# Masked Solving of Linear Equations System and Application to UOV Signatures

Jean-Sébastien Coron[1], François Gérard[1] and Bowen Zhang[2,1]

[1] DCS, University of Luxembourg, Esch-sur-Alzette, Luxembourg
jean-sebastien.coron@uni.lu
fragerar@protonmail.com
[2] Crypto Group, ICTEAM, UCLouvain, Louvain-la-Neuve, Belgium
bowen.zhang@uclouvain.be

**Abstract.** In response to the looming quantum threat, NIST has selected four algorithms for standardization (i.e., ML-KEM, ML-DSA, SLH-DSA, and FN-DSA), yet three of the four schemes are based on Euclidean lattices, which raises concerns about the mathematical diversity of post-quantum algorithms. NIST has therefore announced an additional call for post-quantum signatures with a preference for schemes constructed from assumptions other than lattices. Among such candidates, the Unbalanced Oil and Vinegar (UOV) signature over the multivariate quadratic (MQ) problem is attractive for its short signature and security against quantum cryptanalysis. However, the practical implementations of UOV remain vulnerable to power side-channel attacks.

In this paper, we address this issue by presenting two improved techniques for masking linear equations system solving at arbitrary order, with a proof of security in the $t$-probing model. We show that in the masked setting, our inversion-based techniques outperform Gaussian elimination, unlike the unmasked setting where Gaussian elimination is typically more efficient. As an application, we develop a complete C implementation of the fully masked UOV signing using our improved techniques. Compared to masked Gaussian elimination, our techniques achieve at least 2.2 times speedup at high-order. Against Kundu's latest implementation (CCS 2025), our masked UOV signing is faster by an average factor of 4.0 at higher masking orders.

**Keywords:** Multivariate signature · Unbalanced Oil and Vinegar · Masking

## 1 Introduction

**Post-quantum cryptography.** For more than a decade now, the threat of quantum computing has loomed over the world of cybersecurity. The theoretical results in quantum information theory support the fact that a large-scale quantum computer would be able to run algorithms capable of jeopardizing the security of information systems by attacking the underlying cryptographic algorithms. Hence, the cryptographic research community has been putting a lot of effort in the past years to develop quantum-proof algorithms. This process has been particularly propelled by the post-quantum cryptography standardization competition that has been started by NIST in 2016. This led to four new standards, one encryption scheme and three signature schemes, all except one (namely ML-KEM, ML-DSA, and FN-DSA) based on the hardness of computational problems over Euclidean lattices. To scale the mathematical diversity of the post-quantum signature schemes, and investigate the possibility of having short signatures and fast verification, NIST launched

an additional call for digital signature schemes in 2022. By October 2024, 14 out of 50 candidates had advanced to the second round[1].

**Multivariate signatures.** Among all the second-round candidates, four are based on multivariate cryptography: MAYO [BCC+23], QR-UOV [FIH+23], SNOVA [WTKC23], and UOV [BCD+23]. Actually, all of them are modern variants of the classic Oil and Vinegar (OV) scheme proposed by Patarin in 1997 [Pat97], whose security is based on the difficulty of solving systems of quadratic multivariate polynomials (over finite field). By making the vinegar space larger than the oil space, one obtains a stronger scheme called the Unbalanced Oil and Vinegar (UOV) scheme [KPG99]. Previously, the research community showed great interest in other OV variants such as Rainbow [DS05] for its better performance and shorter key size than the plain UOV. However, Rainbow was broken by Beullens in 2022 [Beu22] and was not selected as a finalist in the NIST PQC standardization Process. Since the classic UOV scheme with properly optimized parameter sets is already practical to implement [BCH+23] and has undergone intensive cryptanalysis for more than two decades, it has returned to the spotlight of the community. The four selected multivariate candidates are either UOV itself, or close variants. In this paper, we will focus on the UOV signature scheme.

**Solving linear equations system in UOV.** UOV signature generation involves solving a linear system $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$, which can be done using two main approaches: (1) computing $\mathbf{A}^{-1}$ and multiplying it by $\mathbf{b}$, or (2) applying Gaussian elimination to obtain $\mathbf{x}$ directly. Both methods follow similar principles. They begin by constructing an augmented matrix $\mathbf{T}$ (in slightly different ways) and then transforming $\mathbf{T}$ into row echelon form. During this process, all subsequent rows are conditionally added to the pivot row to ensure that the pivot element is non-zero and thus invertible. This step has to be performed in constant-time [BCH+23]. If the pivot remains zero, the matrix is singular, and the algorithm aborts with $\perp$. Next, the pivot is normalized to one, and all entries above and below it are eliminated. After these steps, the left half of $\mathbf{T}$ is the identity matrix. In the case of matrix inversion, one extracts the right half of $\mathbf{T}$ as $\mathbf{A}^{-1}$, and then computes the solution $\mathbf{x} = \mathbf{A}^{-1} \cdot \mathbf{b}$. In the case of Gaussian elimination, the solution $\mathbf{x}$ is recovered directly in the last column of $\mathbf{T}$. While the first technique was used in [SLK22, CKY21], the authors of [BCH+23] showed that Gaussian elimination is more efficient. Accordingly, modern UOV-based schemes employ the latter approach.

**Side-channel vulnerabilities.** While cryptanalysts are assessing the UOV for resistance against computational attacks, it is equally important to explore practical challenges that real world implementations will face. One of those practical issues is resistance against power side-channel attacks. In this class of attacks, the adversary is observing the physical behaviors (e.g., power consumption and electromagnetic emanations) of the device performing cryptographic operations and using the results of those measurements to learn potential secret information. Although the UOV scheme has withstood more than two decades of cryptanalysis, its implementation on embedded devices is still vulnerable to side-channel exploitation. Indeed, several recent studies have already demonstrated the feasibility of power analysis on UOV(-based) signatures [ACK+23, JD24, PSKH18, ACK25].

**The masking countermeasures.** Masking is a well-known and provably secure countermeasure against side-channel attacks. Side-channel attacks exploit correlations between secret values and physical measurements, so masking breaks such correlations by introducing randomness into every sensitive intermediate value. More precisely, a secret value

[1] See https://csrc.nist.gov/projects/pqc-dig-sig/round-2-additional-signatures.

$x$ is split into $x_1 \circ x_2 \cdots \circ x_d = x$ for some operation $\circ$, and all cryptographic operations are then performed on the shares rather than on $x$. When $\circ$ is addition, the shares are called *arithmetic shares*, and when $\circ$ is $\oplus$, the shares are called *boolean shares*. Notably, in UOV signature, the field addition over $\mathbb{F}_{2^k}$ is the same as bit-wise XOR, which makes the arithmetic shares identical to the boolean shares. While several masking schemes have been designed over the years for various cryptographic algorithms, including some post-quantum candidates, only few have studied the efficiency of masking for UOV(-based) schemes. To our knowledge, the authors of [NKO+25] described the first masked Gaussian elimination that can be used to solve linear equations system in the UOV signing procedure, and in [KNK+25] they proposed a more detailed analysis for the masking of the whole UOV scheme, using the techniques from [NKO+25]. Concurrently, the authors in [ACK25] provided a systematic overview on the physical security of UOV-based schemes.

**Masking Gaussian elimination.** In [NKO+25], the authors introduced masked Gaussian elimination built upon two main gadgets: `SecRowEch` that performs the masked matrix reduction, and `SecBackSub` that performs a back substitution on the reduced matrix to find the solution. The bottleneck of their approach lies in the `SecRowEch` procedure, and more specifically in the secure normalisation of all elements below the pivot to zero, which is done by subtracting scalar multiples of the pivot row from all rows below (refer to Algorithm 6 in [NKO+25]). In the ISW probing model, a secure multiplication over $\mathbb{F}_q$ requires $\mathcal{O}(d^2)$ operations and $\mathcal{O}(d^2)$ randomness [ISW03]. Therefore, the overall run-time and randomness complexity of `SecRowEch` from [NKO+25] are both $\mathcal{O}(m^3 d^2)$ for $m$-dimension oil space and $d$ shares.

**Our contributions.** This paper introduces improved high-order masking techniques for solving the linear equations system in the UOV signature generation, and gives security proofs for all proposed gadgets in the $t$-probing model. Additionally, we provide a C implementation of the fully masked UOV signature scheme to demonstrate the practicality and efficiency of our countermeasures.

More specifically, our first contribution is presenting two new techniques for solving linear equations system at high-order that both outperform the masked Gaussian elimination from [NKO+25]. Although using *Gaussian elimination* is more efficient than matrix inversion to find the solution in the unmasked UOV signing, we show that this is not necessarily the case in masked implementation.

Our first technique is based on masking matrix inversion via additive shares, which enables us to directly compute $\mathbf{x} = \mathbf{A}^{-1} \cdot \mathbf{b}$. For this, we multiply $\mathbf{A}$ on both sides with two random, additively masked matrices $\mathbf{C}$ and $\mathbf{D}$, which yields a new matrix $\mathbf{B} = (\mathbf{D}_1 + \cdots + \mathbf{D}_d) \cdot (\mathbf{A}_1 + \cdots + \mathbf{A}_d) \cdot (\mathbf{C}_1 + \cdots + \mathbf{C}_d)$. We prove that we can then compute $\mathbf{B}$ and its inverse in the clear. Therefore, we obtain the additively masked $\mathbf{A}^{-1} = (\mathbf{C}_1 + \cdots + \mathbf{C}_d) \cdot \mathbf{B}^{-1} \cdot (\mathbf{D}_1 + \cdots + \mathbf{D}_d)$, from which we eventually obtain the additively masked solution $\mathbf{x}$. Note that in this technique, the invertibility of $\mathbf{B}$ depends jointly on $\mathbf{A}$, $\mathbf{C}$, and $\mathbf{D}$, which introduces an additional failure probability compared to the black-box algorithm. To address this issue, when $\mathbf{B}$ is non-invertible, we resample fresh $\mathbf{C}$ and $\mathbf{D}$, and recompute $\mathbf{B}$. This procedure is repeated until $\mathbf{B}$ is invertible or for at most 16 iterations. In this case, the additional failure probability is reduced to a negligible level, which is $2^{-111}$ for $q = 256$. We describe our first technique as `SecLinSolveMatMult` in Section 3.3, and provide a detailed completeness analysis.

Our second technique follows the same principle to compute $\mathbf{x}$, but relies on multiplicative shares. Starting from $\mathbf{A} = \mathbf{A}_1 + \cdots + \mathbf{A}_d$, we multiply both sides of $\mathbf{A}$ by random invertible matrices $\mathbf{U}_i$ and $\mathbf{R}_i$, which gives $\mathbf{B} = \mathbf{U}_d \cdots \mathbf{U}_1 \cdot (\mathbf{A}_1 + \cdots + \mathbf{A}_d) \cdot \mathbf{R}_1 \cdots \mathbf{R}_d$. We prove as previously that the matrix $\mathbf{B}^{-1}$ can now be computed in the clear, so that the multiplicative shares of $\mathbf{A}^{-1}$ are constructed as $\mathbf{A}^{-1} = \mathbf{R}_1 \cdots \mathbf{R}_d \cdot \mathbf{B}^{-1} \cdot \mathbf{U}_d \cdots \mathbf{U}_1$,

from which we eventually derive an additive sharing of the solution $\mathbf{x}$. We describe the corresponding procedure as `SecLinSolveA2M` in Section 4.2.

Compared to masked Gaussian elimination [NKO+25], our techniques significantly reduce the randomness complexity, from $\mathcal{O}(m^3d^2)$ down to $\mathcal{O}(m^2d^2)$, where $m$ denotes the matrix dimension and $d$ the number of shares. The runtime complexity of our methods remains $\mathcal{O}(m^3d^2)$ as in [NKO+25], while we require fewer operations in practice. We refer to Section 6.1 for concrete measurements of randomness usage and run-time of our techniques.

Our second contribution is a more efficient high-order masked implementation of UOV signing. We integrate our new techniques into the signing procedure, and develop the masked implementation in C on top of the UOV reference implementation[2]. In particular, the latest masked UOV signing implementation from Kundu et al. [KNK+25] relies on the masked Gaussian elimination from [NKO+25], and does not incorporate the implementation-level optimizations used in the reference code. As a result, compared to their work, we achieve an average of 4.0× speedup in signing at higher masking orders under `UOV-Ip` parameter. Detailed performance results are provided in Section 6.2. Our open source implementation can be found at

https://github.com/zh-bw/pqov-masked

**Future work.** This paper only discusses the theoretical security of our countermeasures in the $t$-probing model, thus we leave the concrete leakage evaluation of our countermeasures as future work.

