Protecting the Unbalanced Oil and Vinegar Signature Scheme against Side-channel Attack

DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
in
Cryptology and Security

by

Uttam Kumar Ojha
[Roll No: CrS2223]

Under the guidance of

Prof. Dr. Ingrid Verbauwhede
Prof. Dr. Bimal Kumar Roy

Daily supervisors

Suparna Kundu
Quinten Norga
Dr. Angshuman Karmakar

June 2024

CERTIFICATE

This is to certify that the dissertation entitled **"Protecting the Unbalanced Oil and Vinegar Signature Scheme against Side-channel Attack"** submitted by **Uttam Kumar Ojha** to Indian Statistical Institute, Kolkata, in partial fulfilment for the award of the degree of **Master of Technology in Cryptology and Security** is a bonafide record of work carried out by him under my supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission.

Dr. Ingrid Verbauwhede
Professor,
COSIC,
KU Leuven,
Leuven, 3001, BELGIUM.

Dr. Bimal Kumar Roy
Professor,
Applied Statistics Unit,
Indian Statistical Institute,
Kolkata, 700108, INDIA.

Acknowledgments

I feel so blessed because Prof. Dr. Bimal Kumar Roy introduced me to the wonderland of cryptography years back and now guides me towards my journey from crypto enthusiast to researcher; I owe to him. I want to show my highest gratitude to Prof. Dr. Bimal Kumar Roy, Prof. Dr. Ir. Bart Preneeland and Prof. Dr. Ingrid Verbauwhede for their supervision and continuous support.

I'm quite fortunate to have excellent daily supervisors, Suparna Kundu, Quinten Norga, and Dr. Angshuman Karmakar. I would also like to thank you all for spending your precious time, valuable suggestions, and discussions on this thesis. This thesis could not be in reality without you.

I am thankful to administrative people, especially Péla Noé, for making my life as easy as possible from all administrative aspects.

My deepest thanks to all my teachers for the knowledge I have, which added an essential dimension to me. Finally, I thank my parents, family and friends for their everlasting support.

Uttam Kumar Ojha

Abstract

With the recent development of quantum computing, there is an urge for Post-Quantum Cryptography(PQC). The National Institute of Standards and Technology(NIST) initiated a public process to standardize PQC algorithms to address this issue in 2016. To search for new signature schemes with diverse hardness problems, short signature sizes and fast verification, NIST called for additional digital signature schemes for the PQC in 2022.

Based on multivariate cryptography, the Unbalanced Oil and Vinegar(UOV) signature scheme is a candidate for this additional round. This scheme has stood out for two decades of cryptanalysis and has a short signature size and fast verification. We believe this is a potential candidate for this round. As usual, this scheme is mainly designed to resist mathematical attacks; however, deploying this scheme in an actual device leaks unintended information through side-channels such as power consumption. Side-channel analysis helps to exploit those unintended information and recover the secrets of the scheme. Recently, a few attacks have been shown using correlation power analysis in this scheme.

Masking is a well-known and provably secure countermeasure against such attacks. In this thesis, we describe the first masked implementation of the UOV scheme. We also produce security proof of our implementation in the probing model.

Keywords: *Side-channel attack, Masking, Post-quantum cryptography, UOV signature*

Contents

List of Tables

Chapter 1

Introduction

Research shows us that the present hardness assumptions for public key cryptography, like factorization and elliptic curve discrete logarithm, are no longer sound due to the recent development of quantum computing. We need new public key cryptography schemes whose hardness assumptions are quantum-safe but efficient to implement. NIST initiated a public process to standardize post-quantum cryptography algorithms in 2016 and selected lattice-based CRYSTALS-Dilithium and Falcon and hash-based SPHINCS+ for the digital signature in 2022. In the same year, NIST called for additional digital signature schemes [14] with diverse hardness assumptions, short signature sizes, fast verification, or schemes that outperform significantly if lattice-based.

Beullens et al. proposed the unbalanced oil and vinegar signature scheme [5] for the additional standardized process. UOV is an excellent choice for signature size and time efficiency. This scheme is based on multivariate cryptography. More than twenty years of cryptanalysis provide confidence in the mathematical security of this scheme. UOV is a potential candidate for this selection process. Side-channel resistant implementation is crucial while deploying a cryptographic scheme in the real world. An attacker can acquire knowledge about sensitive data of a scheme using side-channel analysis. Unfortunately, recent research [1, 16, 15] shows NIST submission UOV is not out of this risk using power analysis.

Masking is a well-studied countermeasure against side-channel attacks using power or electromagnetic trace. In the masking technique, we split the sensitive data into multiple shares and securely perform the operations with each share. Thus, we convert the original cryptographic scheme to its equivalent form so that it provides provable security against the probing model. However, it comes with performance and resource overhead, so we need to address those issues by carefully choosing optimised novel techniques.

1.1 Motivation

The research community should study the masking of UOV. Recent masking publications on other schemes show the conversion, produce security analysis on the probing model, and physical security analysis using Test Vector Leakage Assessment (TVLA) [10] methodology. We believe no publication has been made on the UOV digital signature scheme yet. So, we try to address this in this thesis.

1.2 Our Contribution

Our contributions are summarized as follows.

- We propose a first-order masked version of the NIST submission UOV digital signature scheme,
- We prove the security of our implementation in the 1-probe model,
- We analyse the complexity of our implementation in terms of required random bits.

1.3 Thesis Outline

We discuss the required preliminaries, i.e. the NIST submission UOV scheme, masking and probing model in Chapter 2. In detail, we describe our work, i.e., masked UOV scheme implementation, provide security analysis in 1-porbe model, and complexity regarding required randomness in Chapter 3. We summarize our work in Chapter 4.

Chapter 2

Preliminaries

In sections 2.1-4, we recall an overview of the original UOV [12], the design rationale behind the NIST submission UOV and its functionalities and specification of variants from [5]. In section 2.5, we recall the security definitions in the probing model from [4, 3] for further security analysis of our implementation.

2.1 Notations and Conventions

All logarithms are in base 2. All indices start with 0. $\mathbb{F}_q$ (i.e. $GF(q)$) denotes a finite field with q ($= 2^k$) elements. Each element in $\mathbb{F}_q$ is represented as a polynomial over $\mathbb{F}_2$. Since $x = -x$ in F_q, we abuse notation by considering field addition and subtraction as the same. Let $x, y \in \mathbb{F}_q$, $x + y$ denotes addition/subtraction (i.e. bitwise XOR) and xy denotes multiplication of x, y. $\mathbb{F}_q^*$ denotes $\mathbb{F}_q \setminus \{0\}$. All the polynomials, vectors, and matrices are defined over $\mathbb{F}_q$. All the vectors are in column form and denoted as bold lower-case letters; all the matrices are defined as bold upper-case letters. $\mathbf{x} = (x_i)$ denotes a column vector whose i-th coordinate is x_i. $\mathbf{A}^T$ denotes transposition of matrix $\mathbf{A}$. $\mathbf{0}_d$ denotes the d-dimensional zero vector, and $\mathbf{I}_d$ denotes d-dimensional identity matrix. $||$ denotes string concatenation, $:=$ denotes the assignment of a variable. $\neg, \vee, \wedge, \ll, and \gg$ denote logical not, or, and, left shift, and right shift, respectively.

2.2 An Overview of UOV

In a multivariate public-key cryptosystem, the public key $\mathcal{P} = (p_0, \ldots, p_{m-1})$ is a system of a nonlinear equation in n variables over $\mathbb{F}_q$, where n, m, q are public parameters. For cryptographic purposes, $\mathcal{P}$ is chosen so that it should work as a trapdoor one-way function: for any given $\mathbf{x} \in \mathbb{F}_q^n$, the evaluation $\mathcal{P}(\mathbf{x}) = (p_0(\mathbf{x}), \ldots, p_{m-1}(\mathbf{x}))$ is easy; for any given $\mathbf{t} \in \mathbb{F}_q^m$, finding a preimage $\mathbf{s} \in \mathbb{F}_q^n$ such that $\mathcal{P}(\mathbf{s}) = \mathbf{t}$ is easy with the trapdoor information but hard without knowledge of trapdoor. The homogeneous

quadratic polynomial of the form

$$f(x_0, \ldots, x_{n-1}) = \sum_{i=0}^{n-m-1} \sum_{j=i}^{n-1} \alpha_{ij} x_i x_j$$

is called **oil-vinegar polynomial** or simply **OV-polynomial**; the $n - m$ variables $x_0, \ldots, x_{n-m-1}$ are referred as **vinegar variables**, the remaining m variables $x_{m-n}, \ldots, x_{n-1}$ as **oil variables**.

Original UOV. The original UOV digital signature scheme follows the hash-and-sign paradigm with $n > 2m$:

- In the key generation algorithm, given the security parameter $params$, it returns a public-secret key pair $(pk = \mathcal{P}, sk = (\mathcal{F}, \mathcal{T}))$ randomly.
- In the signing algorithm, given the secret key $sk = (\mathcal{F}, \mathcal{T})$ and a message $\mu \in \{0,1\}^*$, it returns a signature σ of message μ. $\mathcal{F}$ has special structure to compute its preimage efficiently and $\mathcal{T}$ hides this structure in public key $pk = \mathcal{P}$.
- In the verification algorithm, given the public key $pk = \mathcal{P}$ and a message-sign pair (μ, σ), it returns whether the given pair is valid.

