which is why all gadgets work on a matrix row (or element) level. As such, each of these gadgets only requires the shares of a specific row of the input. And because the rows are independent, $t_{G_6}$

shares of row $k$ and $t_{G_6}$ shares of row $j$ are required, and thus the simulation succeeds. For the entire composition, the required set of shares of the input is $|I| = t_{G_1} + o_{G_1} + t_{G_2} + o_{G_2} + t_{G_3} + o_{G_3} + t_{G_4} + t_{G_6}$, which is independent from $|O|$. As a result, each iteration is $t$-SNI secure, and as a result, the whole loop (Step 1) is too. □

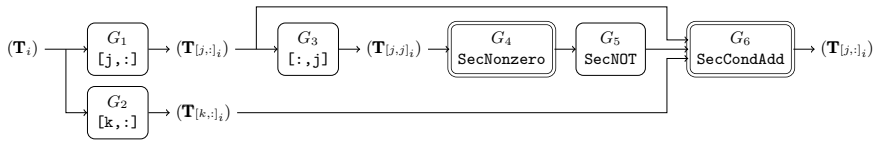


Fig. 6: An abstract diagram of a single iteration $k$ of Step 1 in `SecRowEch` (Algorithm 6). The $t$-NI gadgets are depicted with a single border and the $t$-SNI gadgets with a double border. Probes are defined at the row/column and element level (and not matrix-level) to ensure sound simulation.

***Step 2 ($G_7$ - $G_9$, $t$-NIo, Fig. 7)***: we model the extraction of element $j$ from a matrix-row as $t$-NI gadget $G_7$. As can be seen in the abstract diagram, `SecNonzero` is the $t$-SNI gadget $G_8$ and `FullAdd` is $t$-NI secure $G_9$. It is clear from its chained structure that the full Step 2 is $t$-NI secure if $\mathbf{c}_{[j]}$ is given to the simulator. □

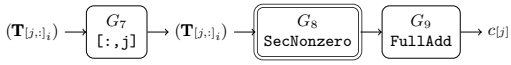


Fig. 7: An abstract diagram of Step 2 in `SecRowEch` (Algorithm 6). The $t$-NI gadgets are depicted with a single border and the $t$-SNI gadgets with a double border.

***Step 3 ($G_{10}$ - $G_{12}$, $t$-SNI, Fig. 8)***: initially, the pivot is extracted from row $j$ of matrix $(\mathbf{T}_i)$, which we model as $t$-NI gadget $G_{10}$. The $t$-SNI operations `B2Minv` and `SecScalarMult` are modeled as gadgets $G_{11}$ and $G_{12}$, respectively. An adversary can probe the intermediate values $t_{G_i}$ and output shares $o_{G_i}$ of

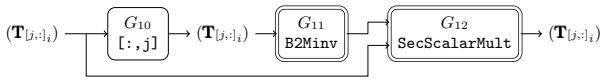


Fig. 8: An abstract diagram of Step 3 in `SecRowEch` (Algorithm 6). The $t$-NI gadgets are depicted with a single border, the $t$-SNI gadgets with a double border.

each gadget $G_i$. The total number of possible probes in Step 3 is defined as $t_{S_3}$

and its output shares as $|O|$, with

$$t_{S_3} = \sum_{i=10}^{12} t_{G_i} + \sum_{i=10}^{11} o_{G_i}, \quad |O| = o_{G_{12}}$$

We now show that the internal and output probes of each gadget in Step 3 can be perfectly simulated with $\leq t_{S_3}$ input shares. To simulate the $t_{G_{12}}$ intermediate values and $o_{G_{12}}$ output shares of $G_{12}$, only $t_{G_{12}}$ shares of the input $(\mathbf{T}_{[j,:]_i})$ and the output of $G_{11}$ are required. For $G_{11}$ too, the $t$-SNI property allows to simulate all probes with only $t_{G_{11}}$ of its input. The simulation of $t_{G_{10}} + o_{G_{10}}$ probes on $G_{10}$ requires the same amount of shares from its input. We now sum the required shares of the input $|I|$ that are required to simulate all probes on the gadgets in Step 3. As $|I| = t_{G_{10}} + o_{G_{10}} + t_{G_{11}} + t_{G_{12}} \leq t_{S_3}$ and independent of $|O|$, Step 3 is $t$-SNI. □

***Step 4 ($G_{13}$ - $G_{16}$, $t$-NI, Fig. 9)***: we first show that a single iteration $k$ of the loop (Line 17-20) is $t$-NI secure. The extraction of row $k$ and its coefficient $j$ from matrix $(\mathbf{T}_i)$ are modeled as $t$-NI gadgets $G_{13}$ & $G_{14}$. We model `StrongRefresh` as $t$-SNI gadget $G_{15}$ and the $t$-NI secure `SecMultSub` as $G_{16}$. An adversary can

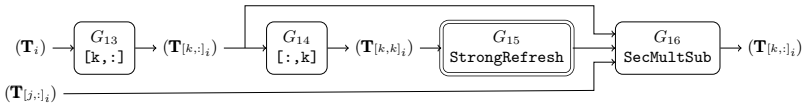


Fig. 9: An abstract diagram of a single iteration $k$ of Step 4 in `SecRowEch` (Algorithm 6). The $t$-NI gadgets are depicted with a single border and the $t$-SNI gadgets with a double border.

probe both intermediate values $t_{G_i}$ and output shares $o_{G_i}$ of each gadget $G_i$. The total number of probes in this step $t_{S_4}$ is defined as:

$$t_{S_4} = \sum_{i=13}^{16} t_{G_i} + \sum_{i=13}^{15} o_{G_i}$$

We now show that all probes on intermediate values and output shares of each gadget in Step 4 can be perfectly simulated with $\leq t_{S_4}$ shares of both inputs. Simulating $t_{G_{16}}$ probes in $G_16$, requires $t_{G_{16}}$ probes of the output of $G_{15}$, input $(\mathbf{T}_{[j,:]_i})$ and $(\mathbf{T}_{[k,:]_i})$. Gadget $G_{15}$ stops the propagation of probes from the output to the input, as it is $t$-SNI. Starting at the output of Step 4 and following the flow through all gadgets towards the input, all probes are summed ($|I|$). As $|I| = t_{G_{13}} + o_{G_{13}} + t_{G_{14}} + o_{G_{14}} + t_{G_{15}} + t_{G_{16}} \leq t_{S_4}$ shares of both inputs are required for simulation of an iteration $k$ in Step 4, it is $t$-NI secure. As each iteration is independent and computing a single row $k$, they can be assumed to be executed in parallel. As a result, we can summarize the gadgets in each iteration as single gadgets across all iterations. This means the entire loop is $t$-NI. □