**Open problem.** In the UOV signing procedure, the matrix of the linear system should be invertible, otherwise the procedure is restarted (see Line 8 in Figure 2). In our two countermeasures, we additionally assume that the rank of the matrix is leaked to the adversary. This means that if the matrix is non-invertible, the adversary will additionally learn its rank, which is not the case in the black-box signing procedure. We note that this is also the case in the masked Gaussian elimination proposed in [NKO+25], which inevitably discloses the rank as well. We leave as an open problem the design of a countermeasure that does not disclose the rank of the matrix whenever it is non-invertible.

# 2 Background

## 2.1 Notations

We denote by $\mathbb{F}_q$ the finite field with $q$ elements where $q$ is a power-of-two integer, and $\mathbb{F}_q^\ell$ the vector space of dimension $\ell$ over $\mathbb{F}_q$. We denote by $\mathbb{F}_q^{m\times m}$ the set of $m \times m$ matrices over $\mathbb{F}_q$, and by $\mathbb{F}_q^{m\times m^*}$ the group of invertible $m \times m$ matrices over $\mathbb{F}_q$. The identity matrix of dimension $m$ is denoted by $\mathbf{I}_m$. We use bold lower case letters to denote vectors (e.g., $\mathbf{v}$) and bold upper case letters to denote matrices (e.g., $\mathbf{A}$). The transpose of a matrix $\mathbf{A}$ is denoted by $\mathbf{A}^\top$. The rank of a matrix $\mathbf{A}$ is denoted by $\mathrm{rank}(\mathbf{A})$. We denote $d = t + 1$ as the number of shares, where $t$ is the security order.

## 2.2 Security definitions

To formalize the security of masking countermeasures, Ishai et al. [ISW03] introduced the $t$-probing security model, in which an adversary can have at most $t$ probes on the circuit's wires. More precisely, they showed how to protect against side-channel attacks by splitting the secret into $d = 2t + 1$ shares, and proved that any boolean circuit with

[2] https://github.com/pqov/pqov.

size $|C|$ can be transformed to a $t$-probing secure circuit of size $\mathcal{O}(d^2|C|)$. However, when the size of the circuit is large, it is complex to analyze its security in the $t$-probing model. In [BBD+16], the authors proposed a scalable way to approach the $t$-probing security and reduced the number of shares to $d = t + 1$. A large circuit can be decomposed into multiple smaller circuits, which are called *gadgets*. Security is then proven per gadget and preserved under composition. They introduced two security definitions, called (Strong) Non-Interference (SNI/NI) security. Specifically, SNI security is stronger than NI security because the number of input shares needed to simulate all probes depends only on the number of internal probes and is independent of how many output variables must be simulated. Moreover, composing a NI secure gadget with a SNI secure gadget yields overall SNI security. In this paper, we show that all proposed gadgets satisfy either SNI or NI security. We now recall the SNI/NI security definitions from [BBD+16].

**Definition 1** ($t$-NI security)**.** Let $G$ be a gadget taking as input $d$ shares $(a_i)_{1\leq i\leq d}$ and outputting $d$ shares $(b_i)_{1\leq i\leq d}$. The gadget $G$ is said to be $t$-NI secure if for any set of $t_1$ probed intermediate variables, 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 can be perfectly simulated from $a_{|I}$.

**Definition 2** ($t$-SNI security)**.** Let $G$ be a gadget taking as input $d$ shares $(a_i)_{1\leq i\leq d}$ and outputting $d$ shares $(b_i)_{1\leq i\leq d}$. The gadget $G$ is said to be $t$-SNI secure if for any set of $t_1$ probed intermediate variables and any subset $O$ of output indices, such that $t_1 + |O| \leq t$, there exists a subset $I$ of input indices with $|I| \leq t_1$, such that the $t_1$ intermediate variables and the output variables $b_{|O}$ can be perfectly simulated from $a_{|I}$.

Since we will use the secure share recombination technique, we recall the extension of the $t$-SNI security notion called *free-$t$-SNI* from [CS21]. The *free-$t$-SNI* notion allows the simulator to perfectly simulate all output variables except one.

**Definition 3** (*free-$t$-SNI* security)**.** Let $G$ be a gadget taking as input $d$ shares $(a_i)_{1\leq i\leq d}$ and outputting $d$ shares $(b_i)_{1\leq i\leq d}$. The gadget $G$ is said to be *free-$t$-SNI* secure if for any set of $t_1 < t$ probed intermediate variables, there exists a subset $I$ of input indices with $|I| \leq t_1$, such that the $t_1$ intermediate variables and the output variables $b_{|I}$ can be perfectly simulated from $a_{|I}$, while for any $O \subsetneq [1, d] \setminus I$, the output variable $b_{|O}$ are uniformly and independently distributed, conditioned on the probed variables and $b_{|I}$.

As explained in introduction, we assume that the adversary learns the rank of the matrix of the linear system. Therefore, in our security proof, the rank is provided to the simulator.

## 2.3 The Unbalanced Oil and Vinegar signature scheme

Multivariate public-key schemes base their security on the hardness of solving systems of multivariate equations. In Figure 1 we briefly recall the mechanism of UOV signature. The core element in UOV is the multivariate quadratic map $\mathcal{P} : \mathbb{F}_q^n \mapsto \mathbb{F}_q^m$. Given a target $\mathbf{t} \in \mathbb{F}_q^m$, it is hard to find a preimage $\mathbf{s} \in \mathbb{F}_q^n$ such that $\mathcal{P}(\mathbf{s}) = \mathbf{t}$, and this is considered to be hard even for quantum computers when $n \sim m$. However, this inversion of $\mathcal{P}$ can be efficiently done when a trapdoor is present. In UOV, this trapdoor is known as the *oil space*, which is a $m$-dimension linear subspace $O \subset \mathbb{F}_q^n$, and $\mathcal{P}$ vanishes on $O$ (i.e., $\mathcal{P}(\mathbf{o}) = 0$ for all $\mathbf{o} \in O$). To sign a message $\mu$, one first computes the target $\mathbf{t} = \mathcal{H}(\mu) \in \mathbb{F}_q^m$, where $\mathcal{H}$ is a cryptographic hash function. Then the signing procedure boils down to finding a preimage $\mathbf{s}$ such that $\mathcal{P}(\mathbf{s}) = \mathbf{t}$. This is easy with the knowledge of the oil space. One needs to guess a vector $\mathbf{v}$ and tries to finding another vector $\mathbf{o} \in O$ satisfying $\mathcal{P}(\mathbf{v} + \mathbf{o}) = \mathbf{t}$. For any quadratic map $\mathcal{P} : \mathbb{F}_q^n \mapsto \mathbb{F}_q^m$, we have a bilinear and symmetric differential form $\mathcal{P}' : \mathbb{F}_q^n \times \mathbb{F}_q^n \mapsto \mathbb{F}_q^m$, which gives $\mathcal{P}(\mathbf{v} + \mathbf{o}) = \mathcal{P}(\mathbf{v}) + \mathcal{P}(\mathbf{o}) + \mathcal{P}'(\mathbf{v}, \mathbf{o})$. Now finding the

**KeyGen**($\texttt{params}(n, m, q)$) :

1: Choose $m$ OV-polynomials $f^{(1)}, \cdots, f^{(m)}$ uniformly at random
2: $\mathcal{F} := f^{(1)}, \cdots, f^{(m)}$
3: Choose an invertible linear transformation $\mathcal{T} : \mathbb{F}_q^n \to \mathbb{F}_q^n$
4: $\mathcal{P} := \mathcal{F} \circ \mathcal{T}$
5: $\texttt{pk} := \mathcal{P}$
6: $\texttt{sk} := (\mathcal{F}, \mathcal{T})$
7: **return** $(\texttt{pk}, \texttt{sk})$

**Sign**($\texttt{params}, \texttt{sk} = (\mathcal{F}, \mathcal{T}), \mu \in \{0,1\}^*$) :

1: $\mathbf{t} \leftarrow \texttt{Hash}(\mu)$
2: Find a random preimage $\mathbf{u} \in \mathbb{F}_q^n$ of $\mathbf{t}$ such that $\mathcal{F}(\mathbf{u}) = \mathbf{t}$
3: $\mathbf{s} := \mathcal{T}^{-1}(\mathbf{u})$
4: $\sigma := \mathbf{s}$
5: **return** $\sigma$

**Verify**($\texttt{params}, \texttt{pk} = \mathcal{P}, (\mu, \sigma = \mathbf{s}) \in \{0,1\}^* \times \mathbb{F}_q^n$) :

1: $\mathbf{t} \leftarrow \texttt{Hash}(\mu)$
2: $\mathbf{t}' := \mathcal{P}(\sigma)$
3: **return** $(\mathbf{t} == \mathbf{t}')$

Figure 1: Simplified version of original UOV signature.

preimage of $\mathbf{t}$ can be reduced to solving the linear equations system in $\mathbf{o}$, and is in the form of

$$\underbrace{\mathcal{P}(\mathbf{v})}_{\text{constant}} + \underbrace{\mathcal{P}(\mathbf{o})}_{\text{zero}} + \underbrace{\mathcal{P}'(\mathbf{v}, \mathbf{o})}_{\text{linear in } \mathbf{o}} = \underbrace{\mathbf{t}}_{\text{constant}} \tag{1}$$

with $m$ equations and $m$ variables. Finally, we compute $\mathbf{s} = \mathbf{v} + \mathbf{o}$, and it holds that $\mathcal{P}(\mathbf{s}) = \mathbf{t}$. This high-level signing strategy not only applies to UOV, but also to other UOV-based schemes, such as MAYO, QR-UOV, and SNOVA. Thus, solving the linear equations system in the form of (1) is the key component in the signing procedure of UOV-based schemes.

**Variants.** The UOV scheme provides a flexible way to represent its keypair, either in compressed or expanded forms. Based on where the key expansion is performed, we can distinguish three variants of UOV.

- In the `classic` variant, both public key and secret key expansion are performed during key generation, which results in larger key sizes but faster signing and verification.
- In the `pkc` variant, secret key expansion is carried out at key generation, while public key expansion is done at run time during verification. This makes the public key smaller, but slows down the verification.
- Finally, in the `pkc + skc` variant, both public key and secret key expansion are performed on-the-fly. This double-compressed version provides the smallest key sizes, but the slowest signing and verification.

**Parameter sets.** Table 1 summarizes the four recommended parameter sets for UOV in NIST Round-2 submission. Note that there are two recommended parameter sets for Security Level 1, namely 1) `UOV-Ip` works over $\mathbb{F}_{256}$ and achieves smaller key size, and 2) `UOV-Is` works over $\mathbb{F}_{16}$ with shorter signatures. For Security Level 3 and 5, there is only one recommended parameter set per level.

Table 1: Recommended parameter sets for UOV signature scheme [BCD+23].

| | NIST S.L. | $n$ | $m$ | $q$ |
|---|---|---|---|---|
| `UOV-Ip` | 1 | 112 | 44 | 256 |
| `UOV-Is` | 1 | 160 | 64 | 16 |
| `UOV-III` | 3 | 184 | 72 | 256 |
| `UOV-V` | 5 | 244 | 96 | 256 |

# 3 Masked solving linear equations via additive shares

In this section, we present our first technique for masking linear equations system solving via additive shares. We begin with two building blocks: masked matrix-vector multiplication and masked matrix multiplication, and then we describe how to compose them to perform the masked matrix inversion and masked solving of the linear equations system.

## 3.1 Masked matrix-vector multiplication

Matrix-vector multiplication is a core operation in UOV. We now give its masked variant, which is derived from the original ISW multiplication [ISW03]; we consider the variant with linear memory complexity [Cor14]. Given a matrix $\mathbf{A} = \mathbf{A}_1 + \cdots + \mathbf{A}_d$, and a vector $\mathbf{b} = \mathbf{b}_1 + \cdots + \mathbf{b}_d$, the goal is to output shares $\mathbf{x}_i$ such that $\sum_{i=1}^{d} \mathbf{x}_i = (\sum_{i=1}^{d} \mathbf{A}_i) \cdot (\sum_{i=1}^{d} \mathbf{b}_i)$. We present the procedure in Algorithm 1 and analyse its cost and security. While we assume that $q$ is a power-of-two, the algorithm is easily extended to any finite field $\mathbb{F}_q$.