Algorithm 1 KeyGen

Input: $params = (n, m, q)$
Output: (pk, sk)
1: Pick m uniformly random OV-polynomials $f_0, \ldots, f_{m-1}$
2: $\mathcal{F} := (f_0, \ldots, f_{m-1})$
3: Pick a uniformly random invertible linear transformation $\mathcal{T} : \mathbb{F}_q^n \to \mathbb{F}_q^n$
4: $\mathcal{P} := \mathcal{F} \circ \mathcal{T}$
5: $pk := \mathcal{P}$
6: $sk := (\mathcal{F}, \mathcal{T})$
7: **return** (pk, sk)

2.3 Design Rationale Behind UOV

Reformulation of trapdoor. In OV-polynomial, the oil variables never mix with the oil variable, i.e. there is no term with $x_i x_j$ in OV-polynomials such that both x_i and x_j are oil variables. Due to this special structure, for a random choose $\mathcal{F} : \mathbb{F}_q^n \to \mathbb{F}_q^m$, every fixed **vinegar vector v** $\in \mathbb{F}_q^{n-m}$ induces a full-rank linear transformation

Algorithm 2 Sign

Input: $params, sk = (\mathcal{F}, \mathcal{T}), \mu \in \{0,1\}^*$
Output: Signature σ
1: $\mathbf{t} := \text{Hash}(\mu)$ ▷ Hash : $\{0,1\}^* \to \mathbb{F}_q^m$
2: Pick 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** σ

Algorithm 3 Verify

Input: $params, pk = \mathcal{P}, (\mu, \sigma)$
Output: $true$ or $false$
1: $\mathbf{t} := \text{Hash}(\mu)$
2: $\mathbf{t}' := \mathcal{P}(\sigma)$
3: **return** $\mathbf{t} == \mathbf{t}'$

$\mathcal{F}\left(\begin{bmatrix}\mathbf{v}\\ .\end{bmatrix}\right) : \mathbb{F}_q^m \to \mathbb{F}_q^m$ with significant high probability; thus, it's easy to compute a preimage $\mathbf{u} = \begin{bmatrix}\mathbf{v}\\ \mathbf{w}\end{bmatrix} \in \mathbb{F}_q^n$ such that $\mathcal{F}(\mathbf{u}) = \mathbf{t}$ and a preimage $\mathbf{s} \in \mathbb{F}_q^n$ such that $\mathcal{T}(\mathbf{s}) = \mathbf{u}$. Therefore,

$$\mathbf{s} = \mathcal{T}^{-1}\left(\begin{bmatrix}\mathbf{v}\\ \mathbf{0}_m\end{bmatrix} + \begin{bmatrix}\mathbf{0}_{n-m}\\ \mathbf{w}\end{bmatrix}\right) = \mathcal{T}^{-1}\left(\begin{bmatrix}\mathbf{v}\\ \mathbf{0}_m\end{bmatrix}\right) + \mathcal{T}^{-1}\left(\begin{bmatrix}\mathbf{0}_{n-m}\\ \mathbf{w}\end{bmatrix}\right).$$

$\mathcal{T}^{-1}\left(\begin{bmatrix}\mathbf{0}_{n-m}\\ \mathbf{w}\end{bmatrix}\right)$ is in **oil space** $O = \text{span}(\mathcal{T}^{-1}(\mathbf{e}_{n-m}), \ldots, \mathcal{T}^{-1}(\mathbf{e}_{n-1}))$, where $\mathbf{e}_i$ denotes the vector with 1 in i-th coordinate and zeros elsewhere. O is m-dimensional subspace of $\mathbb{F}_q^n$ which can be seen as column space of the matrix $\overline{\mathbf{O}} = \begin{bmatrix}\mathbf{O}\\ \mathbf{I}_m\end{bmatrix}$, where $\mathbf{O}$ is picked uniformly random from $\mathbb{F}_q^{(n-m)\times m}$. Therefore, a short description of trapdoor information is secret $seed_{sk}$ that is used to sample $\mathbf{O}$.

Generation of the public key $\mathcal{P}$. Note that there is one vinegar variable in each term in OV-polynomial, and the oil space O is a linear span of $(\mathcal{T}^{-1}(\mathbf{e}_{n-m}), \ldots, \mathcal{T}^{-1}(\mathbf{e}_{n-1}))$. Therefore, $\mathcal{P} = \mathcal{F} \circ \mathcal{T}$ vanishes on every element in oil space. Each homogeneous multivariate polynomial p_i can be represented by an upper triangular matrix $\mathbf{P}_i \in \mathbb{F}_q^{n\times n}$ such that $p_i(\mathbf{x}) = \mathbf{x}^T\mathbf{P}_i\mathbf{x}$. Let

$$\mathbf{P}_i = \begin{bmatrix}\mathbf{P}_i^{(0)} & \mathbf{P}_i^{(1)}\\ \mathbf{0} & \mathbf{P}_i^{(3)}\end{bmatrix}$$

where $\mathbf{P}_i^{(0)} \in \mathbb{F}_q^{(n-m)\times(n-m)}, \mathbf{P}_i^{(3)} \in \mathbb{F}_q^{m\times m}$ are upper-triangular, and $\mathbf{P}_i^{(1)} \in \mathbb{F}_q^{(n-m)\times m}$.

Lemma 1 *Let* $\mathbf{A} \in \mathbb{F}_q^{n\times n}$. *Then* $\mathbf{x}^T\mathbf{A}\mathbf{x} = 0$ *for all* $\mathbf{x} \in \mathbb{F}_q^n$ *if and only if* $\mathbf{A}$ *is skew-symmetric, i.e.* $\mathbf{A}^T = -\mathbf{A}$.

Therefore, p_i vanishes on the oil space O, i.e. column space of $\overline{\mathbf{O}}$ if and only if

$$\begin{bmatrix}\mathbf{O}^T & \mathbf{I}_m\end{bmatrix} \mathbf{P}_i \begin{bmatrix}\mathbf{O}\\ \mathbf{I}_m\end{bmatrix} = \mathbf{O}^T\mathbf{P}_i^{(0)}\mathbf{O} + \mathbf{O}^T\mathbf{P}_{1,k} + \mathbf{P}_i^{(3)}$$

is skew-symmetric. Thus, given the trapdoor $\overline{\mathbf{O}}$, we can generate $\{\mathbf{P}_i \mid 0 \le i \le m-1\}$ as follows: first $\{\mathbf{P}_i^{(0)}, \mathbf{P}_i^{(1)} \mid 0 \le i \le m-1\}$ is picked uniformly random; then compute

$$\mathbf{P}_i^{(3)} := \text{Upper}\left(-\mathbf{O}^T\mathbf{P}_i^{(0)}\mathbf{O} - \mathbf{O}^T\mathbf{P}_i^{(1)}\right)$$

for all $0 \le i \le m-1$, where Upper($\mathbf{M}$) denotes the unique upper triangular matrix $\mathbf{N}$ such that $\mathbf{M} - \mathbf{N}$ is skew-symmetric.

Inversion operation. Given the trapdoor $\overline{\mathbf{O}}$ and $\{\mathbf{P}_i \mid 0 \le i \le m-1\}$, improved process of calculating a preimage $\mathbf{s}$ for a given $\mathbf{t}$ as follows: first a vinegar vector $\mathbf{v} \in \mathbb{F}_q^{n-m}$ is picked uniformly random, and then try to find a $\mathbf{x} \in \mathbb{F}_q^m$, where $\mathbf{s}$ is of following form

$$\mathbf{s} = \begin{bmatrix}\mathbf{v}\\ \mathbf{0}_m\end{bmatrix} + \begin{bmatrix}\mathbf{O}\\ \mathbf{I}_m\end{bmatrix}\mathbf{x}$$

Lemma 2 *Let* $\mathbf{s}$ *be of the above form. Then* $\mathcal{P}(\mathbf{s}) = \mathbf{t}$ *if and only if* $\mathbf{L}\mathbf{x} = \mathbf{t} - \mathbf{y}$, *where*

- $\mathbf{S}_i = (\mathbf{P}_i^{(0)} + \mathbf{P}_i^{(0)T})\mathbf{O} + \mathbf{P}_i^{(1)} \in \mathbb{F}_q^{n-m\times m}$,
- $\mathbf{L} \in \mathbb{F}_q^{m\times m}$ *with the* i*-th row being* $\mathbf{v}^t\mathbf{S}_i$, *and*
- $\mathbf{y} = [\mathbf{v}^T\mathbf{P}_i^{(0)}\mathbf{v}]_{0\le i\le m-1} \in \mathbb{F}_q^m$.

When L is invertible (with a significantly high probability), we can compute $\mathbf{x}$ with the Gaussian elimination method; otherwise, repeat the forgoing process with a fresh vinegar vector. Note that $\mathbf{S}_i$ depends only on public $\mathbf{P}_i^{(0)}, \mathbf{P}_i^{(1)}$, and secret $\mathbf{O}$.

2.4 The UOV Digital Signature Scheme

In this section, we discuss the NIST additional digital signature schemes round one submission UOV digital signature scheme.

2.4.1 Parameters in UOV

The UOV digital signature scheme is parameterized as follows:

$$params = (n, m, q, salt\_len, sk\_seed\_len, pk\_seed\_len),$$

where

- n denotes the number of variables in the polynomials in $\mathcal{P}$,
- m denotes the number of polynomials in $\mathcal{P}$,
- q denotes the size of a finite field,
- $salt\_len$ denotes the bit length of $salt \in \{0,1\}^{salt\_len}$,
- sk_seed_len denotes the bit length of $seed_{sk} \in \{0,1\}^{sk\_seed\_len}$,
- pk_seed_len denotes the bit length of $seed_{pk} \in \{0,1\}^{pk\_seed\_len}$.

	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

Table 2.1: Recommended parameter sets for UOV variants

For all recommended parameter sets, $salt\_len = 128$, $sk\_seed\_len = 256$, and $pk\_seed\_len = 128$.

2.4.2 Functionalities in UOV

We describe the five algorithms in the UOV digital signature scheme.

- *CompactKeyGen.* Given the security parameter $params$, it returns a compact representation of public-secret key pair (cpk, csk) randomly.
- *ExpandSK.* Given the compact representation of secret key csk, it returns the expanded representation of secret key esk.
- *Sign.* Given the expanded representation of secret key esk, and a message μ, it returns a signature σ of message μ.
- *ExpandPK.* Given the compact representation of public key cpk, it returns the expanded representation of public key epk.

- *Verify.* Given the expanded representation of public key epk, and a message-sign pair (μ, σ), it returns whether the given pair is valid.

Expand_{sk}, Hash, and $\text{Expand}_{\mathbf{V}}$ are instantiated with shake256. $\text{Expand}_{\mathbf{P}}$ is instantiated with 4 rounds aes128ctr using $seed_{pk}$ as key and zero as nonce.

