# QUANTUM SAFE DIGITAL SIGNATURES FROM SYMMETRIC KEY PRIMITIVES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF APPLIED MATHEMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ŞEYMA ERBAŞ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
CRYPTOGRAPHY

SEPTEMBER 2019

Approval of the thesis:

**QUANTUM SAFE DIGITAL SIGNATURES FROM SYMMETRIC KEY PRIMITIVES**

submitted by **ŞEYMA ERBAŞ** in partial fulfillment of the requirements for the degree of **Master of Science in Cryptography Department, Middle East Technical University** by,

Prof. Dr. Ömür Uğur _____
Director, Graduate School of **Applied Mathematics**

Prof. Dr. Ferruh Özbudak _____
Head of Department, **Cryptography**

Assoc. Prof. Dr. Murat Cenk _____
Supervisor, **Cryptography, METU**

**Examining Committee Members:**

Assoc. Prof. Dr. Ali Doğanaksoy
Mathematics Department, METU _____

Assoc. Prof. Dr. Murat Cenk
Cryptography Department, METU _____

Assist. Prof. Dr. Eda Tekin
Business Administration, Karabük University _____

**Date:** _____

**I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.**

Name, Last Name: ŞEYMA ERBAŞ

Signature :

# ABSTRACT

QUANTUM SAFE DIGITAL SIGNATURES FROM SYMMETRIC KEY PRIMITIVES

Erbaş, Şeyma

M.S., Department of Cryptography

Supervisor : Assoc. Prof. Dr. Murat Cenk

September 2019, 41 pages

When powerful quantum computers are built, they will break most of the public key cryptography schemes due to Shor's quantum algorithm. Therefore, public key cryptography algorithm schemes that is secure against classical and quantum computers are needed. In this thesis, we study Picnic algorithm, a post-quantum digital signature scheme. Picnic digital signature algorithm has the security of symmetric-key primitives that is considered to be secure against quantum attacks. In Picnic algorithm, zero knowledge proof systems and circuits to compute their protocol are used.

Keywords: Post quantum cryptography, multiparty computation, zero-knowledge proof, LowMC, Picnic.

# ÖZ

SİMETRİK ANAHTAR TEMELLİ KUANTUM GÜVENLİ SAYISAL İMZALAR

Erbaş, Şeyma
Yüksek Lisans, Kriptografi Bölümü
Tez Yöneticisi : Doç. Dr. Murat Cenk

Eylül 2019, 41 sayfa

Kuantum bilgisayarlarının icadıyla birlikte Shor'un kuantum algoritması sayesinde günümüzde kullanılan açık anahtar şifreleme yöntemlerinin güvenli olmayacaktır. Bu yüzden hem klasik hemde kuantum saldırılarına karşı güvenli olan açık anahtar şifreleme yöntemleri gereklidir. Bu tezde, Picnic sayısal imza algoritmasını inceleyeceğiz. Picnic algoritması güvenliğini kuantum saldırılarına karşı güvenli olduğu düşünülen simetrik anahtar temellerinden(özet fonksiyonları ve blok şifreleme algoritmaları) alır. Picnic algoritmasında sıfır bilgi ispat yöntemi kullanılmaktadır.

Anahtar Kelimeler: Kuantum-sonrası kriptografi, LowMC, Picnic.

# ACKNOWLEDGMENTS

I would like to express my very great appreciation to my thesis supervisor Assoc. Prof. Dr. Murat Cenk for his guidance, enthusiastic encouragement and valuable advices during the development and preparation of this thesis. His willingness to give his time and to share his experiences has brightened my path. It was a great pleasure to work with him.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF ALGORITHMS

# LIST OF ABBREVIATIONS

| | |
|---|---|
| FS | Fiat-Shamir transform |
| UR | Unruh transform |
| MPC | Multiparty computation |
| PQC | Post-quantum cryptography |
| ROM | Random oracle model |
| QROM | Quantum-accessible random oracle model |
| PRNG | Pseudorandom number generator |
| SPN | Substitution-Permutation Network |

# CHAPTER 1

# INTRODUCTION

Cryptography has been important for secure communication since ancient times. As we know it, it started with a basic encryption scheme Caesar Cipher [30]. Later on, mechanical cryptosystems have been used for many years. Most famous mechanical cryptosystem was Enigma [34] used in world war II. Today, communication, shopping, banking etc. can be used securely with computers thanks to cryptography.