**Algorithm 1** Masked matrix-vector multiplication (`SecMatVecMult`)

**Input:** shares $\mathbf{A}_1, \ldots, \mathbf{A}_d \in \mathbb{F}_q^{m \times m}$, and shares $\mathbf{b}_1, \ldots, \mathbf{b}_d \in \mathbb{F}_q^m$
**Output:** shares $\mathbf{x}_1, \ldots, \mathbf{x}_d \in \mathbb{F}_q^m$ satisfying $\sum_{i=1}^{d} \mathbf{x}_i = (\sum_{i=1}^{d} \mathbf{A}_i) \cdot (\sum_{i=1}^{d} \mathbf{b}_i)$

1: **for** $i = 1$ to $d$ **do** $\mathbf{x}_i \leftarrow \mathbf{A}_i \cdot \mathbf{b}_i$
2: **for** $i = 1$ to $d$ **do**
3:  **for** $j = i + 1$ to $d$ **do**
4:  $\mathbf{r}_{i,j} \leftarrow \mathbb{F}_q^m$
5:  $\mathbf{r}_{j,i} \leftarrow (\mathbf{A}_i \cdot \mathbf{b}_j + \mathbf{r}_{i,j}) + \mathbf{A}_j \cdot \mathbf{b}_i$
6:  $\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{r}_{i,j}$ ▷ Referred by $\mathbf{x}_{i,j}$
7:  $\mathbf{x}_j \leftarrow \mathbf{x}_j + \mathbf{r}_{j,i}$ ▷ Referred by $\mathbf{x}_{j,i}$
8: **return** $\mathbf{x}_1, \ldots, \mathbf{x}_d$

**Complexity.** The run-time and randomness complexity of `SecMatVecMult` are:

$$T_{\texttt{SecMatVecMult}}(m, d) = d \cdot (2m-1)m + \frac{d(d-1)}{2} \cdot \big(m + (4m-2)m + m + 2m\big) = \mathcal{O}(m^2 d^2),$$

$$R_{\texttt{SecMatVecMult}}(m, d) = \frac{d(d-1)}{2} \cdot m = \mathcal{O}(md^2).$$

**Security.** The following lemma shows that `SecMatVecMult` achieves $t$-SNI security. We provide a detailed proof in Appendix A.

**Lemma 1** ($t$-SNI security of `SecMatVecMult` (Algorithm 1)**.** *For any set of $t_1$ intermediate variables and any subset $O$ of output indices with $t_1 + |O| \leq t < d$, the $t_1$ intermediate variables and the output variables $\mathbf{x}_{|O}$ can be perfectly simulated from the input variables $\mathbf{A}_{|I}$ and $\mathbf{b}_{|J}$ with $|I|, |J| \leq t_1$.*

## 3.2 Masked matrix multiplication

To multiply two masked matrices, we simply apply the masked matrix-vector multiplication to each column of the second matrix. Concretely, with $\mathbf{A}_1, \ldots, \mathbf{A}_d \in \mathbb{F}_q^{m \times m}$ and $\mathbf{C}_1, \ldots, \mathbf{C}_d \in \mathbb{F}_q^{m \times m}$, we process each column $\mathbf{C}^{[j]}$ of $\mathbf{C}$ using `SecMatVecMult` to obtain masked columns $\mathbf{B}^{[j]} = \mathbf{A} \cdot \mathbf{C}^{[j]}$. Stacking these columns yields additive shares of $\mathbf{B} = \mathbf{A} \cdot \mathbf{C}$. We describe the full procedure as `SecMatMult` in Algorithm 2.

**Algorithm 2** Masked matrix-matrix multiplication (`SecMatMult`)

**Input:** shares $\mathbf{A}_1, \ldots, \mathbf{A}_d \in \mathbb{F}_q^{m \times m}$, and shares $\mathbf{C}_1, \ldots, \mathbf{C}_d \in \mathbb{F}_q^{m \times m}$
**Output:** $\mathbf{B}_1, \ldots, \mathbf{B}_d$ satisfying $\sum_{i=1}^{d} \mathbf{B}_i = (\sum_{i=1}^{d} \mathbf{A}_i) \cdot (\sum_{i=1}^{d} \mathbf{C}_i)$

1: **for** $j = 1$ to $m$ **do**
2: $\quad (\mathbf{B}_1^{[j]}, \ldots, \mathbf{B}_d^{[j]}) \leftarrow \texttt{SecMatVecMult}((\mathbf{A}_1, \ldots, \mathbf{A}_d), (\mathbf{C}_1^{[j]}, \ldots, \mathbf{C}_d^{[j]}))$
3: **return** $\mathbf{B}_1, \ldots, \mathbf{B}_d$

**Complexity.** The run-time and randomness complexity of `SecMatMult` are:

$$T_{\texttt{SecMatMult}}(m, d) = m \cdot T_{\texttt{SecMatVecMult}}(m, d) = \mathcal{O}(m^3 d^2),$$
$$R_{\texttt{SecMatMult}}(m, d) = m \cdot R_{\texttt{SecMatVecMult}}(m, d) = \mathcal{O}(m^2 d^2).$$

**Security.** The following lemma shows that `SecMatMult` achieves $t$-SNI security.

**Lemma 2** ($t$-SNI security of `SecMatMult` (Algorithm 2))**.** *For any set of $t_1$ intermediate variables and any subset $O$ of output indices with $t_1 + |O| \leq t < d$, the $t_1$ intermediate variables and the output variables $\mathbf{B}_{|O}$ can be perfectly simulated from the input variables $\mathbf{A}_{|I}$ and $\mathbf{C}_{|J}$ with $|I|, |J| \leq t_1$.*

*Proof.* The $t$-SNI security of `SecMatMult` follows from Lemma 1. Since each iteration is $t$-SNI secure and independent, the entire loop is $t$-SNI secure. □

## 3.3 Masked linear equations system solving using secure matrix multiplication

Now with masked matrix-vector and matrix multiplication in place, we show how to securely solve the linear equations system $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$ by computing $\mathbf{A}^{-1}$ in additive shares. We first left- and right-multiply $\mathbf{A}$ with two randomly generated matrices in additive shares. This step depends on the secure matrix multiplication (Algorithm 2). More precisely, given as input an additively masked $\mathbf{A} = \mathbf{A}_1 + \cdots + \mathbf{A}_d$, we generate $d$ random shares $\mathbf{C}_i$ of $\mathbf{C} = \mathbf{C}_1 + \cdots + \mathbf{C}_d$, and $\mathbf{D}_i$ of $\mathbf{D} = \mathbf{D}_1 + \cdots + \mathbf{D}_d$. By `SecMatMult`, we compute the shares as:

$$\mathbf{B}_1 + \cdots + \mathbf{B}_d = (\mathbf{D}_1 + \cdots + \mathbf{D}_d) \cdot (\mathbf{A}_1 + \cdots + \mathbf{A}_d) \cdot (\mathbf{C}_1 + \cdots + \mathbf{C}_d).$$

We then unmask the shares to $\mathbf{B} = \mathbf{B}_1 + \cdots + \mathbf{B}_d$, after applying a full refreshing on its shares (refer to `RefreshMasks` in Alg. 8 in App. B.2). If $\mathbf{B}$ is non-invertible, the above procedure is repeated with freshly sampled $\mathbf{C}$ and $\mathbf{D}$. This resampling is performed independently and bounded to at most 16 iterations. We argue that knowing $\mathbf{B}$ leaks no information about $\mathbf{A}$ other than its rank. In other words, $\mathbf{B}$ can be efficiently simulated from rank($\mathbf{A}$) only.

When $\mathbf{B}$ is invertible, from $\mathbf{B} = \mathbf{D} \cdot \mathbf{A} \cdot \mathbf{C}$, we can obtain the additively masked $\mathbf{A}^{-1}$ as:

$$\mathbf{A}^{-1} = \mathbf{C} \cdot \mathbf{B}^{-1} \cdot \mathbf{D} = (\mathbf{C}_1 + \cdots + \mathbf{C}_d) \cdot \mathbf{B}^{-1} \cdot (\mathbf{D}_1 + \cdots + \mathbf{D}_d).$$

Finally, we compute the additively shared solution $\mathbf{x}$ as:

$$\mathbf{x} = \mathbf{A}^{-1} \cdot \mathbf{b} = (\mathbf{C}_1 + \cdots + \mathbf{C}_d) \cdot \mathbf{B}^{-1} \cdot (\mathbf{D}_1 + \cdots + \mathbf{D}_d) \cdot (\mathbf{b}_1 + \cdots + \mathbf{b}_d)$$

We first process $\mathbf{b}_1, \ldots, \mathbf{b}_d$ with the shares $\mathbf{D}_1, \ldots, \mathbf{D}_d$ using the masked matrix-vector multiplication (`SecMatVecMult`) to obtain an intermediate masked vector, then since $\mathbf{B}^{-1}$ is unmasked, a share-wise matrix-vector multiplication (see `halfSecMatVecMult` in Alg. 6 in App. B.1) is sufficient to compute their product. Finally, we compute the result with the shares $\mathbf{C}_1, \ldots, \mathbf{C}_d$ using `SecMatVecMult` again. Therefore, given additively masked $\mathbf{b}$ and $\mathbf{A}$, we get an additively masked $\mathbf{x}$. We present the corresponding procedure as `SecLinSolveMatMult` in Algorithm 3.

**Algorithm 3** `SecLinSolveMatMult`

**Input:** shares $\mathbf{A}_1, \ldots, \mathbf{A}_d \in \mathbb{F}_q^{m \times m}$, and shares $\mathbf{b}_1, \ldots, \mathbf{b}_d \in \mathbb{F}_q^m$

**Output:** shares $\mathbf{x}_1, \ldots, \mathbf{x}_d \in \mathbb{F}_q^m$ satisfying $\sum_{i=1}^d \mathbf{b}_i = (\sum_{i=1}^d \mathbf{A}_i) \cdot (\sum_{i=1}^d \mathbf{x}_i)$, or $\perp$

**for** $t = 1$ to $16$ **do** ▷ Repeat up to 16 times if $\mathbf{B}$ is non-invertible
  **for** $i = 1$ to $d$ **do**
    $\mathbf{C}_i \leftarrow \mathbb{F}_q^{m \times m}$, $\mathbf{D}_i \leftarrow \mathbb{F}_q^{m \times m}$ ▷ Generate random matrices
  $(\mathbf{B}_1, \ldots, \mathbf{B}_d) \leftarrow$ `SecMatMult`$((\mathbf{A}_1, \ldots, \mathbf{A}_d), (\mathbf{C}_1, \ldots, \mathbf{C}_d))$
  $(\mathbf{B}_1, \ldots, \mathbf{B}_d) \leftarrow$ `SecMatMult`$((\mathbf{D}_1, \ldots, \mathbf{D}_d), (\mathbf{B}_1, \ldots, \mathbf{B}_d))$
  $(\mathbf{B}_1, \ldots, \mathbf{B}_d) \leftarrow$ `RefreshMasks`$(\mathbf{B}_1, \ldots, \mathbf{B}_d)$ ▷ Refer to Algorithm 8
  $\mathbf{B} \leftarrow \mathbf{B}_1 + \cdots + \mathbf{B}_d$
  **if** $\mathbf{B}$ is invertible **then**
    $(\mathbf{x}_1, \ldots, \mathbf{x}_d) \leftarrow$ `SecMatVecMult`$((\mathbf{D}_1, \ldots, \mathbf{D}_d), (\mathbf{b}_1, \ldots, \mathbf{b}_d))$
    $(\mathbf{x}_1, \ldots, \mathbf{x}_d) \leftarrow$ `halfSecMatVecMult`$(\mathbf{B}^{-1}, (\mathbf{x}_1, \ldots, \mathbf{x}_d))$ ▷ Refer to Algorithm 6
    $(\mathbf{x}_1, \ldots, \mathbf{x}_d) \leftarrow$ `SecMatVecMult`$((\mathbf{C}_1, \ldots, \mathbf{C}_d), (\mathbf{x}_1, \ldots, \mathbf{x}_d))$
    **return** $\mathbf{x}_1, \ldots, \mathbf{x}_d$
**return** $\perp$

**Completeness.** The following lemma shows that Algorithm `SecLinSolveMatMult` solves the linear system with probability $p \geq 1 - (2/(q-1))^{16}$. Therefore, for $q = 256$, the linear system is solved with probability $1 - (\frac{2}{255})^{16} > 1 - 2^{-111}$. Note that in the UOV signature scheme, the signature process already needs to be restarted when the matrix $\mathbf{A}$ is non-invertible. Therefore, when integrated in the full masking of UOV (see Sec. 5), the signature process will need to be restarted with additional probability less than $2^{-111}$.