Algorithm 4 CompactKeyGen

Input: Security parameter $params$
Output: Compact representation of public-secret key pair (cpk, csk)

1: Pick a uniformly random $seed_{sk} \in \{0,1\}^{sk\_seed\_len}$
2: Pick a uniformly random $seed_{pk} \in \{0,1\}^{pk\_seed\_len}$
3: $\mathbf{O} := \text{Expand}_{sk}(seed_{sk})$
4: $\{\mathbf{P}_i^{(0)}, \mathbf{P}_i^{(1)}\}_{0 \le i \le m-1} := \text{Expand}_{\mathbf{P}}(seed_{pk})$
5: **for** $i = 0$ upto $m-1$ **do**
6: $\quad \mathbf{P}_i^{(3)} := \text{Upper}\left(-\mathbf{O}^T\mathbf{P}_i^{(0)}\mathbf{O} - \mathbf{O}^T\mathbf{P}_i^{(1)}\right)$
7: **end for**
8: $cpk := \left(seed_{pk}, \left\{\mathbf{P}_i^{(3)}\right\}_{0 \le i \le m-1}\right)$
9: $csk := (seed_{pk}, seed_{sk})$
10: **return** (cpk, csk)

Algorithm 5 ExpandSK

Input: Compact representation of secret key csk
Output: Expanded representation of secret key esk

1: $\mathbf{O} := \text{Expand}_{sk}(seed_{sk})$
2: $\left\{\mathbf{P}_i^{(0)}, \mathbf{P}_i^{(1)}\right\}_{0 \le i \le m-1} := \text{Expand}_{\mathbf{P}}(seed_{pk})$
3: **for** $i = 0$ upto $m-1$ **do**
4: $\quad \mathbf{S}_i := \left(\mathbf{P}_i^{(0)} + \mathbf{P}_i^{(0)T}\right)\mathbf{O} + \mathbf{P}_i^{(1)}$
5: **end for**
6: $esk := \left(seed_{sk}, \mathbf{O}, \left\{\mathbf{P}_i^{(0)}, \mathbf{S}_i\right\}_{0 \le i \le m-1}\right)$
7: **return** esk

2.4.3 Specification of the UOV variants

We instantiate the usual three-part API in a digital signature (Key Generation, Sign, verify) in three different variants: classic, pkc, and pkc+skc, combining the above five algorithms. Note that the three variants have the same signature scheme but different public and secret key representations. One variant can verify a signature generated

Algorithm 6 Sign

Input: Expanded representation of secret key $esk = \left(seed_{sk}, \mathbf{O}, \{\mathbf{P}_i^{(0)}, \mathbf{S}_i\}_{0 \leq i \leq m-1}\right)$, message μ
Output: Signature σ

Pick a uniformly random $salt \in \{0,1\}^{salt\_len}$
$\mathbf{t} := \text{Hash}(\mu || salt)$
for $ctr = 0$ upto 255 **do**
    $\mathbf{v} := \text{Expand}_v(\mu || salt || seed_{sk} || ctr)$
    $\mathbf{L} := \mathbf{0}_{m \times m}$
    for $i = 0$ upto $m-1$ **do**
        Set i-th row of $\mathbf{L}$ to $\mathbf{v}^T \mathbf{S}_i$
    end for
    if $\mathbf{L}$ is invertible **then**
        $\mathbf{y} \leftarrow [\mathbf{v}^T \mathbf{P}_i^{(0)} \mathbf{v}]_{0 \leq i \leq m-1}$
        $\mathbf{x} := \mathbf{L}^{-1}(\mathbf{t} - \mathbf{y})$
        $\mathbf{s} := \begin{bmatrix} \mathbf{v} \\ \mathbf{0}_m \end{bmatrix} + \begin{bmatrix} \mathbf{O} \\ \mathbf{I}_m \end{bmatrix} \mathbf{x}$
        $\sigma := (\mathbf{s}, salt)$
        return σ
    end if
end for
return $\perp$

Algorithm 7 ExpandPK

Input: Compact representation of public key cpk
Output: Expanded representation of public key epk

$\left\{\mathbf{P}_i^{(0)}, \mathbf{P}_i^{(1)}\right\}_{0 \leq i \leq m-1} := \text{Expand}_{\mathbf{P}}(seed_{pk})$
for $i = 0$ upto $m-1$ **do**
    $\mathbf{P}_i = \begin{bmatrix} \mathbf{P}_i^{(0)} & \mathbf{P}_i^{(1)} \\ \mathbf{0} & \mathbf{P}_i^{(3)} \end{bmatrix}$
end for
$epk := \{\mathbf{P}_i\}_{0 \leq i \leq m-1}$
return epk

Algorithm 8 Verify

Input: Expanded representation of public key $epk = \{\mathbf{P}_i\}_{0 \leq i \leq m-1}$, message-sign pair $(\mu, \sigma = (\mathbf{s}, salt))$
Output:

$\mathbf{t} := Hash(\mu || salt)$
return $\left(\mathbf{t} == [\mathbf{s}^T \mathbf{P}_i \mathbf{s}]_{0 \leq i \leq m-1}\right)$

by other variants, and each variant achieves the same hardness for the same security parameter.

- classic. In this variant, the public-secret key pair is (epk, esk), i.e. ExpandSK and ExpandPK are part of the Key Generation algorithm. This makes the key size large, but sign and verify faster.
- pkc. In this public key compressed variant, the public-secret key pair is (cpk, esk), i.e. ExpandSK is part of the Key Generation algorithm, but ExpandPk is part of the Verify algorithm. This makes the public key size small but verify slower.
- pkc+skc. In this public-secret key compressed variant, the public-secret key pair is (cpk, csk), i.e. ExpandSK is part of the Sign algorithm, but ExpandPk is part of the Verify algorithm. This makes the public-secret key size small but slower to sign and verify.

	key pair	public key compressed	secret key compressed
classic	(epk, esk)		
pkc	(cpk, esk)	✓	
pkc+skc	(cpk, csk)	✓	✓

Table 2.2: Qualitative comparisons of three UOV variants

2.5 Masking and Probing model

Let $(G, \circ)$ be a group and d be a positive integer. For masking countermeasure, we split every sensitive variable $x \in G$ into $d+1$ shares $x_0, \ldots, x_d \in G$ randomly such that $x = x_0 \circ x_1^{-1} \circ \ldots \circ x_d^{-1}$, where x_i^{-1} denotes the inverse of x_i with respect to $\circ$. The $(d+1)$-tuple $(x_0, \ldots, x_d)$ is called a **d-th order masking** of x. Each $x_i \in \{x_0, \ldots, x_d\}$ is called a **share** of variable x. Throughout this thesis, when group operation is an addition/bitwise XOR(resp. multiplication), we call it **Additive**/**Boolean**(resp. **Multiplicative**) masking. Let u, v be two positive integers and $\mathbf{x}_i$(resp. $\mathbf{y}_i$) be $(u-1)$-th(resp. $(v-1)$-th) order masking of x_i(resp. y_i).

Definition 1 (gadget) *A (u, v)-gadget for a function $f : \mathbb{F}_q^n \to \mathbb{F}_q^m$ is a (randomize) algorithm $\mathcal{A}$ such that for all $(\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}) \in (\mathbb{F}_q^u)^n$ (and randomness r), $(\mathbf{y}_0, \ldots, \mathbf{y}_{m-1}) := \mathcal{A}(\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}; r)$ satisfies $(y_0, \ldots, y_{m-1}) = f(x_0, \ldots, x_{n-1})$.*

The above definition can naturally be extended for any domain of a function f. A **probe** is a share of a gadget's input/internal/output variable. Let t, l be two positive integers and $\mathbf{x}_i = \{x_{ij} \mid 0 \leq j \leq u-1\}$ be $(u-1)$-th order masking of x_i.

Definition 2 (t-simulatability of l probes) *Let $\mathcal{A}$ be a (u, v)-gadget for a function define over $\mathbb{F}_q^n$. A set $P = \{p_0, \ldots, p_{l-1}\}$ of l probes on $\mathcal{A}$ is t-simulatable, if there exits n sets $I_0, \ldots, I_{n-1} \subseteq \{0, \ldots, u-1\}$ with $|I_i| \leq t$ for all i and a (randomized) function $sim : (\mathbb{F}_q^t)^n \to \mathbb{F}_q^l$ such that for all $(\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}) \in (\mathbb{F}_q^u)^n$, the distributions $\{p_0, \ldots, p_{l-1}\}$ (which depends on $(\mathbf{x}_0, \ldots, \mathbf{x}_{n-1})$ and the randomness used by the gadget) and $\{sim((\mathbf{x}_0)_{I_0}, \ldots, (\mathbf{x}_{n-1})_{I_{n-1}})\}$ are identical, where $(\mathbf{x_i})_{I_i}$ denotes $\{x_{ij} \mid j \in I_i\}$.*

Definition 3 (d-non-interference (d-NI)) *A (u, v)-gadget $\mathcal{A}$ is d-non-interfering if and only if every set of at most d probes on $\mathcal{A}$ is d-simulatabale.*

Definition 4 (d-strong non-interference (d-SNI)) *A (u, v)-gadget $\mathcal{A}$ is d-strong non-interfering if and only if for every set P_0 of d_0 probes on input or internal shares and every set P_1 of d_1 probes on output shares such that $d_0 + d_1 \leq d$, the set $P_0 \cup P_1$ of probes on $\mathcal{A}$ is d_0-simulatable.*

Since we propose first-order masking of the UOV signature scheme, we build 1-NI, 1-NI with public output X or 1-SNI gadgets. A gadget is 1-NI if and only if any probe on this gadget can be simulated with at most one share of each input of the gadget. A gadget is 1-NI with public output X if any probe on this gadget can be simulated with public output X and at most one share of each input of the gadget. A gadget is 1-SNI if any internal probe can be simulated with at most one share of each input of the gadget, but any probe on output should be independent of inputs.

Chapter 3

Masking UOV signature scheme

In this chapter, we build gadgets for the UOV signature scheme in the 1-probe model, i.e. first-order masking. We mask CompactKeyGen, ExpandSK, and Sign algorithms since those algorithms deal with secrets. ExpandPK and Verify are public algorithms and don't deal with any secrets, so they don't require any masking.

3.1 Overall structure

In CompactKeyGen, ExpandSK, and Sign algorithm, the following variables are secret/sensitive: $seed_{sk}$, $\mathbf{O}$, $\{\mathbf{S}_i\}_{0\leq i\leq m-1}$, $\mathbf{v}$, $\mathbf{L}$, and $\mathbf{y}$, and following variables are public: $seed_{pk}$, $\{\mathbf{p}_i^{(0)}, \mathbf{p}_i^{(1)}, \mathbf{p}_i^{(3)}\}_{0\leq i\leq m-1}$, μ (i.e. message), and $\sigma = (\mathbf{s}, salt)$ (i.e. signature). We build gadgets MaskedCompactKeyGen, MaskedExpandSK, and MaskedSign for CompactKeyGen, ExpandSK, and Sign, respectively. We consider the internal variable $\mathbf{x}$ in line 11 of the original sign algorithm as the public output of the gadget MaskedSign. The last m coordinate of $\mathbf{s}$ is the same as $\mathbf{x}$, so we believe the above consideration doesn't harm security. Due to the nature of the algorithms, we mostly use additive/boolean masking; multiplicative masking is also used in a few gadgets.

3.2 Masked gadgets

We use or take motivation from gadgets already in the literature; we provide a short description, security type and reference of those in Table 3.1. We reproduce the algorithm of those gadgets in Appendix A.