Cryptography uses two different methods to achieve security; symmetric key cryptography and public key cryptography. Symmetric key cryptography has fast encryption schemes, such as AES [18]. They use the same key when encrypting the plaintext and decrypting the ciphertext. However, we need to share this key before encryption with the parties that will communicate each other and symmetric key cryptography is unable to perform the key exchange [31]. Therefore, we need public key cryptography. Public key cryptography has relatively slow encryption schemes, such as RSA [24]. They mostly based on factoring large numbers and computing discrete logarithms [23]. Public key cryptography uses different keys when encrypting the plaintext(secret key) and decrypting the ciphertext(public key). This method includes key exchange and digital signature algorithms.

If we look at the future, when powerful quantum computers are built, they will break our public key cryptography schemes due to Shor's quantum algorithm [33]. This algorithm can compute discrete logarithms and factor large numbers in polynomial time. We will not be able to use most of the current public key encryption schemes in quantum computers. Thus, algorithms that is secure against attacks by both quantum and classical computers are needed before quantum computers are built. In order

not to rush the process, National Institute of Standards and Technology(NIST) has announced a post-quantum cryptography competition in 2017 [2]. The aim of the competition is to built new standards about post quantum cryptography. Public key cryptography algorithm proposals that is secure against classical and quantum computers were accepted for the competition.

We can study digital signature algorithms with post quantum(PQ) security in five categories.

- Hash-based digital signature schemes
- Code-based digital signature schemes
- Lattice-based digital signature schemes
- MQ-based digital signature schemes
- Supersingular Isogeny signature schemes

Among all the signature schemes, hash-based signatures are seem to be the preferred signatures because they don't require hard algebraic problems to have security against quantum computers, as well as they have minimal security requirements [13].

In this thesis, we study Picnic algorithm, a new class of PQ digital signature scheme. Picnic digital signature algorithm has the security of symmetric-key primitives(hash functions and block ciphers) that is considered to be secure against quantum attacks. Picnic has small key pairs, parameterizable structure and it does not require hard algebraic problems.

In Picnic signature algorithm, public key($y$) can be derived from secret key($x$) over a secure one way function($f$) as $y = f(x)$. Signature is the secret key's non-interactive zero knowledge proof. For this purpose, zero knowledge proof systems ($\Sigma$ protocols by Giacomelli at al.) are used [22]. Zero knowledge proof systems uses circuits to compute the protocol. This technique is based on the "Multiparty computation-in-the-head" paradigm to zero-knowledge of Ishai et al [26].

To make the proof non-interactive, Fiat-Shamir(FS) transform and Unruh's(UR) transform can be used. FS transform provides security in random oracle model(ROM) [19]

while UR transform provides security in quantum random oracle model(QROM) [15].

In this thesis, we will study following topics:

- In Chapter 2, we introduce some background information including zero knowledge proof systems.
- In Chapter 3, we mention about cryptographic components that Picnic algorithm uses including hash functions and block ciphers.
- In Chapter 4, we explain the Picnic digital signature scheme.
- In Chapter 5, we compare Picnic algorithm with other post quantum schemes.
- Chapter 6 is the conlusion chapter.

# CHAPTER 2

# BACKGROUND INFORMATION

In this section, we mention background information that Picnic algorithm uses. In Chapter 2.1, symmetric key primitives is described. In Chapter 2.2, multiparty computation(MPC) protocol and ZKBoo, a zero knowledge proof system that uses MPC protocol as a base is explained. In Chapter 2.3, ZKB++, an improved version of ZKBoo, is described with optimizations over ZKBoo.