**Lemma 3.** *If $\mathbf{A}$ is invertible, then Algorithm* `SecLinSolveMatMult` *outputs shares satisfying $\sum_{i=1}^d \mathbf{b}_i = (\sum_{i=1}^d \mathbf{A}_i) \cdot (\sum_{i=1}^d \mathbf{x}_i)$, with probability $p > 1 - (2/(q-1))^{16}$.*

*Proof.* A random $m \times m$ matrix in $\mathbb{F}_q$ is invertible with probability $\prod_{i=1}^m (1 - q^{-i}) \geq 1 - \sum_{i=1}^m q^{-i} \geq 1 - 1/(q-1)$. For $\mathbf{B} = \mathbf{D} \cdot \mathbf{A} \cdot \mathbf{C}$ to be invertible, both $\mathbf{C}$ and $\mathbf{D}$ must be invertible. The sampling of $(\mathbf{C}, \mathbf{D})$ is repeated up to 16 times. The result follows. □

**Complexity.** The run-time and randomness complexity of `SecLinSolveMatMult` are:

$$\begin{aligned} T_{\texttt{SecLinSolveMatMult}}(m,d) &= 2\,T_{\texttt{SecMatMult}}(m,d) + T_{\texttt{RefreshMasks}}(m,d) + m^2(d-1) + 2m^3 \\ &\quad + T_{\texttt{halfSecMatVecMult}}(m,d) + 2\,T_{\texttt{SecMatVecMult}}(m,d) = \mathcal{O}(m^3 d^2), \\ R_{\texttt{SecLinSolveMatMult}}(m,d) &= 2R_{\texttt{SecMatMult}}(m,d) + R_{\texttt{RefreshMasks}}(m,d) \\ &\quad + 2R_{\texttt{SecMatVecMult}}(m,d) = \mathcal{O}(m^2 d^2). \end{aligned}$$

Relative to masked Gaussian elimination [NKO+25], our methods reduce the randomness complexity from $\mathcal{O}(m^3 d^2)$ to $\mathcal{O}(m^2 d^2)$, with the same runtime complexity of $\mathcal{O}(m^3 d^2)$.

**Security.** We first show that the distribution of a matrix masked on both sides by independent uniform matrices depends only on its rank.

**Lemma 4** (Distribution of a randomly masked matrix)**.** *Let $\mathbf{A} \in \mathbb{F}_q^{m\times m}$ be a fixed matrix of rank $r$, and let $\mathbf{D}, \mathbf{C} \in \mathbb{F}_q^{m\times m}$ be independent and uniformly distributed. Let $\mathbf{E}_r \in \mathbb{F}_q^{m\times m}$ be the diagonal matrix whose first $r$ diagonal entries are 1 and whose remaining entries are 0. Then $\mathbf{DAC}$ and $\mathbf{DE}_r\mathbf{C}$ have the same distribution. Equivalently, for every $\mathbf{X} \in \mathbb{F}_q^{m\times m}$,*

$$\#\{(\mathbf{D}, \mathbf{C}) : \mathbf{DAC} = \mathbf{X}\} = \#\{(\mathbf{D}, \mathbf{C}) : \mathbf{DE}_r\mathbf{C} = \mathbf{X}\},$$

*where both pairs range over $(\mathbb{F}_q^{m\times m})^2$.*

*Proof.* By the rank normal form, there exist invertible matrices $\mathbf{P}, \mathbf{Q} \in \mathbb{F}_q^{m\times m}$ such that $\mathbf{A} = \mathbf{PE}_r\mathbf{Q}$. The map

$$(\mathbf{D}, \mathbf{C}) \longmapsto (\mathbf{DP}, \mathbf{QC})$$

is a bijection on $(\mathbb{F}_q^{m\times m})^2$, with inverse $(\mathbf{U}, \mathbf{V}) \longmapsto (\mathbf{UP}^{-1}, \mathbf{Q}^{-1}\mathbf{V})$. Consequently, $(\mathbf{DP}, \mathbf{QC})$ is uniformly distributed on $(\mathbb{F}_q^{m\times m})^2$, just like $(\mathbf{D}, \mathbf{C})$. Since

$$\mathbf{DAC} = (\mathbf{DP})\mathbf{E}_r(\mathbf{QC}),$$

the bijection restricts to a bijection between the two sets of pairs in the statement, proving the equality of their counts. Because every pair $(\mathbf{D}, \mathbf{C})$ is equally likely, the equality of counts also shows that the two products have the same distribution. □

The equality of preimage counts in Lemma 4 has also been formally verified in Lean 4 using Mathlib. The formalization is available in the Lean proof repository (commit `6d06233`).

The following lemma shows that the `SecLinSolveMatMult` algorithm achieves $t$-SNI security.

**Lemma 5** ($t$-SNI security of `SecLinSolveMatMult` (Algorithm 3))**.** *For any set of $t_1$ intermediate variables and any subset $O$ of output indices with $t_1 + |O| \leq t < d$, the $t_1$ intermediate variables and the output variables $\mathbf{x}_{|O}$ can be perfectly simulated from the input variables $\mathbf{A}_{|I}$ and $\mathbf{b}_{|I}$ with $|I| \leq t_1$, when $\mathrm{rank}(\mathbf{A})$ is given to the simulator.*

*Proof.* We denote the two `SecMatMult` gadgets from Line 4 and Line 5 as $G_1$, the share recombination from Line 6 and Line 7 as gadget $G_2$, `SecMatVecMult` at Line 9 as $G_3$, `halfSecMatVecMult` as $G_4$, and `SecMatVecMult` at Line 11 as $G_5$. We first have that $G_2$ is $t$-SNI secure from Lemma 2. Since `RefreshMasks` is *free-$t$-*SNI secure [CS21], then by the definition of *free-$t$-*SNI, we can perfectly simulate all intermediate variables and all outputs of `RefreshMasks` except one. We denote such output as $\mathbf{B}_{i^\star}$, and $\mathbf{B}_{i^\star}$ can be perfectly simulated by computing $\mathbf{B}_{i^\star} = \mathbf{B} - \sum_{i\neq i^\star}^{d} \mathbf{B}_i$.

We now show that $\mathbf{B}$ can be perfectly simulated from $r = \mathrm{rank}(\mathbf{A})$ only. The matrices $\mathbf{C}$ and $\mathbf{D}$ are independent and uniformly distributed in $\mathbb{F}_q^{m\times m}$, and the adversary's view in gadgets $G_1$, $G_3$, $G_4$ and $G_5$ is independent of them. By Lemma 4, $\mathbf{B} = \mathbf{DAC}$ has the same distribution as $\mathbf{D}'\mathbf{E}_r\mathbf{C}'$, where $\mathbf{D}'$ and $\mathbf{C}'$ are freshly and independently sampled uniformly from $\mathbb{F}_q^{m\times m}$. Thus the simulator can generate $\mathbf{B}$ knowing only $r$. The $t$-SNI security within each iteration of the outer `for` loop follows from the sequential composition of the gadgets $G_1$, $G_2$, $G_3$, $G_4$ and $G_5$, where $G_2, G_3$ are $t$-NI secure and $G_1, G_4, G_5$ are $t$-SNI secure. Since each iteration is independent, the entire `SecLinSolveMatMult` is $t$-SNI secure. □

This concludes the description of our first technique for masked solving linear equations. We will describe in Section 5 the application to the full masking of UOV signatures.

# 4 Masked solving linear equations via multiplicative shares

The first technique operates purely on additive shares. In this section, we present a different route by temporarily converting the additive shares to multiplicative shares, so that we can perform the masked matrix inversion. Section 4.1 formally describes the additive to multiplicative (A2M) conversion, and then we build our masked linear equations system solving on top of it.

## 4.1 Additive to multiplicative conversion

Considering a matrix in $d$ additive shares $\mathbf{A} = \mathbf{A}_1 + \cdots + \mathbf{A}_d \in \mathbb{F}_q^{m \times m}$, we convert it from arithmetic shares to multiplicative shares, as follows:

$$(\mathbf{U}_d \cdots \mathbf{U}_1) \cdot \mathbf{A} \cdot (\mathbf{R}_1 \cdots \mathbf{R}_d) = \mathbf{B}$$

where $\mathbf{U}_i, \mathbf{R}_i \in \mathbb{F}_q^{m \times m^*}$ for all $1 \leq i \leq d$. Our conversion algorithm is similar to [CGTZ23] over $\mathbb{Z}_q[X]/\Phi_\ell$ in the context of the NTRU cryptosystem.

First we generate two random invertible matrices $\mathbf{R}_1, \mathbf{U}_1 \in \mathbb{F}_q^{m \times m^*}$, and compute:

$$\mathbf{U}_1 \cdot \mathbf{A} \cdot \mathbf{R}_1 = \mathbf{U}_1 \cdot \mathbf{A}_1 \cdot \mathbf{R}_1 + \cdots + \mathbf{U}_1 \cdot \mathbf{A}_d \cdot \mathbf{R}_1.$$

Then we perform a mask refreshing on the additive shares $\mathbf{B}_i = \mathbf{U}_1 \cdot \mathbf{A}_i \cdot \mathbf{R}_1$ using the $t$-NI secure `LinearRefreshMasks` from [RP10], see Alg. 7 in App. B.2. We proceed similarly for $(\mathbf{R}_2, \mathbf{U}_2), \ldots, (\mathbf{R}_d, \mathbf{U}_d)$, updating the additive shares $\mathbf{B}_i$. Eventually we obtain:

$$(\mathbf{U}_d \cdots \mathbf{U}_1) \cdot \mathbf{A} \cdot (\mathbf{R}_1 \cdots \mathbf{R}_d) = \mathbf{B}_1 + \cdots + \mathbf{B}_d$$

Thanks to the $d$ times mask refreshing, we can securely recombine the shares $\mathbf{B} = \mathbf{B}_1 + \cdots + \mathbf{B}_d$. We describe the A2M conversion formally in Algorithm 4.

**Algorithm 4** Additive to multiplicative conversion (A2M)

**Input:** shares $\mathbf{A}_1, \ldots, \mathbf{A}_d \in \mathbb{F}_q^{m \times m}$

**Output:** $\mathbf{B} \in \mathbb{F}_q^{m \times m}$, and $(\mathbf{U}_1, \mathbf{R}_1), \ldots, (\mathbf{U}_d, \mathbf{R}_d)$ where $\mathbf{U}_i, \mathbf{R}_i \in \mathbb{F}_q^{m \times m^*}$, such that $(\mathbf{U}_d \cdots \mathbf{U}_1) \cdot \mathbf{A} \cdot (\mathbf{R}_1 \cdots \mathbf{R}_d) = \mathbf{B}$.