3.2.1 CondAdd

We perform a conditional add of two rows to make a pivot element of a matrix non-zero while transforming a matrix to its row echelon form. CondAdd gadget takes additive masking (x_0, x_1) of $x \in \mathbb{F}_q$, (y_0, y_1) of $y \in \mathbb{F}_q$ and one-bit boolean masking

Algorithm	Description	Security	Reference
SecAnd	Bitwise AND of boolean shares	d-SNI	[8, 2]
SecOr	Bitwise OR of boolean shares	d-SNI	[6]
Refresh	Refresh of additive shares	d-SNI	[2]
SecNonzero	Nonzero check of additive shares	d-SNI	[6]
FullXor	Unmask additive shares	d-NI	[8, 3]
AMtoMM	Additive to multiplicative conversion	d-SNI	[9, 13]
SecMult	Multiplication of additive shares	d-SNI	[7, 2]

Table 3.1: Gadgets from literature

(b_0, b_1) of b as input and return additive masking (z_0, z_1) of $x + y$ if $b = b_0 + b_1 = 1$ else independent masking (z_0, z_1) of x if $b = 0$.

Algorithm 9 CondAdd

Input: Additive masking (x_0, x_1) of $x \in \mathbb{F}_q$, (y_0, y_1) of $y \in \mathbb{F}_q$; boolean masking (b_0, b_1) of $b \in \{0, 1\}$

Output: Additive masking (z_0, z_1) of $x + y$ if $b = 0$; x otherwise.

1: $s_0 := x_0 + (y_0 \wedge mask_0)$ ▷ $mask_0 = b_0 \dots b_0 (\log q$ times)
2: $s_1 := x_1 + (y_1 \wedge mask_0)$
3: Pick a uniformly random $r \in \mathbb{F}_q$
4: $t_0 := s_0 + r$
5: $t_1 := s_1 + r$
6: $w_0 := y_0 \wedge mask_1$ ▷ $mask_1 = b_1 \dots b_1 (\log q$ times)
7: $w_1 := y_1 \wedge mask_1$
8: $z_0 := t_0 + w_0$
9: $z_1 := t_1 + w_1$
10: **return** (z_0, z_1)

Correctness.

$$\begin{aligned} z_0 + z_1 &= (t_0 + t_1) + (y_0 + y_1) \wedge mask_1 \\ &= ((x_0 + r + x_1 + r) + (y_0 + y_1) \wedge mask_0) + (y_0 + y_1) \wedge mask_1 \\ &= (x_0 + x_1) + (y_0 + y_1) \wedge (mask_0 + mask_1) \\ &= \begin{cases} x + y, & \text{if } b = 1 \\ x, & \text{otherwise} \end{cases} \end{aligned}$$

Security.

Lemma 3 *CondAdd is* 1*-SNI.*

Proof:

We need to show that each variable in $\{x_0, x_1, y_0, y_1, b_0, b_1\}$ (input shares) and $\{s_0, s_1, r, t_0, t_1, w_0, w_1\}$ (internal variables) can be simulated using at most one share from each $\{x_0, x_1\}$, $\{y_0, y_1\}$, and $\{b_0, b_1\}$, and each output share in $\{z_0, z_1\}$ can be simulated without any knowledge of inputs. Simulating any input share is straightforward.

s_0 can be simulated with x_0, y_0, and b_0. Similarly, s_1. r is random. t_0 (resp. t_1) is random, as it is a sum of s_0 (resp. s_1) and a random value r. w_0 can be simulated with y_0 and b_1. Similarly, w_1. z_0 (resp. z_1) is also random, as it is a sum of w_0 (resp. w_1) and a random value t_0 (resp. t_1). □

Complexity. The CondAdd gadget requires one random value in $\mathbb{F}_q$.

3.2.2 AMtoMMinv

We need to calculate the multiplicative inverse of non-zero pivot elements to transform one matrix to its row echelon form. The exponential operation is easier in multiplicative masking than in additive masking, so we convert additive masking to multiplicative masking and perform the inverse operation. AMtoMMinv takes the additive masking (x_0, x_1) of $x \in \mathbb{F}_q^*$ as input and returns (p_0, p_1) such that $x^{-1} = p_0 p_1$. Gadget RowEchelon checks whether a pivot element is non-zero. If a pivot element is zero even after trying to make it non-zero, RowEchelon aborts; otherwise, it calls AMtoMMinv.

Algorithm 10 AMtoMMinv

Input: Additive masking (x_0, x_1) of $x \in \mathbb{F}_q^*$
Output: (p_0, p_1) such that $x^{-1} = p_0 p_1$

1: $(z_0, z_1) :=$ AMtoMM(x_0, x_1)
2: $p_0 := z_0^{-1}$ ▷ z_0^{-1} is multiplicative inverse of z_0
3: $p_1 := z_1$
4: **return** (p_0, p_1)

Correctness. $p_0 p_1 = z_0^{-1} z_1 = \left(z_0 z_1^{-1}\right)^{-1} = (x_0 + (-x_1))^{-1} = x^{-1}$.

Security.

Lemma 4 *AMtoMMinv is* 1*-SNI.*

Proof:

AMtoMM is 1-SNI. The multiplicative inverse is applied only on z_0, independent of z_1. □

Complexity. The AMtoMMinv implicitly calls the AMtoMM gadget, requiring one non-zero random value in $\mathbb{F}_q$.

3.2.3 ScalarMult

Gadget ScalarMult takes additive masking (x_0, x_1) of $x \in \mathbb{F}$ and (p_0, p_1) such that $p_0 p_1 = p \in \mathbb{F}^*$ as input and return additive masking (y_0, y_1) of px. This gadget is used to multiply one row of a matrix by a non-zero scalar value. Alternatively, we could apply MMtoAM [9] on (p_0, p_1^{-1}) then call SecMult, but ScalarMult is lighter computationally and provides the same 1-SNI security.

Algorithm 11 ScalarMult

Input: Additive masking (x_0, x_1) of $x \in \mathbb{F}_q$; (p_0, p_1) such that $p = p_0 p_1 \in \mathbb{F}_q^*$
Output: Additive masking (y_0, y_1) of $px \in \mathbb{F}_q$

1: $s_0 := p_0 x_0$
2: $s_1 := p_0 x_1$
3: Pick a uniformly random $r \in \mathbb{F}_q$
4: $t_0 := s_0 + r$
5: $t_1 := s_1 + r$
6: $y_0 := p_1 t_0$
7: $y_1 := p_1 t_1$
8: **return** (y_0, y_1)

Correctness. $y_0 + y_1 = p_1(t_0 + t_1) = p_1(p_0 x_0 + r + p_0 x_1 + r) = p_0 p_1(x_0 + x_1) = px$.

Security.

Lemma 5 *ScalarMult is 1-SNI.*

Proof:

We need to show that each variable in $\{x_0, x_1, p_0, p_1\}$ (input shares), $\{s_0, s_1, r, t_0, t_1\}$ (internal variables) can be simulated using at most one share from each $\{x_0, x_1\}$ and $\{p_0, p_1\}$, and each output share in $\{y_0, y_1\}$ can be simulated without any knowledge of inputs. Simulating any input share is straightforward.

s_0 can be simulated with p_0 and x_0. Similarly, s_1. r is random. t_0 (resp. t_1) is random, as it is a sum of s_0 (resp. s_1) and a random value r. y_0 (resp. y_1) is random as it is a multiplication of a random value with a non-zero value p_1. □

Complexity. ScalarMult requires one random value in $\mathbb{F}_q$.

3.2.4 MultAdd

MultAdd takes additive mashing (x_0, x_1) of $x \in \mathbb{F}$, $(y_0.y_1)$ of $y \in \mathbb{F}$ and (c_0, c_1) of $c \in \mathbb{F}$ as input and return additive masking (z_0, z_1) of $x + cy$. This gadget is used to multiply one row by a scalar first and then add this row to another row.

Algorithm 12 MultAdd

Input: Additive masking (x_0, x_1) of $x \in \mathbb{F}_q$, (y_0, y_1) of $y \in \mathbb{F}_q$ and (c_0, c_1) of $c \in \mathbb{F}$
Output: Additive masking (z_0, z_1) of $x + cy$
1: $(t_0, t_1) := \text{SecMult}((y_0, y_1), (c_0, c_1))$
2: $z_0 := x_0 + t_0$
3: $z_1 := x_1 + t_1$
4: **return** (z_0, z_1)

Security.

Lemma 6 *MultAdd is* 1*-SNI.*

Proof:
Since SecMult is 1-SNI, t_0 (resp. t_1) is random. z_0 (resp. z_1) is random as it is a sum of x_0 (resp. x_1) and a random value. □

Complexity. MultAdd implicitly calls the SecMult gadget, requiring one random value in $\mathbb{F}_q$.

3.2.5 RowEchelon

To sign a message in the UOV Signature scheme, we need to solve a system of linear equations. We use the Gaussian elimination method, which reduces the augmented matrix of linear equations to its row echelon form. RowEcholon gadget takes additive masking $(\mathbf{A}_0, \mathbf{A}_1)$ of a matrix $\mathbf{A} \in \mathbb{F}_q^{m \times m}$ and $(\mathbf{b}_0, \mathbf{b}_1)$ of a vector $\mathbf{b} \in \mathbb{F}_q^m$ and transforms each share such that unmasked $[\mathbf{A} \mid \mathbf{b}]$ is in row echelon form or abort if one of pivot is zero after unsuccessful attempt to make non-zero, which cause mostly when $\det \mathbf{A} = 0$.

Correctness. To transform a matrix to its row echelon form, we follow the following steps:

- Step 1 (Lines 4-11). Instead of interchanging two rows, the original implementation [5] performs conditional addition of the first row with a certain number of rows below to make the pivot element of the row non-zero. SecNonZero gadget returns boolean masking of $b \in \{0, 1\}$ such that $b = 0$ if $a_{ii} = 0$. In line

Algorithm 13 RowEchelon

Input: Additive maskings $(\mathbf{A}_0, \mathbf{A}_1)$ of $\mathbf{A} \in \mathbb{F}_q^{m \times m}$ and $(\mathbf{b}_0, \mathbf{b}_1)$ of $\mathbf{b} \in \mathbb{F}_q^m$
Output: transform to row echelon form or $\perp$

1: $\mathbf{A}' := [\mathbf{A}_0 \mid \mathbf{b}_0] \in \mathbb{F}_q^{m \times (m+1)}$ ▷ $\mathbf{A}' = [a_{ij0}]$
2: $\mathbf{A}'' := [\mathbf{A}_1 \mid \mathbf{b}_1] \in \mathbb{F}_q^{m \times (m+1)}$ ▷ $\mathbf{A}'' = [a_{ij1}]$
3: **for** $i = 0$ upto $m - 1$ **do**
    /* try to make pivot non-zero */
4: $stop := (i + l < m)\ ?\ i + l\ :\ m$ ▷ $l = 8$ if $q = 256$; 16 if $q = 16$
5: **for** $j = i + 1$ upto $stop - 1$ **do**
6: $(b_{i0}, b_{i1}) := \text{SecNonzero}(a_{ii0}, a_{ii1})$
7: $b_{i0} := \neg b_{i0}$
8: **for** $k = i$ upto m **do**
9: $(a_{ik0}, a_{ik1}) := \text{CondAdd}((a_{ik0}, a_{ik1}), (a_{jk0}, a_{jk1}), (b_{i0}, b_{i1}))$
10: **end for**
11: **end for**
    /* check if pivot is non-zero */
12: $(c_{i0}, c_{i1}) := \text{SecNonzero}(a_{ii0}, a_{ii1})$
13: $c_i := \text{FullXor}(c_{i0}, c_{i1})$
14: **if** $c_i == 0$ **then**
15: **return** $\perp$
16: **end if**
    /* multiplication of a row by the inverse of its pivot */
17: $(p_{i0}, p_{i1}) := \text{AMtoMMinv}(a_{ii0}, a_{ii1})$
18: **for** $k = i$ upto m **do**
19: $(a_{ik0}, a_{ik1}) := \text{ScalarMult}((a_{ik0}, a_{ik1}), (p_{i0}, p_{i1}))$
20: **end for**
    /* addition of a scalar multiple of one row to another row */
21: **for** $j = i + 1$ upto $m - 1$ **do**
22: **for** $k = i$ upto m **do**
23: $(a_{jk0}, a_{jk1}) := \text{MultAdd}((a_{jk0}, a_{jk1}), (a_{ik0}, a_{ik1}), (a_{ji0}, a_{ji1}))$
24: **end for**
25: **end for**
26: **end for**
27: **return** $((\mathbf{A}_0, \mathbf{A}_1), (\mathbf{b}_0, \mathbf{b}_1))$

7, we flip b_{i0} so that we get new b such that $b = 1$ if $a_{ii} = 0$. Then, we call the CondAdd gadget in lines 8-10 to get the exact same effect of the original conditional addition of rows.

- Step 2 (Lines 12-16). After conditional addition, the original implementation checks whether the pivot element is still zero. If it's zero, then abort; else, proceed to the next step. We call the SecNonzero gadget with additive masking of a_{ii} as input, then unmask the output c_i. We consider c_i as public output. If we observe c_i equals zero, we abort as the original implementation does. Otherwise, we proceed to Step 3.

- Step 3 (Lines 17-20). In this step, we call the AMtoMMinv gadget with additive masking of the non-zero pivot element as input and get shares of the multiplicative inverse. Then, we call the ScalarMult gadget to multiply a row by the multiplicative inverse of the non-zero pivot element of this row.

- Step 4 (Lines 21-25). We use the MultAdd gadget to add a scalar multiple of the first row to the current row to make the first element of the current row zero in masked form. We repeat this from the second row to the last using the for loop in line 21.

After successfully executing the above steps, we ignore the first row and first column of the matrix and repeat the above steps till the last row using the for loop in line 3.

Security.

Lemma 7 *RowEchelon is* 1*-NI with public output* $\{c_i\}_{0\le i\le m-1}$.

If we observe c_i equals zero in line 13, we abort as the original implementation does. Otherwise, c_i is always non-zero. So we can consider $\{c_i\}_{0\le i\le m-1}$ as public output. We bit-flip independently the first share of the output of 1-SNI gadget SecNonzero in line 7, so we can consider lines 6-7 as 1-SNI gadget.

Complexity. RowEchelon requires $\mathcal{O}(m^3 \log q)$ random bits, using table 3.2.

3.2.6 BackSub

If the RowEchelon gadget executes successfully, then $\det \mathbf{A} \neq 0$. We perform back substitution to get the unique solution. BackSub gadget takes additive masking $(\mathbf{A}_0, \mathbf{A}_1)$ of a matrix $\mathbf{A} \in \mathbb{F}_q^{m\times m}$ and $(\mathbf{b}_0, \mathbf{b}_1)$ of a vector $\mathbf{b} \in \mathbb{F}_q^m$ where $[\mathbf{A} \mid \mathbf{b}]$ is in row echelon form as input and return unique $\mathbf{x} \in \mathbb{F}_q^m$ such that $\mathbf{Ax} = \mathbf{b}$.

Gadget	Required random bits per call	Number of calls
SecNonZero	$2(\log q - 1)$	$7(m-4)+m$
CondAdd	$\log q$	$7(m-7)(m+10)/2+133$
FullXor	1	m
AMtoMMinv	$\mathcal{O}(\log q)$	m
ScalarMult	$\log q$	$m(m+3)/2$
MultAdd	$\log q$	$(m-1)m(2m+5)/6$

Table 3.2: Required random bits in RowEchelon

Algorithm 14 BackSub

Input: Additive maskings $(\mathbf{A}_0, \mathbf{A}_1)$ of $\mathbf{A} \in \mathbb{F}_q^{m\times m}$ in row echelon form with $\det \mathbf{A} \neq 0$ and $(\mathbf{b}_0, \mathbf{b}_1)$ of $\mathbf{b} \in \mathbb{F}_q^m$
Output: Unique $\mathbf{x} \in \mathbb{F}_q^m$ such that $\mathbf{Ax} = \mathbf{b}$

1: **for** $i = m-1$ downto 1 **do**
2: $\quad x_i = \text{FullXor}(b_{i0}, b_{i1})$ ▷ $\mathbf{b}_0 = (b_{i0}), \mathbf{b}_1 = (b_{i1})$
3: $\quad$ **for** $j = 0$ upto $i-1$ **do**
4: $\quad\quad b_{j0} := b_{j0} + x_i a_{ji0}$ ▷ $\mathbf{A}_0 = [a_{ij0}]$
5: $\quad\quad b_{j1} := b_{j1} + x_i a_{ji1}$ ▷ $\mathbf{A}_1 = [a_{ij1}]$
6: $\quad$ **end for**
7: **end for**
8: $x_0 = \text{FullXor}(b_{00}, b_{01})$
9: **return x** ▷ $\mathbf{x} = (x_i)$

Correctness. Since $[\mathbf{A} \mid \mathbf{b}]$ is in row echelon form, back substitution is equivalent to multiplying first i values of i-th column of $[\mathbf{A} \mid \mathbf{b}]$ with x_i then add to first i values of $\mathbf{b}$ for all i such that $m-1 \geq i \geq 1$.

Security.

Lemma 8 *BackSub is* 1*-NI with public output* $\mathbf{x}$*.*

Proof:

We need to show that each share in $\{\mathbf{A}_0, \mathbf{A}_1, \mathbf{b}_0, \mathbf{b}_1\}$ (input shares), each internal variable of FullXor and updated share from $\{\mathbf{b}_0, \mathbf{b}_1\}$ in lines 4-5 can be simulated using at most one share from each $\{\mathbf{A}_0, \mathbf{A}_1\}$, $\{\mathbf{b}_0, \mathbf{b}_1\}$, and public output $\mathbf{x}$. Simulating any input share is straightforward.

In lines 2 and 8, we compute public output x_i using the FullXor gadget. Since it is 1-NI, each internal share of it can be simulated with at most one share from $\{b_{i1}, b_{i2}\}$. In line 4, one column of $\mathbf{A}_0$ is multiplied by public output and added to $\mathbf{b}_0$, so it can be simulated with $\mathbf{A}_0$, $\mathbf{b}_0$, and public output. Similarly, in line 5. □

Complexity. Backsub implicitly calls FullXor gadget m times, requiring m random value in $\mathbb{F}_q$.

3.2.7 SecDotProd

SecDotProd takes additive masking $(\mathbf{x}_0, \mathbf{x}_1)$ of a vector $\mathbf{x} \in \mathbb{F}_q^l$ and $(\mathbf{y}_0, \mathbf{y}_1)$ of another vector $\mathbf{y} \in \mathbb{F}_q^l$ and returns additive masking (z_0, z_1) of $\mathbf{x}^T\mathbf{y}$. We built this gadget using inspiration from SecAnd. This gadget is used as a subgadget of gadgets like SecMatVec and SecQuad.

Algorithm 15 SecDotProd

Input: Additive masking $(\mathbf{x}_0, \mathbf{x}_1)$ of $\mathbf{x} \in \mathbb{F}_q^l$ and $(\mathbf{y}_0, \mathbf{y}_1)$ of $\mathbf{y} \in \mathbb{F}_q^l$
Output: Additive masking (z_0, z_1) of $\mathbf{x}^T\mathbf{y}$

1: $(s, t, u, v) := (0, 0, 0, 0)$
2: **for** $i = 0$ upto $l - 1$ **do**
3: $\quad s := s + x_{i0}y_{i0}$ $\quad \triangleright\ \mathbf{x}_0 = (x_{i0}), \mathbf{y}_0 = (y_{i0})$
4: $\quad t := t + x_{i1}y_{i1}$ $\quad \triangleright\ \mathbf{x}_1 = (x_{i1}), \mathbf{y}_1 = (y_{i1})$
5: $\quad u := u + x_{i0}y_{i1}$
6: $\quad v := v + x_{i1}y_{i0}$
7: **end for**
8: Pick a uniformly random $r \in \mathbb{F}_q$
9: $z_0 := s + r$
10: $z_1 := t + ((u + r) + v)$
11: **return** (z_0, z_1)

Security.

Lemma 9 *SecDotProd is* 1*-SNI.*

Proof:

We need to show that each variable in $\{\mathbf{x}_0, \mathbf{x}_1, \mathbf{y}_0, \mathbf{y}_1\}$ (input shares) and $\{s, t, u, v, r\}$ (internal variables) can be simulated using at most one share from each $\{\mathbf{x}_0, \mathbf{x}_1\}$, $\{\mathbf{y}_0, \mathbf{y}_1\}$, and each output share in $\{z_0, z_0\}$ can be simulated without any knowledge of inputs. Simulating any input share is straightforward.

Each variable in $\{s, t, u, v\}$ can be simulated using one share of $\mathbf{x}$ and one share of $\mathbf{y}$. r is uniformly random. z_0 is random since it is a sum of two values, one of which is random. Similarly, z_1 is also random. □

Complexity. SecDotProd requires one random value in $\mathbb{F}_q$.

3.2.8 SecMatVec

SecMatVec takes additive masking $(\mathbf{A}_0, \mathbf{A}_1)$ of a matrix $\mathbf{A} \in \mathbb{F}_q^{m \times l}$ and $(\mathbf{x}_0, \mathbf{x}_1)$ of a vector $\mathbf{x} \in \mathbb{F}_q^l$ as input and return additive masking $(\mathbf{b}_0, \mathbf{b}_1)$ of $\mathbf{Ax} \in \mathbb{F}_q^m$.