1: $(\mathbf{B}_1, \ldots, \mathbf{B}_d) \leftarrow (\mathbf{A}_1, \ldots, \mathbf{A}_d)$
2: **for** $i = 1$ to $d$ **do**
3: $\quad \mathbf{R}_i \leftarrow \mathbb{F}_q^{m \times m^*}$, $\mathbf{U}_i \leftarrow \mathbb{F}_q^{m \times m^*}$ ▷ Generate random invertible matrices
4: $\quad$ **for** $j = 1$ to $d$ **do** $\mathbf{B}_j \leftarrow \mathbf{U}_i \cdot \mathbf{B}_j \cdot \mathbf{R}_i$
5: $\quad (\mathbf{B}_1, \ldots, \mathbf{B}_d) \leftarrow$ `LinearRefreshMasks`$(\mathbf{B}_1, \ldots, \mathbf{B}_d)$ ▷ Refer to Algorithm 7
6: $\mathbf{B} \leftarrow \mathbf{B}_1 + \cdots + \mathbf{B}_d$
7: **return** $\mathbf{B}, (\mathbf{U}_1, \mathbf{R}_1), \ldots, (\mathbf{U}_d, \mathbf{R}_d)$

**Complexity.** The run-time and randomness complexity of A2M are:

$$T_{\texttt{A2M}}(m, d) = d \cdot \left(2m^2 + d \cdot 4(m-2)m^2 + T_{\texttt{LinearRefreshMasks}}(m, d)\right) = \mathcal{O}(m^3 d^2),$$
$$R_{\texttt{A2M}}(m, d) = d \cdot \left(2m^2 + R_{\texttt{LinearRefreshMasks}}(m, d)\right) = \mathcal{O}(m^2 d^2).$$

**Security.** The following Lemma shows that the A2M conversion achieves $t$-SNI security.

**Lemma 6** ($t$-SNI security of A2M conversion (Algorithm 4))**.** *The* A2M *takes as input $d$ arithmetic shares $\mathbf{A}_i$ for $1 \leq i \leq d$, and outputs $\mathbf{B}$, $(\mathbf{U}_i, \mathbf{R}_i)$ for $1 \leq i \leq d$ such that $\mathbf{B} = \mathbf{U}_d \cdots \mathbf{U}_1 \cdot (\mathbf{A}_1 + \cdots + \mathbf{A}_d) \cdot \mathbf{R}_1 \cdots \mathbf{R}_d$. For any set of $t_1$ intermediate variables and any subset $O$ of output indices, with $t_1 + |O| \leq t < d$, the $t_1$ intermediate variables, the output variable $\mathbf{B}$, and the output variables $\left(\mathbf{U}_{|O}, \mathbf{R}_{|O}\right)$ can be perfectly simulated from the input variables $\mathbf{A}_{|J}$ with $|J| \leq t_1$, when* $\mathrm{rank}(\mathbf{A})$ *is given to the simulator.*

*Proof.* We denote $\texttt{loop}_i$ for $1 \leq i \leq d$ the steps of the algorithm from Line 2 to Line 5 in the `for` loop with index $i$, by $\mathbf{B}_j^{(i)}$ the value of share $\mathbf{B}_j$ at the end of $\texttt{loop}_i$, and by $\mathbf{B}^{(i^\star)}$ the sum of all shares $\mathbf{B}_j^{(i^\star)}$. We construct a subset $J \subset [1, d]$ of input indices, which is initially empty. For any probed input variable $\mathbf{A}_j$ and for any probed intermediate variable $\mathbf{B}_j$ at $\texttt{loop}_i$, we add the index $j$ to $J$. By construction we have $|J| \leq t_1$ as required. Let

$$P = \{i \,|\, \texttt{loop}_i \text{ has been probed, or } i \in O\}.$$

From $t_1 + |O| \leq t < d$, we deduce $P \subsetneq [1, d]$ and therefore there exists $i^\star$ such that $\texttt{loop}_{i^\star}$ has not been probed and $i^\star \notin O$.

We now show that any probed variables can be perfectly simulated from $\mathbf{A}_{|J}$ and the knowledge of $\text{rank}(\mathbf{A})$, and moreover $\mathbf{B}$ and $\left(\mathbf{U}_{|O}, \mathbf{R}_{|O}\right)$ can be perfectly simulated. For every probed variable in $\texttt{loop}_i$ with $i < i^\star$, the simulation of all variables with index $j$ is straightforward from $\mathbf{A}_{|J}$; by definition, this includes all probed variables corresponding to $i < i^\star$.

We now show that we can simulate all shares $\mathbf{B}_j^{(i^\star)}$ at the output of $\texttt{loop}_{i^\star}$. For this, we proceed with a sequence of two games.

**Game 1.** In the first game, the simulation is performed using the knowledge of $\mathbf{A}$. Since by assumption $\texttt{loop}_{i^\star}$ has not been probed, the corresponding `LinearRefreshMasks` gadget has not been probed. Therefore, we can perfectly simulate all output shares $\mathbf{B}_j^{(i^\star)}$ from $\mathbf{B}^{(i^\star)}$. We compute $\mathbf{B}^{(i^\star)}$ as

$$\mathbf{B}^{(i^\star)} = \mathbf{U}_{i^\star} \cdot \mathbf{B}^{(i^\star - 1)} \cdot \mathbf{R}_{i^\star} \tag{2}$$

where $\mathbf{B}^{(i^\star - 1)} = (\mathbf{U}_{i^\star - 1} \cdots \mathbf{U}_1) \cdot \mathbf{A} \cdot (\mathbf{R}_1 \cdots \mathbf{R}_{i^\star - 1})$.

**Game 2.** In the second game, the simulation is performed using the knowledge of $\text{rank}(\mathbf{A})$ only. Instead of writing $\mathbf{B}^{(i^\star)} = \mathbf{U}_{i^\star} \cdot \mathbf{B}^{(i^\star - 1)} \cdot \mathbf{R}_{i^\star}$ where $\mathbf{B}^{(i^\star - 1)}$ is computed from $\mathbf{A}$, we replace $\mathbf{B}^{(i^\star - 1)}$ with the same matrix $\mathbf{E}_r$ as defined in the proof of Lemma 5, for $r = \text{rank}(\mathbf{A})$, and we compute:

$$\mathbf{B}^{(i^\star)'} = \mathbf{U}_{i^\star} \cdot \mathbf{E}_r \cdot \mathbf{R}_{i^\star}\,.$$

This terminates the description of Game 2.

Now we show that Game 1 and Game 2 are equivalent, which boils down to prove that $\mathbf{B}^{(i^\star)}$ and $\mathbf{B}^{(i^\star)'}$ have the same distribution. As in the proof of Lemma 5, by Gaussian elimination, there exists two invertible matrices $\mathbf{P}, \mathbf{Q} \in \mathbb{F}_q^{m \times m^*}$ such that $\mathbf{B}^{(i^\star - 1)} = \mathbf{P} \cdot \mathbf{E}_r \cdot \mathbf{Q}$. From (2) we can write:

$$\mathbf{B}^{(i^\star)} = (\mathbf{U}_{i^\star} \cdot \mathbf{P}) \cdot \mathbf{E}_r \cdot (\mathbf{Q} \cdot \mathbf{R}_{i^\star})\,.$$

By assumption $(\mathbf{U}_{i^\star}, \mathbf{R}_{i^\star})$ has not been probed and is not used elsewhere in the computation. Moreover multiplication by an invertible matrix is a bijective map, therefore $\mathbf{B}^{(i^\star)}$ and $\mathbf{B}^{(i^\star)'}$ have the same distribution. Therefore Game 1 and Game 2 are equivalent, and we can perform the simulation using $\text{rank}(\mathbf{A})$ only.

Since we can simulate all shares $\mathbf{B}_j^{(i^\star)}$ at the output of $\texttt{loop}_{i^\star}$, we can simulate all variables in $\texttt{loop}_i$ for index $i > i^\star$. Therefore $\mathbf{B}$ can also be simulated, which completes the proof. □

## 4.2 Masked linear equations system solving using `A2M` conversion

In the following, we describe how to solve the linear system $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$ using masked matrix inversion based on the previously introduced algorithm. Given an additively masked matrix

$\mathbf{A} = \mathbf{A}_1 + \cdots + \mathbf{A}_d$, we first generate multiplicative shares $(\mathbf{U}_i, \mathbf{R}_i)$, where each $\mathbf{U}_i$ and $\mathbf{R}_i$ is a random invertible matrix. We then compute

$$\mathbf{B} = \mathbf{U}_d \cdots \mathbf{U}_1 \cdot (\mathbf{A}_1 + \cdots + \mathbf{A}_d) \cdot \mathbf{R}_1 \cdots \mathbf{R}_d.$$

From Lemma 6, the unmasking of $\mathbf{B}$ does not leak information about $\mathbf{A}$ except its rank. Therefore, if $\mathbf{B}$ is invertible, which is equivalent to $\mathbf{A}$ being invertible, its inverse can be computed in the clear. This gives:

$$\mathbf{B}^{-1} = \mathbf{R}_d^{-1} \cdots \mathbf{R}_1^{-1} \cdot (\mathbf{A}_1 + \cdots + \mathbf{A}_d)^{-1} \cdot \mathbf{U}_1^{-1} \cdots \mathbf{U}_d^{-1}.$$

Then the multiplicative masking of $\mathbf{A}^{-1}$ can be obtained by:

$$\mathbf{A}^{-1} = \mathbf{R}_1 \cdots \mathbf{R}_d \cdot \mathbf{B}^{-1} \cdot \mathbf{U}_d \cdots \mathbf{U}_1$$

Finally, if $\mathbf{b}$ is additively masked, we can get the solution $\mathbf{x}$ also in additive shares as:

$$\mathbf{x} = \mathbf{A}^{-1} \cdot \mathbf{b} = \mathbf{R}_1 \cdots \mathbf{R}_d \cdot \mathbf{B}^{-1} \cdot \mathbf{U}_d \cdots \mathbf{U}_1 \cdot \mathbf{b}$$

which we implement via the share-wise matrix-vector multiplication `halfSecMatVecMult` (see Alg. 6 in Appendix B.1). We provide the complete procedure as `SecLinSolveA2M` in Algorithm 5 below.

**Algorithm 5** `SecLinSolveA2M`

**Input:** shares $\mathbf{A}_1, \ldots, \mathbf{A}_d \in \mathbb{F}_q^{m \times m}$, and shares $\mathbf{b}_1, \ldots, \mathbf{b}_d \in \mathbb{F}_q^m$.
**Output:** shares $\mathbf{x}_1, \ldots, \mathbf{x}_d \in \mathbb{F}_q^m$ satisfying $(\sum_{i=1}^d \mathbf{A}_i) \cdot (\sum_{i=1}^d \mathbf{x}_i) = \sum_{i=1}^d \mathbf{b}_i$ or $\perp$

$(\mathbf{B}, (\mathbf{U}_1, \mathbf{R}_1), \ldots (\mathbf{U}_d, \mathbf{R}_d)) \leftarrow$ `A2M`$(\mathbf{A}_1, \ldots, \mathbf{A}_d)$
**if** $\mathbf{B}$ is invertible **then**
  $(\mathbf{x}_1, \ldots, \mathbf{x}_d) \leftarrow (\mathbf{b}_1, \ldots, \mathbf{b}_d)$
  **for** $i = 1$ to $d$ **do**
    $(\mathbf{x}_1, \ldots, \mathbf{x}_d) \leftarrow$ `halfSecMatVecMult`$(\mathbf{U}_i, (\mathbf{x}_1, \ldots, \mathbf{x}_d))$ ▷ Refer to Algorithm 6
    $(\mathbf{x}_1, \ldots, \mathbf{x}_d) \leftarrow$ `LinearRefreshMasks`$(\mathbf{x}_1, \ldots, \mathbf{x}_d)$
  $(\mathbf{x}_1, \ldots, \mathbf{x}_d) \leftarrow$ `halfSecMatVecMult`$(\mathbf{B}^{-1}, (\mathbf{x}_1, \ldots, \mathbf{x}_d))$
  **for** $i = d$ to $1$ **do**
    $(\mathbf{x}_1, \ldots, \mathbf{x}_d) \leftarrow$ `halfSecMatVecMult`$(\mathbf{R}_i, (\mathbf{x}_1, \ldots, \mathbf{x}_d))$
    $(\mathbf{x}_1, \ldots, \mathbf{x}_d) \leftarrow$ `LinearRefreshMasks`$(\mathbf{x}_1, \ldots, \mathbf{x}_d)$
  **return** $\mathbf{x}_1, \ldots, \mathbf{x}_d$
**else return** $\perp$

**Complexity.** The run-time and randomness complexity of `SecLinSolveA2M` are:

$$\begin{aligned} T_{\texttt{SecLinSolveA2M}}(m,d) =& T_{\texttt{A2M}}(m,d) + 2m^3 + (2d+1) \cdot T_{\texttt{halfSecMatVecMult}}(m,d) + \\ & d \cdot T_{\texttt{LinearRefreshMasks}}(m,d) = \mathcal{O}(m^3 d^2), \\ R_{\texttt{SecLinSolveA2M}}(m,d) =& R_{\texttt{A2M}}(m,d) + d \cdot R_{\texttt{LinearRefreshMasks}}(m,d) = \mathcal{O}(m^2 d^2). \end{aligned}$$

Therefore, our second technique has the same time and randomness complexities as the additive share technique from Section 3. As previously, relative to masked Gaussian elimination [NKO+25], our methods reduce the randomness complexity from $\mathcal{O}(m^3 d^2)$ to $\mathcal{O}(m^2 d^2)$.

**Security.** The following Lemma shows that the `SecLinSolveA2M` algorithm achieves the $t$-SNI security.

**Lemma 7** ($t$-SNI security of `SecLinSolveA2M` (Algorithm 5))**.** *For any set of $t_1$ intermediate variables and any subset $O$ of output indices, with $t_1 + |O| \leq t < d$, the $t_1$ intermediate variables and the output variables $\mathbf{x}_{|O}$ can can be perfectly simulated from the input variables $\mathbf{A}_{|I}$ and $\mathbf{b}_{|I}$ with $|I| \leq t_1$, when $\mathrm{rank}(\mathbf{A})$ is given to the simulator.*

*Proof.* We denote the `A2M` conversion at Line 1 as $G_1$, the first `for` loop from Line 4 to Line 6 as $G_2$, the `halfSecMatVecMult` at Line 7 as $G_3$, and the second `for` loop from Line 8 to Line 10 as $G_4$. We first prove that $G_2$ is $t$-SNI secure. It follows from the fact that since at most $d-1$ intermediate variables have been probed, in the `for` loop at least one of the `halfSecMatVecMult` and subsequent `LinearRefreshMasks` has not been probed, for some index $i^\star$. We can therefore simulate all $\mathbf{x}_i$ except one at the end of $\texttt{loop}_{i^\star}$ with random values from $\mathbb{F}_q^m$. Such simulation can be propagated until the end of the loop. This concludes the $t$-SNI security of $G_2$. The same reasoning applies to $G_4$. Therefore, the $t$-SNI security of `SecLinSolveA2M` is achieved by serial composition of gadgets $G_{1-4}$, where $G_1, G_2, G_4$ are $t$-SNI secure, and $G_3$ is $t$-NI secure. □

# 5 Masking UOV signature generation

In this section, we introduce the high-order masking of UOV signature generation. Our masking strategy follows [KNK+25], i.e., we mask the same variables and leave the same variables unprotected.

We recall in Figure 2 the detailed UOV signing procedure. To sign a given message $\mu$, we first hash it with a random salt byte to obtain the target vector $\mathbf{t}$. Then we *deterministically* generate the vinegar vector $\mathbf{v}$, and insert it into the linear equations system $\mathbf{L}\mathbf{x} = \mathbf{t} - \mathbf{y}$, where $\mathbf{y}$ is the quadratic evaluation of $\mathbf{P}_i$ (part of the public key) at $\begin{pmatrix}\mathbf{v} & \mathbf{0}_m\end{pmatrix}$. If $\mathbf{L}$ is not invertible, we restart and generate a new $\mathbf{v}$. By solving this linear equations system, we get the coefficients of the oil vector $\mathbf{o}$, and the preimage of $\mathbf{t}$ can be computed as $\mathbf{s} = \begin{pmatrix}\mathbf{v} & \mathbf{0}_m\end{pmatrix} + \overline{\mathbf{O}}\mathbf{x}$.

**UOV.Sign** $\left(\texttt{esk} = \left(\texttt{seed}_{\texttt{sk}}, \mathbf{O}, \{\mathbf{P}_i^{(1)}, \mathbf{S}_i\}_{i\in[m]}\right), \mu\right)$:

1: $\texttt{salt} \leftarrow \{0,1\}^{\texttt{salt_len}}$
2: $\mathbf{t} \leftarrow \texttt{Hash}(\mu||\texttt{salt})$
3: **for** ctr $= 0$ upto $255$ **do**
4: $\quad \mathbf{v} := \texttt{Expand}_{\mathbf{v}}(\mu||\texttt{salt}||\texttt{seed}_{\texttt{sk}}||\texttt{ctr})$
5: $\quad \mathbf{L} := \mathbf{0}_{m\times m}$
6: $\quad$ **for** $i = 1$ upto $m$ **do**
7: $\quad\quad$ Set $i$-th row of $\mathbf{L}$ to $\mathbf{v}^\top \mathbf{S}_i$
8: $\quad$ **if** $\mathbf{L}$ is invertible **then**
9: $\quad\quad \mathbf{y} \leftarrow \left[\mathbf{v}^\top \mathbf{P}_i^{(1)} \mathbf{v}\right]_{i\in[m]}$
10: $\quad\quad$ Solve $\mathbf{L}\mathbf{x} = \mathbf{t} - \mathbf{y}$ for $\mathbf{x}$
11: $\quad\quad \mathbf{s} := \begin{bmatrix}\mathbf{v}\\ \mathbf{0}_m\end{bmatrix} + \left[\mathbf{O}^\top \mathbf{I}_m\right] \cdot \mathbf{x}$
12: $\quad\quad \sigma := (\mathbf{s}, \texttt{salt})$
13: $\quad\quad$ **return** $\sigma$
14: **return** $\perp$

Figure 2: Detailed signing procedure of UOV scheme.

We now summarize below the high-order masking of UOV signing operations.

1. No masking for Line 1 and Line 2 are needed, since the target vector $\mathbf{t}$ is derived from the message $\mu$ and salt, which are both public.

2. At Line 4, the generation of the vinegar vector $\mathbf{v}$ requires side-channel protection since it is derived from the secret key seed $\texttt{seed}_{\texttt{sk}}$. We instantiate this $\texttt{Expand}_{\mathbf{v}}$ as `masked_shake256`, which we take from the masked Kyber [CGMZ23].

3. At Line 7, both $\mathbf{v}$ and $\mathbf{S}_i$ are sensitive and masked, thus its computation needs to be carried out in shares. We form the shared matrix $\mathbf{L}$ in column-wise instead of row-wise, i.e., we reinterpret it as $\mathbf{L}^\top$. In this case, instead of setting each row of $\mathbf{L}$ to $\mathbf{v}^\top\mathbf{S}_i$, we compute each column of $\mathbf{L}$ as $\mathbf{S}_i^\top\mathbf{v}$, and this can be done using the masked matrix-vector multiplication.

4. At Line 9, we perform the masked quadratic evaluation of $\mathbf{P}_i^{(1)}$ at $\mathbf{v}$. We first multiply the public $\mathbf{P}_i^{(1)}$ with masked $\mathbf{v}$ to obtain each column of the intermediate matrix, which we denote as $\mathbf{T}$. Note that to keep each input variable independent, we must apply a $t$-SNI `RefreshMasks` to $\mathbf{v}$ before we use it again. Then for the $\mathbf{v}^\top\mathbf{T}$, we transpose it to $\mathbf{T}^\top\mathbf{v}$ so that `SecMatVecMult` can be applied. To form the constant part of the linear equations system, after obtaining the masked $\mathbf{y}$, we need to subtract the first share of $\mathbf{y}$ with the public $\mathbf{t}$. Finally, we get $\mathbf{t} - \mathbf{y}$ in shares.

5. At Line 10, this linear equations system $\mathbf{L}\mathbf{x} = \mathbf{t} - \mathbf{y}$ can be solved using either of our two techniques described in Section 3 and Section 4. Since the solution $\mathbf{x}$ will be included in the signature, we can securely unmask it after a full refresh.

6. At Line 11, we can finally compute the preimage of $\mathbf{t}$. $\begin{bmatrix}\mathbf{O}^\top & \mathbf{I}_m\end{bmatrix} \cdot \mathbf{x}$ can be computed in a share-wise manner since $\mathbf{x}$ is in the clear. Then we add it to the vinegar vector share by share. After a $t$-SNI refresh, we can securely unmask $\mathbf{s}$, which is part of the signature.

# 6 Evaluation

We carried out benchmarking experiments to evaluate the practical performance at high-order of our masked linear equations system solving. The experiments were conducted on an Intel Xeon Gold 6128 CPU clocked at 3.4 GHz, and no TurboBoost was enabled during measurements. The source code of masked Gaussian elimination [NKO+25], Kundu's masked UOV[3] [KNK+25], and ours were compiled using `gcc 13.3`, with optimization flag `-O3`. Our masked implementation is built directly on top of the UOV reference implementation and therefore inherits most of its implementation-level optimizations. Kundu's implementation, on the other hand, appears to focus on demonstrating feasibility and does not apply the same optimization techniques. The details of such implementation techniques are beyond the scope of this paper, we refer the reader to [BCD+23] for more information. Our open source implementation can be found at

https://github.com/zh-bw/pqov-masked

## 6.1 Comparison of masked linear equations system solving

Table 2 presents the randomness usage of three techniques for solving the linear equations system for security order up to $t = 6$, across three UOV parameter sets, namely `UOV-Ip`, `UOV-III`, and `UOV-V`. We counted actual PRNG calls and converted them into kilobytes of randomness. Both of our techniques significantly outperform the masked Gaussian elimination [NKO+25] in terms of randomness usage. Starting at $t = 2$, we reduce the randomness usage by at least 90%. For `UOV-V` parameter, the reduction can reach up to

[3] It uses masked Gaussian elimination in the signature generation.

98%. This confirms the theoretical improvement in randomness complexity, which we slash from $\mathcal{O}(m^3d^2)$ to $\mathcal{O}(m^2d^2)$, with $d = t + 1$.

Table 2: Randomness usage (in KB) for masked linear equations system solving across three parameter sets, namely `UOV-Ip`, `UOV-III`, and `UOV-V`, up to security order $t = 6$. Numbers in parentheses indicate the percentage reduction in randomness compared to masked Gaussian elimination [NKO+25].

| Parameters | | Security order $t$ | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 |
| UOV-Ip | masked GE [NKO+25] | 948 | 2844 | 5674 | 9479 | 14218 | 19906 |
| | SecLinSolveMatMult | 133 (−86%) | 286 (−90%) | 497 (−91%) | 765 (−92%) | 1091 (−92%) | 1474 (−93%) |
| | SecLinSolveA2M | 78 (−92%) | 157 (−95%) | 276 (−95%) | 435 (−95%) | 633 (−96%) | 871 (−96%) |
| UOV-III | masked GE [NKO+25] | 3953 | 11859 | 23718 | 39530 | 58926 | 83012 |
| | SecLinSolveMatMult | 391 (−90%) | 764 (−94%) | 1324 (−95%) | 2039 (−95%) | 3195 (−95%) | 3928 (−95%) |
| | SecLinSolveA2M | 206 (−95%) | 415 (−97%) | 728 (−97%) | 1145 (−97%) | 1665 (−97%) | 2296 (−97%) |
| UOV-V | masked GE [NKO+25] | 9186 | 27335 | 55114 | 82814 | 133704 | 179247 |
| | SecLinSolveMatMult | 632 (−93%) | 1356 (−95%) | 2351 (−96%) | 3619 (−96%) | 5158 (−96%) | 6969 (−96%) |
| | SecLinSolveA2M | 365 (−96%) | 733 (−97%) | 1285 (−98%) | 2021 (−98%) | 2950 (−98%) | 4045 (−98%) |

We further evaluate the concrete performance of all techniques in thousands of CPU cycles, and the results appear in Table 3. Since our techniques have better randomness complexity, the choice of PRNG (i.e., how many clock cycles are required for generating one byte randomness) will affect the comparison results. In this case, for a fair comparison, we align ours to use the same PRNG as in [NKO+25] and Kundu's implementation [KNK+25]. More precisely, we apply PRNG from the standard C library which takes 175 cycles per byte. Moreover, we benchmark with a `xorshift` PRNG (taking 18 cycles per byte) to simulate the situation where the randomness generation is fast. Those randomness generators are not cryptographically secure, and we use them only for benchmarking purposes, but it can be easily replaced with a secure PRNG in practice.

Per Table 3, when PRNG is slow, we achieve an average speedup of 3.0× (resp. 2.9×) for `SecLinSolveA2M` (resp. `SecLinSolveMatMult`) compared to masked Gaussian elimination [NKO+25] at high-order under `UOV-Ip` parameter. As the dimension of the oil space increases, the average speedup becomes more significant, reaching 3.6× (resp., 3.4×) under `UOV-III` and 4.2× (resp., 4.0×) under `UOV-V` parameters. Our two techniques exhibit very similar performance at low security orders, while for $t > 2$ `SecLinSolveA2M` is more efficient.