Algorithm 16 SecMatVec

Input: Additive masking $(\mathbf{A}_0, \mathbf{A}_1)$ of $\mathbf{A} \in \mathbb{F}_q^{m \times l}$ and $(\mathbf{x}_0, \mathbf{x}_1)$ of $\mathbf{x} \in \mathbb{F}_q^l$
Output: Additive masking $(\mathbf{b}_0, \mathbf{b}_1)$ of $\mathbf{Ax} \in \mathbb{F}_q^m$
1: **for** $i = 0$ upto $m - 1$ **do**
2: $(b_{i0}, b_{i1}) :=$ SecDotProd$((i$-th row of $\mathbf{A}_0, i$-th row of $\mathbf{A}_1), (\mathbf{x}_0, \mathbf{x}_1))$
3: **end for**
4: **return** $(\mathbf{b}_0, \mathbf{b}_1)$ ▷ $\mathbf{b}_j = (b_{ij})$

Security.

Lemma 10 *SecMatVec is* 1*-SNI.*

Proof:

We need to show that each variable in $\{\mathbf{A}_0, \mathbf{A}_1, \mathbf{x}_0, \mathbf{x}_1\}$ (input shares) and each internal variable of SecDotProd can be simulated using at most one share from each $\{\mathbf{A}_0, \mathbf{A}_1\}$ and $\{\mathbf{x}_0, \mathbf{x}_1\}$, and each output share in $\{\mathbf{b}_0, \mathbf{b}_1\}$ can be simulated without any knowledge of inputs. Simulating any input share is straightforward.

Since SecDotProd is 1-SNI (by Lemma 9), each internal variable of it can be simulated using almost one share from $\{\mathbf{x}_0, \mathbf{x}_1\}$ and (one row of) one share from $\{\mathbf{A}_0, \mathbf{A}_1\}$; output share b_{i0} (resp. b_{i1}) is random. So, $\mathbf{b}_0$ (resp. $\mathbf{b}_1$) is a random vector. □

Complexity. SecMatVec implicitly calls the SecDotProd gadget m times, requires m random value in $\mathbb{F}_q$.

3.2.9 SecQuad

SecQuad gadget takes m public matrix $\{\mathbf{P}_k \in \mathbb{F}_q^{l \times l}\}_{0 \le k \le m-1}$ and additive masking $(\mathbf{x}_0, \mathbf{x}_1)$ of a vector $\mathbf{x} \in \mathbb{F}_q^l$ as input; returns additive masking $(\mathbf{y}_0, \mathbf{y}_1)$ of $\mathbf{y} = [\mathbf{x}^T \mathbf{P}_k \mathbf{x}]_{0 \le k \le m-1} \in \mathbb{F}_q^m$.

Correctness. $y_{k0} + y_{k1} = (\mathbf{s}_0 + \mathbf{s}_1)^T (\mathbf{t}_{k0} + \mathbf{t}_{k1}) = (\mathbf{x}_0 + \mathbf{x}_1)^T (\mathbf{P}_k \mathbf{x}_0 + \mathbf{P}_k \mathbf{x}_1) = \mathbf{x}^T \mathbf{P}_k \mathbf{x}$

Security.

Lemma 11 *SecQuad is* 1*-SNI.*

Algorithm 17 SecQuad

Input: Public $\{\mathbf{P}_k \in \mathbb{F}_q^{l\times l}\}_{0\le k\le m-1}$; additive masking $(\mathbf{x}_0, \mathbf{x}_1)$ of $\mathbf{x} \in \mathbb{F}_q^l$
Output: Additive masking $(\mathbf{y}_0, \mathbf{y}_1)$ of $\mathbf{y} = [\mathbf{x}^T\mathbf{P}_k\mathbf{x}]_{0\le k\le m-1} \in \mathbb{F}_q^m$

1: **for** $i = 0$ upto $l-1$ **do**
2: $(s_{i0}, s_{i1}) := \text{Refresh}(x_{i0}, x_{i1})$ ▷ $\mathbf{x}_j = (x_{ij}), \mathbf{s}_j = (s_{ij})$
3: **end for**
4: **for** $k = 0$ up to $m-1$ **do**
5: $(\mathbf{t}_{k0}, \mathbf{t}_{k1}) := (\mathbf{P}_k\mathbf{x}_0, \mathbf{P}_k\mathbf{x}_1)$
6: $(y_{k0}, y_{k1}) := \text{SecDotProd}((\mathbf{s}_0, \mathbf{s}_1), (\mathbf{t}_{k0}, \mathbf{t}_{k1}))$ ▷ $\mathbf{y}_j = (y_{ij})$
7: **end for**
8: **return** $(\mathbf{y}_0, \mathbf{y}_1)$

Proof:

We need to show that each variable in $\{\mathbf{x}_0, \mathbf{x}_1\}$ (input shares), $\{\mathbf{s}_0, \mathbf{s}_1, \mathbf{t}_{i0}, \mathbf{t}_{i1}\}$ (internal variables) and each internal variable of Refresh and SecDotProd can be simulated using at most one share from $\{\mathbf{x}_0, \mathbf{x}_1\}$, and each output share in $\{\mathbf{y}_0, \mathbf{y}_1\}$ can be simulated without any knowledge of inputs. Simulating any input share is straightforward.

- Since Refresh is 1-SNI, each internal variable of it can be simulated with almost one share from $\{\mathbf{x}_0, \mathbf{x}_1\}$, and each output share in $\{\mathbf{s}_0, \mathbf{s}_1\}$ is random.
- $\mathbf{t}_{k0}$ (resp. $\mathbf{t}_{k1}$) is a linear combinations of $\mathbf{x}_0$ (resp. $\mathbf{x}_1$), so can be simulated with $\mathbf{x}_0$ (resp. $\mathbf{x}_1$).
- Since SecDotProd is 1-SNI (by Lemma 9), each internal variable of it can be simulated with almost one share from each $\{\mathbf{t}_{k0}, \mathbf{t}_{k1}\}$ (which can be simulated with at most one share from $\{\mathbf{x}_0.\mathbf{x}_1\}$) and $\{\mathbf{s}_0, \mathbf{s}_1\}$ (but we can replace $\mathbf{s}_i$ with a random value); output share y_{k0} (resp. y_{k1}) is random. So, $\mathbf{y}_0$ (resp. $\mathbf{y}_1$) is a random vector.

□

Complexity. SecQuad implicitly calls Refresh gadget l times and SecDotProd gadget m times, requiring $l + m$ random values in F_q.

3.2.10 MaskedCompactKeyGen

MaskedCompactKeyGen gadget is a first-order masked implementation of the UOV CompactKeyGen algorithm. It takes a security parameter *params* as input and returns compact representation of public and masked secret key pair $(cpk, mcsk)$.

Algorithm 18 MaskedCompactKeyGen

Input: Security parameter $params$
Output: Compact representation of public and masked secret key pair $(cpk, mcsk)$

Pick a uniformly random $seed_{sk0} \in \{0,1\}^{sk\_seed\_len}$
Pick a uniformly random $seed_{sk1} \in \{0,1\}^{sk\_seed\_len}$
Pick a uniformly random $seed_{pk} \in \{0,1\}^{pk\_seed\_len}$
$(\mathbf{O}_0, \mathbf{O}_1) := \text{MaskedExpand}_{sk}(seed_{sk0}, seed_{sk1})$
$\{\mathbf{P}_i^{(0)}, \mathbf{P}_i^{(1)}\}_{0 \le i \le m-1} := \text{Expand}_{\mathbf{P}}(seed_{pk})$
$(\mathbf{Q}_0, \mathbf{Q}_1) := \text{Refresh}(\mathbf{O}_0, \mathbf{O}_1)$
for $i = 0$ upto $m-1$ **do**
    $\mathbf{A}_{i0} := -\mathbf{P}_i^{(0)}\mathbf{O}_0 - \mathbf{P}_i^{(1)}$
    $\mathbf{A}_{i1} := -\mathbf{P}_i^{(0)}\mathbf{O}_1$
    $(\mathbf{B}_{i0}, \mathbf{B}_{i1}) := (\mathbf{0}_{m\times m}, \mathbf{0}_{m\times m})$
    for $k = 0$ upto $m-1$ **do**
        Set k-th column of $\mathbf{B}_{i0}, \mathbf{B}_{i1}$ to
            $\text{SecMatVec}((\mathbf{A}_{i0}^T, \mathbf{A}_{i1}^T), (k\text{-th column of } \mathbf{Q}_0, k\text{-th column of } \mathbf{Q}_1))$
    end for
    $\mathbf{C}_{i0} := \text{Upper}(\mathbf{B}_{i0})$
    $\mathbf{C}_{i1} := \text{Upper}(\mathbf{B}_{i1})$
    $\mathbf{P}_i^{(3)} := \text{FullXor}(\mathbf{C}_{i0}, \mathbf{C}_{i1})$
end for
$cpk := \left(seed_{pk}, \left\{\mathbf{P}_i^{(3)}\right\}_{0 \le i \le m-1}\right)$
$mcsk := (seed_{pk}, (seed_{sk0}, seed_{sk1}))$
return $(cpk, mcsk)$

Security. There is a first-order masked implementation of shake256 in literature [11]. We assume gadget MaskedExpand$_{sk}$ is 1-NI.

Theorem 1 *Gadget MaskedCompactKeyGen is* 1*-NI.*

Proof:

We produce the proof using the traditional "from right to left" approach. As $\mathbf{P}_i^{(0)}$ and $\mathbf{P}_i^{(1)}$ are part of public key, $\mathbf{A}_{i0}$ (resp. $\mathbf{A}_{i1}$) is linear combination of $\mathbf{O}_0$ (resp. $\mathbf{O}_1$). So, we can consider lines 8-9 as a 1-NI gadget. The function Upper [5] acts on an individual share. So, we can consider lines 14-15 as a 1-NI gadget.

Let us assume that an attacker accesses δ_0 probe on MaskedExpand$_{sk}$, δ_1 probe on Refresh, δ_2 probe on gadget in lines 8-9, δ_3 probe on SecMatVec, δ_4 probe on gadget in line 14-15, and δ_5 probe on FullXor such that $\sum_{i=0}^{5} \delta_i = \delta \leq 1$. We need to show that δ probe can be simulated with at most δ share from $\{seed_{sk0}, seed_{sk1}\}$.

Since gadget FullXor is 1-NI, δ_5 probe on it can be simulated with at most δ_5 share from $\{\mathbf{C}_{i0}, \mathbf{C}_{i1}\}$.