Table 3: Benchmarks (in thousands of cycles) of masked linear equations system solving on Intel Xeon Gold 6128 @ 3.4 GHz, for different PRNG speeds and parameter sets (`UOV-Ip`, `UOV-III`, `UOV-V`), with security order up to $t = 6$. Numbers in parentheses denote the speedup factor in cycle count compared to masked Gaussian elimination [NKO+25].

| Parameters | PRNG speed | | Security order $t$ | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | 1 | 2 | 3 | 4 | 5 | 6 |
| `UOV-Ip` | fast | masked GE [NKO+25] | 5772 | 13,060 | 18,718 | 28,813 | 40,721 | 59,473 |
| | | `SecLinSolveMatMult` | 5377 (1.1×) | 11,304 (1.1×) | 17,180 (1.1×) | 26,040 (1.1×) | 36,638 (1.1×) | 49,590 (1.2×) |
| | | `SecLinSolveA2M` | 5897 (1.0×) | 11,675 (1.1×) | 17,411 (1.1×) | 25,936 (1.1×) | 36,344 (1.1×) | 48,705 (1.2×) |
| | slow | masked GE [NKO+25] | 10,914 | 26,560 | 58,296 | 82,916 | 123,936 | 168,961 |
| | | `SecLinSolveMatMult` | 6167 (1.8×) | 11,917 (2.2×) | 20,028 (2.9×) | 30,576 (2.7×) | 43,478 (2.9×) | 57,957 (2.9×) |
| | | `SecLinSolveA2M` | 6485 (1.7×) | 11,514 (2.2×) | 19,061 (3.1×) | 28,534 (2.9×) | 40,038 (3.1×) | 53,523 (3.2×) |
| `UOV-III` | fast | masked GE [NKO+25] | 23,352 | 46,718 | 77,520 | 120,999 | 172,246 | 233,778 |
| | | `SecLinSolveMatMult` | 19,735 (1.2×) | 36,494 (1.3×) | 60,191 (1.3×) | 90,070 (1.3×) | 127,185 (1.4×) | 172,283 (1.4×) |
| | | `SecLinSolveA2M` | 21,475 (1.1×) | 37,963 (1.2×) | 61,008 (1.3×) | 90,686 (1.3×) | 126,953 (1.4×) | 169,945 (1.4×) |
| | slow | masked GE [NKO+25] | 41,244 | 111,009 | 210,499 | 340,959 | 505,936 | 698,396 |
| | | `SecLinSolveMatMult` | 21,801 (1.9×) | 40,619 (2.7×) | 67,440 (3.1×) | 102,538 (3.3×) | 143,989 (3.5×) | 192,540 (3.6×) |
| | | `SecLinSolveA2M` | 22,702 (1.8×) | 40,365 (2.7×) | 65,235 (3.2×) | 97,341 (3.5×) | 136,322 (3.7×) | 182,575 (3.8×) |
| `UOV-V` | fast | masked GE [NKO+25] | 47,268 | 114,326 | 184,792 | 283,308 | 402,382 | 546,860 |
| | | `SecLinSolveMatMult` | 41,279 (1.1×) | 74,547 (1.5×) | 120,824 (1.5×) | 182,693 (1.6×) | 252,561 (1.6×) | 339,604 (1.6×) |
| | | `SecLinSolveA2M` | 45,424 (1.0×) | 78,005 (1.5×) | 123,889 (1.5×) | 179,733 (1.6×) | 255,021 (1.6×) | 339,947 (1.6×) |
| | slow | masked GE [NKO+25] | 96,126 | 267,187 | 491,159 | 793,459 | 1,173,183 | 1,627,318 |
| | | `SecLinSolveMatMult` | 45,126 (2.1×) | 82,586 (3.2×) | 133,728 (3.7×) | 201,155 (3.9×) | 282,939 (4.1×) | 381,568 (4.3×) |
| | | `SecLinSolveA2M` | 47,596 (2.0×) | 82,286 (3.2×) | 131,232 (3.7×) | 194,261 (4.1×) | 271,473 (4.3×) | 362,812 (4.5×) |

When PRNG is fast, our techniques still both outperform masked GE at every security order. Although all three techniques retain the same level of run-time complexity $\mathcal{O}(m^3d^2)$, our techniques require fewer operations in practice, and `SecLinSolveMatMult` is the most efficient one. Based on the above observations, we conclude that when the randomness generation is not a bottleneck, `SecLinSolveMatMult` is the preferable choice, while `SecLinSolveA2M` is more efficient when the randomness generation is costly. We could envision that on some embedded devices where running a True Random Number Generator (TRNG) can be very expensive, the benefits of our techniques will be even greater.

## 6.2 Implementation results of fully masked UOV signing

We integrate `SecLinSolveA2M` into our masked UOV signing, and benchmark it against the unprotected signing (i.e., $t = 0$) across three parameter sets, namely `UOV-Ip`, `UOV-III`, and `UOV-V`. We instantiate the signature scheme with the `pkc` variant, while we claim that our implementation could be easily extended to other variants as well. Table 4 displays the cycle count for signing up to $t = 6$. The overhead introduced by masking countermeasures appears to be identical across three parameter sets, though it decreases slightly as the dimension of the oil space grows.

Table 4: Benchmarks (in thousands of cycles) on Intel Xeon Gold 6128 @ 3.4 GHz of UOV signing across three parameter sets (i.e., `UOV-Ip`, `UOV-III`, and `UOV-V`), for security order up to $t = 6$.

| Parameters | | Security order $t$ | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | Ref. | 1 | 2 | 3 | 4 | 5 | 6 |
| `UOV-Ip` | | 1385 | 9885 | 19,580 | 32,442 | 49,678 | 69,295 | 92,769 |
| | Overhead | 1.0× | 7.1× | 14.1× | 23.4× | 35.9× | 50.0× | 67.0× |
| `UOV-III` | | 5411 | 36,210 | 70,440 | 116,677 | 174,832 | 241,665 | 326,422 |
| | Overhead | 1.0× | 6.7× | 13.0× | 21.6× | 32.3× | 44.7× | 60.3× |
| `UOV-V` | | 11,986 | 76,979 | 152,146 | 245,430 | 367,996 | 510,713 | 683,891 |
| | Overhead | 1.0× | 6.4× | 12.7× | 20.5× | 30.7× | 42.6× | 57.1× |

**Comparison with [KNK+25].** Table 5 shows the benchmarking results of our masked UOV against Kundu's implementation [KNK+25] under `UOV-Ip` parameters. At higher security orders ($t > 2$), our implementation achieves an average speedup of 4.0× compared to Kundu's work.

Table 5: Benchmarks (in thousands of cycles) on Intel Xeon Gold 6128 @ 3.4 GHz of fully masked UOV signing for security order up to $t = 6$ under `UOV-Ip` parameters. Numbers in parentheses denote the speedup factor compared to Kundu's implementation [KNK+25].

| | Security order $t$ | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| [KNK+25] | 19,439 | 75,092 | 126,592 | 202,610 | 286,020 | 341,230 |
| This work | 9885 (2.0×) | 19,580 (3.8×) | 32,442 (3.9×) | 49,678 (4.1×) | 69,295 (4.1×) | 92,769 (3.7×) |

# 7 Conclusion

UOV-based signature schemes are promising candidates for post-quantum digital signatures, offering compact signatures and strong resistance against algebraic attacks. In this work, we introduced two high-order masking techniques for secure matrix inversion: one based on additive sharing and the other on multiplicative sharing. We formally proved the security of both constructions in the $t$-probing model.

Compared to masked Gaussian elimination [NKO+25], our techniques reduce the randomness complexity from $\mathcal{O}(m^3d^2)$ to $\mathcal{O}(m^2d^2)$, leading to significant improvements in practical performance. As an application, we developed a C implementation of masked UOV signing using our proposed techniques and achieved consistently better performance than Kundu's recent implementation [KNK+25] across all security orders. Beyond UOV, our methods are generic and can be applied to related multivariate schemes such as MAYO, QR-UOV, and SNOVA, with some slight modifications to accommodate their specific field arithmetics.

## Acknowledgements

Bowen Zhang is funded by the ERC Advanced Grant number 101096871 (acronym BRIDGE). Views and opinions expressed in the paper are those of the authors and do not necessarily reflect those of the European Union nor the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

## AI statement

AI tools were used to improve the writing of this paper and to generate the Lean proof of Lemma 4.

# References

[ACK+23] Thomas Aulbach, Fabio Campos, Juliane Krämer, Simona Samardjiska, and Marc Stöttinger. Separating oil and vinegar with a single trace: side-channel assisted kipnis-shamir attack on uov. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(3):221–245, 2023. https://doi.org/10.46586/tches.v2023.i3.221-245.

[ACK25] Thomas Aulbach, Fabio Campos, and Juliane Krämer. Sok: On the physical security of uov-based signature schemes. In *International Conference on Post-*

*Quantum Cryptography*, pages 199–231. Springer, 2025. https://doi.org/10.1007/978-3-031-86599-2_7.

[BBD+16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 116–129, 2016. https://dl.acm.org/doi/abs/10.1145/2976749.2978427.

[BCC+23] Ward Beullens, Fabio Campos, Sofía Celi, Basil Hess, and Matthias J. Kannwischer. Mayo, 2023. https://pqmayo.org/.

[BCD+23] Ward Beullens, Ming-Shing Chen, Jintai Ding, Boru Gong, Matthias J. Kannwischer, Jacques Patarin, Bo-Yuan Peng, Dieter Schmidt, Cheng-Jhih Shih, Chengdong Tao, and Bo-Yin Yang. Uov: Unbalanced oil and vinegar, 2023. https://www.uovsig.org/.

[BCH+23] Ward Beullens, Ming-Shing Chen, Shih-Hao Hung, Matthias J Kannwischer, Bo-Yuan Peng, Cheng-Jhih Shih, and Bo-Yin Yang. Oil and vinegar: Modern parameters and implementations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(3):321–365, 2023. https://doi.org/10.46586/tches.v2023.i3.321-365.

[Beu22] Ward Beullens. Breaking rainbow takes a weekend on a laptop. In *Annual International Cryptology Conference*, pages 464–479. Springer, 2022. https://doi.org/10.1007/978-3-031-15979-4_16.

[CGMZ23] Jean-Sébastien Coron, François Gérard, Simon Montoya, and Rina Zeitoun. High-order polynomial comparison and masking lattice-based encryption. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 153–192, 2023. https://tches.iacr.org/index.php/TCHES/article/view/9950.

[CGTZ23] Jean-Sébastien Coron, François Gérard, Matthias Trannoy, and Rina Zeitoun. High-order masking of NTRU. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(2):180–211, 2023. https://tches.iacr.org/index.php/TCHES/article/view/10281.

[CKY21] Tung Chou, Matthias J Kannwischer, and Bo-Yin Yang. Rainbow on cortex-m4. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 650–675, 2021. https://doi.org/10.46586/tches.v2021.i4.650-675.

[Cor14] Jean-Sébastien Coron. Higher order masking of look-up tables. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 441–458. Springer, 2014. https://doi.org/10.1007/978-3-642-55220-5_25.

[CS21] Jean-Sébastien Coron and Lorenzo Spignoli. Secure wire shuffling in the probing model. In *Annual International Cryptology Conference*, pages 215–244. Springer, 2021. https://doi.org/10.1007/978-3-030-84252-9_8.

[DS05] Jintai Ding and Dieter Schmidt. Rainbow, a new multivariable polynomial signature scheme. In *International conference on applied cryptography and network security*, pages 164–175. Springer, 2005. https://doi.org/10.1007/11496137_12.

[FIH+23] Hiroki Furue, Yasuhiko Ikematsu, Fumitaka Hoshino, Tsuyoshi Takagi, Haruhisa Kosuge, Kimihiro Yamakoshi, Rika Akiyama, Satoshi Nakamura, Shingo Orihara, and Koha Kinjo. Qr-uov, 2023. https://info.isl.ntt.co.jp/crypt/qruov/.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. 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*, pages 463–481. Springer, 2003. https://doi.org/10.1007/978-3-540-45146-4_27.

[JD24] Sönke Jendral and Elena Dubrova. Single-trace side-channel attacks on mayo exploiting leaky modular multiplication. *Cryptology ePrint Archive*, 2024. https://eprint.iacr.org/2024/1850.

[KNK+25] Suparna Kundu, Quinten Norga, Angshuman Karmakar, Uttam Kumar Ojha, Anindya Ganguly, and Ingrid Verbauwhede. muov: Masking the unbalanced oil and vinegar digital signature scheme at first-and higher-order. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, pages 1994–2008, 2025. https://doi.org/10.1145/3719027.3765188.

[KPG99] Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced oil and vinegar signature schemes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 206–222. Springer, 1999. https://doi.org/10.1007/3-540-48910-X_15.

[NKO+25] Quinten Norga, Suparna Kundu, Uttam Kumar Ojha, Anindya Ganguly, Angshuman Karmakar, and Ingrid Verbauwhede. Masking gaussian elimination at arbitrary order with application to multivariate-and code-based pqc. In *Cryptographers' Track at the RSA Conference*, pages 249–272. Springer, 2025. https://doi.org/10.1007/978-3-031-88661-4_11.

[Pat97] Jacques Patarin. The oil and vinegar signature scheme. In *Presented at the Dagstuhl Workshop on Cryptography September 1997*, 1997.

[PSKH18] Aesun Park, Kyung-Ah Shim, Namhun Koo, and Dong-Guk Han. Side-channel attacks on post-quantum signature schemes based on multivariate quadratic equations:-rainbow and uov. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 500–523, 2018. https://doi.org/10.13154/tches.v2018.i3.500-523.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of aes. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 413–427. Springer, 2010. https://doi.org/10.1007/978-3-642-15031-9_28.

[SLK22] Kyung-Ah Shim, Sangyub Lee, and Namhun Koo. Efficient implementations of rainbow and uov using avx2. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 245–269, 2022. https://doi.org/10.46586/tches.v2022.i1.245-269.

[WTKC23] Lih-Chung Wang, Po-En Tseng, Yen-Liang Kuan, and Chun-Yen Chou. Snova, 2023. https://snova.pqclab.org/.

# A Proof of Lemma 1

*Proof.* Our proof strategy is adapted from [BBD+16]. We first categorize the probed intermediate variables into four groups:

- Group 1: If $\mathbf{A}_i, \mathbf{b}_i$ or $\mathbf{A}_i \cdot \mathbf{b}_i$ is probed, we add index $i$ to $I$ and $J$.
- Group 2: If $\mathbf{r}_{i,j}$ or $\mathbf{x}_{i,j}$ (for $i \neq j$) is probed, we add index $i$ to $I$ and $J$.
- Group 3: $\mathbf{A}_i \cdot \mathbf{b}_j + \mathbf{r}_{i,j}$ is probed. In this case, if $i$ or $j$ belongs to both $I$ and $J$, we add $\{i, j\}$ to both $I$ and $J$.
- Group 4: If $\mathbf{A}_i \cdot \mathbf{b}_j$ is probed, we add $i$ to $I$ and $j$ to $J$.

We note that after processing Group1 and Group 2 variables, we have $I = J$. We hereby denote the set of common indices in $I$ and $J$ as $U$. Since for every group of probed variables, we add at most one index to $I$ and $J$, by construction we must have $|I|, |J| \leq t_1$. Regarding the interconnection of variables, we note that for $i > j$, $\mathbf{r}_{i,j}$ is used in the computation of all partial sums $\mathbf{x}_{i,k}$ for $k \geq j$. Moreover, $\mathbf{r}_{i,j}$ is used in computing $\mathbf{A}_i \cdot \mathbf{b}_j + \mathbf{r}_{i,j}$, which is used in $\mathbf{r}_{j,i}$, which in turn used in all partial sums $\mathbf{x}_{j,k}$ for $k \geq i$. Therefore, if $i \notin U$, by definition $\mathbf{r}_{i,j}$ has not been probed (otherwise we will have $i \in U$), then $\mathbf{r}_{i,j}$ does not enter into the computation of any probed $\mathbf{x}_{i,k}$. Likewise, if $j \notin U$, then $\mathbf{r}_{j,i}$ is not probed and does not enter into the computation of any probed $\mathbf{x}_{j,k}$.

Now we proceed to simulate all probed intermediate variables except variables at Line 7 using the knowledge of $\mathbf{A}_{|I}$ and $\mathbf{b}_{|J}$. We distinguish the following four cases for $i < j$:

- Case 1: $i \in U \wedge j \in U$. All variables can be perfectly simulated straightforwardly. Particularly we simulate $\mathbf{r}_{i,j} \leftarrow \mathbb{F}_q^m$ like in real execution.
- Case 2: $i \in U \wedge j \notin U$. In this case the probed varibles are either in the form of $\mathbf{r}_{i,j}$ or $\mathbf{A}_i \cdot \mathbf{b}_j + \mathbf{r}_{i,j}$ (Group 3 values). For $\mathbf{r}_{i,j}$ we can perfectly simulate it by a random assignment in $\mathbb{F}_q^m$. For $\mathbf{A}_i \cdot \mathbf{b}_j + \mathbf{r}_{i,j}$, by definition of Group 3 values we must have $j \in J$, therefore we can perfectly simulate it from $\mathbf{A}_{|I}$ and $\mathbf{b}_{|J}$.
- Case 3: $i \notin U \wedge j \in U$. As we explained above, if $i \notin U$, neither $\mathbf{r}_{i,j}$ nor any partial sum $\mathbf{x}_{i,k}$ has been probed. In this case, the probed variables are $\mathbf{r}_{j,i}$ or $\mathbf{A}_i \cdot \mathbf{b}_j + \mathbf{r}_{i,j}$. We can simulate $\mathbf{r}_{j,i} \leftarrow \mathbb{F}_q^m$. As for $\mathbf{A}_i \cdot \mathbf{b}_j + \mathbf{r}_{i,j}$, we can perfectly simulate it by computing $\mathbf{A}_i \cdot \mathbf{b}_j + \mathbf{r}_{i,j} = \mathbf{A}_j \cdot \mathbf{b}_i + \mathbf{r}_{j,i}$ since by definition of Group 3 values we have $j \in U$ and $i \in J$.
- Case 4: $i \notin U \wedge j \notin U$. In this case the probed variables can only be in the form of $\mathbf{A}_i \cdot \mathbf{b}_j + \mathbf{r}_{i,j}$. Since both $i, j$ are not in $U$, which implies that $\mathbf{r}_{i,j}$ has not been probed and does not enter into the computation of other probed variables, we can perfectly simulate such probe by a random assignment.

From case 1, 2 and 3, we can perfectly simulate any variable $\mathbf{r}_{i,j}$ for $i \neq j$ and $i \in U$. This implies that we can therefore simulate all partial sums $\mathbf{x}_{i,k}$ for $i \in U$. All probed intermediate variables are perfectly simulated.

We now consider the simulation of the output variables $\mathbf{x}_i$, which can be distinguished into two cases:

- Case 1: $i \in U$. In this case, $\mathbf{x}_i$ has already been simulated before.
- Case 2: $i \notin U$. We first construct a subset of indices $V$ as follows: for any probed variable $\mathbf{A}_i \cdot \mathbf{b}_j + \mathbf{r}_{i,j}$ in Group 3, if $i \notin U \wedge i \in O \wedge j \notin U$, we add $j$ to $V$, otherwise we put $i$ in $V$. Since only Group 3 variables are considered, we have $|U| + |V| \leq t_1$.

This implies we have $|U| + |V| + |O| \leq d$. Therefore, there exists an index $j^\star \in [1, d]$, such that $j^\star \notin U \cup V \cup O$. For any $i$, we can rewrite $\mathbf{x}_i$ as:

$$\mathbf{x}_i = \mathbf{A}_i \cdot \mathbf{b}_i + \sum_{j \neq i} \mathbf{r}_{i,j} = \mathbf{r}_{i,j^\star} + \left( \mathbf{A}_i \cdot \mathbf{b}_i + \sum_{j \neq i, j^\star} \mathbf{r}_{i,j} \right)$$

We claim that neither $\mathbf{r}_{i,j^\star}$ or $\mathbf{r}_{j^\star,i}$ were used elsewhere. Since $i \notin U$, neither $\mathbf{r}_{i,j^\star}$ or the partial sum $\mathbf{x}_{i,k}$ was probed. Likewise, neither $\mathbf{r}_{j^\star,i}$ nor the partial sum $\mathbf{x}_{j^\star,k}$ was probed. Further, by the construction of $V$, we have neither $\mathbf{A}_i \cdot \mathbf{b}_{j^\star} + \mathbf{r}_{i,j^\star}$ nor $\mathbf{A}_{j^\star} \cdot \mathbf{b}_i + \mathbf{r}_{j^\star,i}$ was probed. Therefore, we can perfectly simulate every $\mathbf{x}_i$ for $i \notin U$ by a random fresh value.

In both cases we can perfectly simulate every $\mathbf{x}_i$. We have simulated all probed variables, which completes the proof. □

# B Auxiliary gadgets

## B.1 Share-wise matrix-vector multiplication.

`halfSecMatVecMult` is a basic gadget that computes the product of a matrix and a vector, where the matrix $\mathbf{A}$ is public, and the vector $\mathbf{b}$ is given in $d$ additive shares. In this case, multiplication in a share-wise manner is sufficient to protect the sensitive vector.

**Algorithm 6** Share-wise matrix-vector multiplication (`halfSecMatVecMult`)

**Input:** $\mathbf{A} \in \mathbb{F}_q^{m \times m}$, and shares $\mathbf{b}_1, \ldots, \mathbf{b}_d \in \mathbb{F}_q^m$
**Output:** shares $\mathbf{x}_1, \ldots, \mathbf{x}_d$ satisfying $\sum_{i=1}^d \mathbf{x}_i = \mathbf{A} \cdot \sum_{i=1}^d \mathbf{b}_i$
1: **for** $i = 1$ to $d$ **do**
2: $\quad \mathbf{x}_i \leftarrow \mathbf{A} \cdot \mathbf{b}_i$
3: **return** $\mathbf{x}_1, \ldots, \mathbf{x}_d$

**Complexity.** The run-time complexity of `halfSecMatVecMult` is:

$$T_{\texttt{halfSecMatVecMult}}(m, d) = d \cdot (2m - 1)m = \mathcal{O}(m^2 d).$$

**Lemma 8.** `halfSecMatVecMult` *(Algorithm 6) is* $t$-NI *secure.*

*Proof.* Since the matrix-vector multiplication is performed share-wise, `halfSecMatVecMult` can be modeled as a $t$-NI gadget. □

## B.2 Refreshing shares

We recall two existing gadgets for refreshing shares, with each satisfies different security property. First we recall `LinearRefreshMasks` from [RP10], which works in any finite abelian group $(G, +)$. To match the notations and requirements of our paper, we instantiate it as $(\mathbb{F}_q^{m \times m}, +)$.

**Algorithm 7** `LinearRefreshMasks`

**Input:** $\mathbf{A}_1, \ldots, \mathbf{A}_d \in \mathbb{F}_q^{m \times m}$

**Output:** $\mathbf{B}_1, \ldots, \mathbf{B}_d \in \mathbb{F}_q^{m \times m}$, with $\sum_i^d \mathbf{B}_i = \sum_i^d \mathbf{A}_i$

1: $\mathbf{B}_d \leftarrow \mathbf{A}_d$
2: **for** $i = 1$ to $d - 1$ **do**
3: $\quad \mathbf{R}_i \leftarrow \mathbb{F}_q^{m \times m}$
4: $\quad \mathbf{B}_i \leftarrow \mathbf{A}_i + \mathbf{R}_i$
5: $\quad \mathbf{B}_d \leftarrow \mathbf{B}_d - \mathbf{R}_i$
6: **return** $\mathbf{B}_1, \ldots, \mathbf{B}_d$

**Complexity.** The run-time and randomness complexity of `LinearRefreshMasks` are:

$$T_{\texttt{LinearRefreshMasks}}(m, d) = (m^2 \cdot (3d - 2)) = \mathcal{O}(m^2 d),$$
$$R_{\texttt{LinearRefreshMasks}}(m, d) = (m^2 \cdot (d - 1)) = \mathcal{O}(m^2 d).$$

**Lemma 9.** `LinearRefreshMasks` *is* $t$-SNI *secure [BBD+16].*

For secure shares recombination, we recall the full mask refreshing gadget `RefreshMasks` which is proven to be *free-*$t$-SNI secure in [CS21]. Again, we specialize the group to $(\mathbb{F}_q^{m \times m}, +)$.

**Algorithm 8** `RefreshMasks`

**Input:** $\mathbf{A}_1, \ldots, \mathbf{A}_d \in \mathbb{F}_q^{m \times m}$

**Output:** $\mathbf{B}_1, \ldots, \mathbf{B}_d \in \mathbb{F}_q^{m \times m}$, with $\sum_i^d \mathbf{B}_i = \sum_i^d \mathbf{A}_i$

1: **for** $i = 1$ to $d$ **do** $\mathbf{B}_i \leftarrow \mathbf{A}_i$
2: **for** $i = 1$ to $d$ **do**
3: $\quad$ **for** $j = i + 1$ to $d$ **do**
4: $\quad\quad \mathbf{R} \leftarrow \mathbb{F}_q^{m \times m}$
5: $\quad\quad \mathbf{B}_i \leftarrow \mathbf{B}_i + \mathbf{R}$
6: $\quad\quad \mathbf{B}_j \leftarrow \mathbf{B}_j - \mathbf{R}$
7: **return** $\mathbf{B}_1, \ldots, \mathbf{B}_d$

**Complexity.** The run-time and randomness complexity of `RefreshMasks` are:

$$T_{\texttt{RefreshMasks}}(m, d) = (m^2 \cdot 3d(d-1)/2) = \mathcal{O}(m^2 d^2),$$
$$R_{\texttt{RefreshMasks}}(m, d) = (m^2 \cdot d(d-1)/2) = \mathcal{O}(m^2 d^2).$$

**Lemma 10.** `RefreshMasks` *is free-*$t$-SNI *secure [CS21].*