Since gadget in lines 14-15 is 1-NI, all probes on it can be simulated with at most $\delta_4 + \delta_5$ share from $\{\mathbf{B}_{i0}, \mathbf{B}_{i1}\}$.

Since gadget SecMatVec is 1-SNI, all probes on it can be simulated with at most δ_3 share from each $\{\mathbf{A}_{i0}, \mathbf{A}_{i1}\}$ and (one column of) $\{\mathbf{Q}_0, \mathbf{Q}_1\}$.

Since gadget in lines 8-9 is 1-NI, all probes on it can be simulated with at most $\delta_2 + \delta_3$ share from $\{\mathbf{O}_1, \mathbf{O}_2\}$.

Since gadget Refresh is 1-SNI, all probes on it can be simulated with at most δ_1 share from $\{\mathbf{O}_1, \mathbf{O}_2\}$.

Gadget MaskedExpand$_{sk}$ is 1-NI. At the end of MaskedExpand$_{sk}$, the simulation relies on at most $\sum_{i=1}^{3} \delta_i$ share from $\{seed_{sk0}, seed_{sk1}\}$. Additionally, δ_0 probes on MaskedExpand$_{sk}$ can simulated with at most δ_0 share from $\{seed_{sk0}, seed_{sk1}\}$. Therefore, we are done. □

Gadget	Required random bits per call	Number of calls
Refresh	$m(n-m)\log q$	1
SecMatVec	$m \log q$	m^2
FullXor	$m^2(m+1)\log q/2$	1

Table 3.3: Required random bits in MaskedCompactKeyGen

Complexity. MaskedCompactKeyGen requires $\mathcal{O}(m^3 \log q)$ random bits, using table 3.3, apart from randomness used by MaskedExpand$_{sk}$ gadget.

3.2.11 MaskedExpandSK

MaskedExpandSk gadget is a first-order masked implementation of the UOV ExandSk algorithm. It takes a masked compact representation of a secret key $mcsk$, i.e. public seed $seed_{pk}$ and additive masking $(seed_{sk0}, seed_{sk1})$ of secret seed $seed_{sk}$ as input and returns expanded representation of masked secret key $mesk$, i.e. additive masking $(seed_{sk0}, seed_{sk1})$ of $seed_{sk}$, $(\mathbf{O}_0.\mathbf{O}_1)$ of $\mathbf{O}$, $\{(\mathbf{S}_{i0}, \mathbf{S}_{i1})\}_{0\leq i\leq m-1}$ of $\{\mathbf{S}_i\}_{0\leq i\leq m-1}$, and public $\{\mathbf{P}_i^{(0)}\}_{0\leq i\leq m-1}$.

Algorithm 19 MaskedExpandSK

Input: Compact representation of masked secret key
$mcsk = (seed_{pk}, (seed_{sk0}, seed_{sk1}))$
Output: Expanded representation of masked secret key
$mesk = ((seed_{sk0}, seed_{sk1}), (\mathbf{O}_0, \mathbf{O}_1), \{\mathbf{P}_i^{(0)}, (\mathbf{S}_{i0}, \mathbf{S}_{i1})\}_{0\leq i\leq m-1})$

1: $(\mathbf{O}_0, \mathbf{O}_1) :=$ MaskedExpand$_{sk}(seed_{sk0}, seed_{sk1})$
2: $\{\mathbf{P}_i^{(0)}, \mathbf{P}_i^{(1)}\}_{0\leq i\leq m-1} :=$ Expand$_{\mathbf{P}}(seed_{pk})$
3: **for** $i = 0$ upto $m-1$ **do**
4: $\quad \mathbf{S}_{i0} := \left(\mathbf{P}_i^{(0)} + \mathbf{P}_i^{(0)T}\right)\mathbf{O}_0 + \mathbf{P}_i^{(1)}$
5: $\quad \mathbf{S}_{i1} := \left(\mathbf{P}_i^{(0)} + \mathbf{P}_i^{(0)T}\right)\mathbf{O}_1$
6: **end for**
7: $mesk := ((seed_{sk0}, seed_{sk1}), (\mathbf{O}_0, \mathbf{O}_1), \{\mathbf{P}_i^{(0)}, (\mathbf{S}_{i0}, \mathbf{S}_{i1})\}_{0\leq i\leq m-1})$
8: **return** $mesk$

Security. We assume gadget MaskedExpand$_{sk}$ is 1-NI.

Theorem 2 *MaskedExpandSK is* 1*-NI.*

Proof:

We produce the proof using the traditional "from right to left" approach. As $\mathbf{P}_i^{(0)}$ and $\mathbf{P}_i^{(1)}$ are part of public key, $\mathbf{S}_{i0}$ (resp. $\mathbf{S}_{i1}$) is linear combination of $\mathbf{O}_0$ (resp. $\mathbf{O}_1$). So, we can consider lines 4-5 as a 1-NI gadget.

Let us assume that an attacker accesses δ_0 probe on MaskedExpand$_{sk}$ and δ_1 probe on gadget in lines 4-5 such that $\delta_0 + \delta_1 = \delta \leq 1$. We need to show that δ probe can be simulated with at most δ share from $\{seed_{sk0}, seed_{sk1}\}$.

Since gadget in lines 4-5 is 1-NI, δ_1 probe on it can be simulated with δ_1 share from $\{\mathbf{O}_0, \mathbf{O}_1\}$.

Gadget MaskedExpand$_{sk}$ is 1-NI. At the end of MaskedExpand$_{sk}$, the simulation relies on δ_1 share from $\{seed_{sk0}, seed_{sk1}\}$. Additionally, δ_0 probes on MaskedExpand$_{sk}$ can be simulated with at most δ_0 share from $\{seed_{sk0}, seed_{sk1}\}$. Therefore, we are done. □

Complexity. MaskedExpandSk does not require any random bits apart from randomness used by MaskedExpand_{sk}.

3.2.12 MaskedSign

MaskedSign gadget is a first-order masked implementation of the UOV sign algorithm. It takes a expanded representation of a masked secret key $mesk$, i.e. additive masking $(seed_{sk0}, seed_{sk1})$ of secret seed $seed_{sk}$, $(\mathbf{O}_0.\mathbf{O}_1)$ of $\mathbf{O}$, $\{(\mathbf{S}_{i0}, \mathbf{S}_{i1})\}_{0\le i\le m-1}$ of $\{\mathbf{S}_i\}_{0\le i\le m-1}$, public $\{\mathbf{P}_i^{(0)}\}_{0\le i\le m-1}$ and message μ as input and returns signature σ of the message.

Algorithm 20 MaskedSign

Input: Expanded representation of masked secret key
$mesk = ((seed_{sk0}, seed_{sk1}), (\mathbf{O}_0, \mathbf{O}_1), \{\mathbf{P}_i^{(0)}, (\mathbf{S}_{i0}, \mathbf{S}_{i1})\}_{0\le i\le m-1})$, message μ
Output: Signature σ

Pick a uniformly random $salt \in \{0,1\}^{salt\_len}$
$\mathbf{t} := \text{Hash}(\mu || salt)$
for $ctr = 0$ upto 255 **do**
  $(\mathbf{v}_0, \mathbf{v}_1) := \text{MaskedExpand}_{\mathbf{v}}(\mu, salt, (seed_{sk0}, seed_{sk1}), ctr)$ ▷ $\mathbf{v}_j \in \mathbb{F}_q^{n-m}$
  $(\mathbf{L}_0, \mathbf{L}_1) := (\mathbf{0}_{m\times m}, \mathbf{0}_{m\times m})$
  for $i = 0$ upto $m-1$ **do**
    Set i-columns of $\mathbf{L}_0, \mathbf{L}_1$ to $\text{SecMatVec}((\mathbf{S}_{i0}^T, \mathbf{S}_{i1}^T), (\mathbf{v}_0, \mathbf{v}_1))$
  end for
  $(\mathbf{y}_0, \mathbf{y}_1) := \text{SecQuad}(\{\mathbf{P}_i^{(0)}\}_{0\le i\le m-1}, (\mathbf{v}_0, \mathbf{v}_1))$
  $((\mathbf{M}_0, \mathbf{M}_1), (\mathbf{z}_0, \mathbf{z}_1)) := \text{RowEchelon}((\mathbf{L}_0, \mathbf{L}_1), (\mathbf{t} + \mathbf{y}_0, \mathbf{y}_1))$
  if $((\mathbf{M}_0, \mathbf{M}_1), (\mathbf{z}_0, \mathbf{z}_1)) \neq \perp$ **then**
    $\mathbf{x} := \text{BackSub}((\mathbf{M}_0, \mathbf{M}_1), (\mathbf{z}_0, \mathbf{z}_1))$
    $\mathbf{s} := \begin{bmatrix} \mathbf{0}_{n-m} \\ \mathbf{x} \end{bmatrix}$ ▷ $\mathbf{s} = (s_i) \in \mathbb{F}_q^n$
    $\mathbf{u}_0 := \mathbf{v}_0 + \mathbf{O}_0\mathbf{x}$
    $\mathbf{u}_1 := \mathbf{v}_1 + \mathbf{O}_1\mathbf{x}$
    $\mathbf{s} := \text{FullXor}(\mathbf{u}_0, \mathbf{u}_1)$
    $\sigma := (\mathbf{s}, salt)$
    return σ
  end if
end for
return $\perp$

Security. We assume gadget $\text{Expand}_{\mathbf{v}}$ is 1-SNI and gadget MaskedExpand_{sk} is 1-NI.

Theorem 3 *MaskedSign is* 1*-NI with public outputs* $\{c_i\}_{0\le i\le m}$ *and* $\mathbf{x}$.

Proof:

We produce the proof using the traditional "from right to left" approach. As $\mathbf{u}_0$ (resp. $\mathbf{u}_1$) can be simulated with public output $\mathbf{x}$ and $\mathbf{v}_0$ (resp. $\mathbf{v}_1$) and $\mathbf{O}_0$ (resp. $\mathbf{O}_1$), we can consider lines 14-15 as a 1-NI gadget with public output $\mathbf{x}$.

Let us assume that an attacker accesses δ_0 probe on MaskedExpand$_{\mathbf{v}}$, δ_1 probe on SecMatVec, δ_2 probe on SecQuad, δ_3 probe on RowEchelon, δ_4 probe on Backsub, δ_5 probe on gadget in line 14-15, and δ_6 probe on FullXor such that $\sum_{i=0}^{6} \delta_i = \delta \leq 1$.

Since gadget FullXor is 1-NI, δ_6 probe on it can be simulated with at most δ_6 share from $\{\mathbf{u}_0, \mathbf{u}_1\}$.

Since gadget in lines 14-15 is 1-NI, all probes on it can be simulated with at most $\delta_5 + \delta_6$ share from each $\{\mathbf{v}_0, \mathbf{v}_1\}$, $\{\mathbf{O}_0, \mathbf{O}_1\}$ and public $\mathbf{x}$.

Since gadget Backsub is 1-NI with public output $\mathbf{x}$, δ_4 probe on it can be simulated with at most δ_4 share from each $\{\mathbf{M}_0, \mathbf{M}_1\}$, $\{\mathbf{z}_0, \mathbf{z}_1\}$ and public output $\mathbf{x}$.

Since gadget RowEchlon is 1-NI with public output $\{c_i\}_{0 \leq i \leq m}$, all probes on it can be simulated with at most $\delta_3 + \delta_4$ share from each $\{\mathbf{L}_0, \mathbf{L}_1\}$, $\{\mathbf{y}_0, \mathbf{y}_1\}$ and public $\mathbf{t}$.

Since gadget SecQuad is 1-SNI, all probes on it can be simulated with at most δ_2 share from $\{\mathbf{v}_0, \mathbf{v}_1\}$ and public key.

Since gadget SecMatVec is 1-SNI, all probes on it can be simulated with at most δ_1 share from each $\{\mathbf{S}_{i0}, \mathbf{S}_{i1}\}$ and $\{\mathbf{v}_0, \mathbf{v}_1\}$.

Since gadget MaskedExpand$_{\mathbf{v}}$ is 1-SNI, all probes on it can be simulated with at most δ_0 share from $\{seed_{sk0}, seed_{sk1}\}$.

Therefore, all δ probes can be simulated with at most δ_0 share from $\{seed_{sk0}, seed_{sk1}\}$, δ_1 share from $\{\mathbf{S}_{i0}.\mathbf{S}_{i1}\}$, and $\delta_5+\delta_6$ share from $\{\mathbf{O}_0, \mathbf{O}_1\}$, i.e., $\delta_0+\delta_1+\delta_5+\delta_6 \leq \delta$ share from $\{seed_{sk0}, seed_{sk1}\}$. So we are done. □

Gadget	Required random bits per call	Number of calls
SecMatVec	$m \log q$	$\mathcal{O}(m)$
SecQuad	$n \log q$	$\mathcal{O}(1)$
RowEchelon	$\mathcal{O}(m^3 \log q)$	$\mathcal{O}(1)$
BackSub	$m \log q$	1
FullXor	$(n-m) \log q$	1

Table 3.4: Required random bits in MaskedSign

Complexity. MaskedSign requires $\mathcal{O}(m^3 \log q)$ random bits, using table 3.4, apart from randomness used by MaskedExpand$_{\mathbf{v}}$ and MaskedExpand$_{sk}$ gadgets.

Chapter 4

Conclusion and Future Work

In this thesis, we describe first-order masking of the NIST first-round submission UOV digital signature scheme and prove our implementation is 1-NI secure with public output. We will assess leakage information from the power/electromagnetic trace of our implementation using TVLA methodology and performance in terms of clock cycles. We believe that the gadgets we build can be extended naturally to any order. We observe that RowEchlon required $\mathcal{O}(m^3 \log q)$ random bits. Investing in a more efficient alternative to the RowEchlon gadget could be an excellent future work.

Although our implementation is provably secure in the 1-probe model, it could still be possible to recover both the share of the vinegar vector $\mathbf{v}_0, \mathbf{v}_1$ using the attack strategy in [1] due to the mathematical structure of the UOV scheme and the gf256v_mul_u32 algorithm in implementation. So, we recommend mixing countermeasures suggested in [1] with our implementation, such as picking a uniformly random $r \in \{0, 1, \ldots, 7\}$ inside _gf256v_madd_u32 and replacing gf256v_mul_u32 with its shuffled version while computing $\mathbf{P}_k\mathbf{v}_0, \mathbf{P}_k\mathbf{v}_1$ in the SecQuad gadget. We reproduce Shuffled_gf256v_mul_u32 in Appendix B.

Appendix A

Further Algorithms

Algorithm 21 SecAnd [8]

Input: Additive/boolean masking (x_0, x_1) of $x \in \mathbb{F}_q$, (y_0, y_1) of $y \in \mathbb{F}_q$
Output: Additive/boolean masking (z_0, z_1) of $x \wedge y$

1: $z_0 := x_0 \wedge y_0$
2: $z_1 := x_1 \wedge y_1$
3: Pick a uniformly random $r \in \mathbb{F}_q$
4: $s := (x_0 \wedge y_1 + r) + x_1 \wedge y_0$
5: $z_0 := z_0 + r$
6: $z_1 := z_1 + s$
7: **return** (z_0, z_1)

Algorithm 22 SecOr [6]

Input: Additive/boolean masking (x_0, x_1) of $x \in \mathbb{F}_q$, (y_0, y_1) of $y \in \mathbb{F}_q$
Output: Additive/boolean masking (z_0, z_1) of $x \vee y$

1: $(t_0, t_1) := (\neg x_0, x_1)$
2: $(s_o, s_1) := (\neg y_0, y_1)$
3: $(z_0, z_1) := \text{SecAnd}((s_0, s_1), (t_0, t_1))$
4: $z_0 := \neg z_0$
5: **return** (z_0, z_1)

Algorithm 23 Refresh [2]

Input: Additive masking (x_0, x_1) of $x \in \mathbb{F}_q$
Output: Independent additive masking (y_0, y_1) of x

Pick a uniformly random $r \in \mathbb{F}_q$
$y_0 := x_0 + r$
$y_1 := x_1 + r$
return (y_0, y_1)

Algorithm 24 SecNonzero [6]

Input: Additive/boolean shares (x_0, x_1) of $x \in \mathbb{F}_q$
Output: Boolean shares (b_0, b_1) of b such that $b = b_0 + b_1 = 0 \iff x = 0$

$(t_0, t_1) := (x_0, x_1)$
$len := (\log q)/2$
while $len \geq 1$ **do**
    $(l_0, l_1) := \text{Refresh}((t_0^{[2len:len]}, t_1^{[2len:len]}), len)$
    $(r_0, r_1) := (t_0^{[len:1]}, t_1^{[len:1]})$
    $(t_0, t_1) := \text{SecOr}((l_0, l_1), (r_0, r_1))$
    $len := len \gg 1$
end while
return $(t_0^{[1:1]}, t_1^{[1:1]})$

Algorithm 25 FullXor [8]

Input: Additive masking (x_0, x_1) of $x \in \mathbb{F}_q$
Output: The unmasked value x

$(y_0, y_1) := \text{Refresh}(x_0, x_1)$
return $y_0 + y_1$

Algorithm 26 AMtoMM [9]

Input: Additive masking (x_0, x_1) of $x \in \mathbb{F}_q^*$
Output: Multiplicative masking (z_0, z_1) of x

Pick a uniformly random $z_1 \in \mathbb{F}_q^*$
$y_0 := x_0 z_1$
$y_1 := x_1 z_1$
$z_0 := y_0 + y_1$
return (z_0, z_1)

Algorithm 27 SecMult [7]

Input: Additive masking (x_0, x_1) of $x \in \mathbb{F}_q$, (y_0, y_1) of $y \in \mathbb{F}_q$
Output: Additive masking (z_0, z_1) of xy

1: $z_0 := x_0 y_0$
2: $z_1 := x_1 y_1$
3: Pick a uniformly random $r \in \mathbb{F}_q$
4: $s := (x_0 y_1 + r) + x_1 y_0$
5: $z_0 := z_0 + r$
6: $z_1 := z_1 + s$
7: **return** (z_0, z_1)

Appendix B

Shuffled gf256v_mul_u32

Algorithm 28 Shuffled_gf256v_mul_u32 [1]

Input: $uint32\_t\ \alpha, uint8\_t\ x, size\_t\ r$
Output: $uint32\_t\ \alpha x$

1: **for** $i = 0$ upto 7 **do**
2: $j := i + r \pmod 8$
3: **if** $((x \gg j) \wedge 1)$ **then**
4: $ret := ret + \alpha$
5: $tmp := tmp + 0\text{x}0$
6: **else**
7: $tmp := tmp + \alpha$
8: $ret := ret + 0\text{x}0$
9: **end if**
10: $\alpha\_msb := \alpha \wedge 0\text{x}80808080$
11: $\alpha := \alpha + \alpha\_msb$
12: $\alpha := (\alpha \ll 1) + ((\alpha_{msb} \gg 7)0\text{x}1b)$
13: **end for**
14: **return** ret

Bibliography

[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 Transactions on Cryptographic Hardware and Embedded Systems 2023(3), 221–245 (2023)

[2] 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: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 116–129 (2016)

[3] 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)

[4] Belaïd, S., Benhamouda, F., Passelègue, A., Prouff, E., Thillard, A., Vergnaud, D.: Private multiplication over finite fields. In: Annual International Cryptology Conference. pp. 397–426. Springer (2017)

[5] Beullens, W., Chen, M., Ding, J., Gong, B., Kannwischer, M., Patarin, J., Peng, B., Schmidt, D., Shih, C., Tao, C., et al.: Uov: Unbalanced oil and vinegar algorithm specifications and supporting documentation version 1.0 (2018)

[6] 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)

[7] Coron, J.S.: Higher order masking of look-up tables. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 441–458. Springer (2014)

[8] 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)

[9] Genelle, L., Prouff, E., Quisquater, M.: Thwarting higher-order side channel analysis with additive and multiplicative maskings. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 240–255. Springer (2011)

[10] Gilbert Goodwill, B.J., Jaffe, J., Rohatgi, P., et al.: A testing methodology for side-channel resistance validation. In: NIST non-invasive attack testing workshop. vol. 7, pp. 115–136 (2011)

[11] Heinz, D., Kannwischer, M.J., Land, G., Pöppelmann, T., Schwabe, P., Sprenkels, A.: First-order masked kyber on arm cortex-m4. Cryptology ePrint Archive (2022)

[12] Kipnis, A., Patarin, J., Goubin, L.: Unbalanced oil and vinegar signature schemes. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 206–222. Springer (1999)

[13] Mathieu-Mahias, A., Quisquater, M.: Mixing additive and multiplicative masking for probing secure polynomial evaluation methods. IACR Transactions on Cryptographic Hardware and Embedded Systems 2018(1), 175–208 (2018)

[14] NIST: Call for additional digital signature schemes for the post-quantum cryptography standardization process (2022)

[15] Park, A., Shim, K.A., Koo, N., Han, D.G.: Side-channel attacks on post-quantum signature schemes based on multivariate quadratic equations:-rainbow and uov. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 500–523 (2018)

[16] Pokornỳ, D., Socha, P., Novotnỳ, M.: Side-channel attack on rainbow post-quantum signature. In: 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 565–568. IEEE (2021)