## 2.1 Symmetric Key Primitives

Symmetric key primitives use a secret information to keep the secure interaction between parties that is communicating each other. This secret information is called key. All parties use the same shared key. For this reason, it is called symmetric key primitives. If someone other than authorized parties learns the key, communication security will fail immediately.

Symmetric key primitives is used to create secure encryption schemes, signature schemes and protocols such as TLS and IPsec.

Asymmetric key primitives use a pair of keys. A private key is only belong to the one party and public key is open to anyone who will communicate with this party. Asymmetric key primitives can be used even if parties don't have any shared information. They are much slower than symmetric key primitives. They generally used for key exchange. Thus, when building a secure communication, both symmetric and asymmetric key primitives are used. Asymmetric key primitives are used to share the secret

key for the symmetric key primitives to use and maintain secure communication.

Block ciphers are the most famous symmetric key primitives. Block ciphers are mostly used to provide confidentiality of the communication. A block cipher maps a block of an input to the block of an output with the help of the secret key. The design of the block cipher affects both security and efficiency of the cryptographic system. Block ciphers are believed to be secure against attacks by both classical and quantum computers [29].

In Picnic digital signature algorithm, LowMC, a block cipher family described in Chapter 3.1, is used as a symmetric key primitive. However, other block ciphers can be used such as AES. AES is well-studied cipher compared to LowMC. However, AES leads larger signatures and computation time [20].

Picnic also uses hash functions(SHA-3 family described in Chapter 3.2) as a symmetric key primitive. Hash functions are also believed to be quantum secure.

## 2.2 ZKBOO Protocol

Zero knowledge proof system allow the signer to convince the verifier that the signer knows the secret key for signature generation. That is, signer only needs to prove the knowledge of having the secret key. ZKBoo [25], a zero knowledge proof system, is based on multiparty computation protocol(MPC protocol). It has both interactive and non-interactive versions. We will only recall the non-interactive one because signature scheme requires non-interactive version. Interactive version is described in [25]. Firstly, let's recall the MPC protocol briefly.

**MPC Protocol:** In multiparty computation, input is the witness. Let say $y = f(x)$ where $f$ is a secure hash function. MPC can calculate $y$(public) and $x$(secret) is the witness in this equation. Each player have a share of $x$. Signer models MPC "in the head" paradigm of Ishai et al [27]. He commits to the state and transcripts of the players. Then, signer opens a random subset of these commitments. Now, verifier checks whether the calculation is done correctly or not. If so, he has some confidence that signer knows the witness. If they do this calculation many rounds, then verifier

will have higher confidence that signer knows the witness.

ZKBoo takes this idea as a base but generalizes it by using "circuit decomposition" instead of multiparty computation. Circuit decomposition is a more efficient protocol in practice because it does not need to have all properties of multiparty computation [16]. In circuit decomposition, number of players are three which means that witness will be divided into three.

The signer aims to prove that he knows the witness for $f(x) = y$. First, he starts with a circuit to calculate $f$. Then he uses a circuit decomposition. To do this, he needs five functions.
**Share function** divides the witness($x$) into three shares.
Three **Output**$_{i\in\{1,2,3\}}$ **function** randomize and finalize each view($w_1, w_2, w_3$).
**Reconstruct function** builds the last output($y$) from three shares.

In circuit decomposition, the view is updated with the output wire value for each gate. When the view is serialized, it includes

- The input share
- Output values for binary multiplication gates
- The output share

Circuit decomposition starts with signer calculates $f$ using decomposition. He commits to the views and calculates the challenge by using random oracle that needs a commitment and output shares as an input. After that, he needs to open two views for each run. Remember that opening two third of the views does not weaken the proof since it does not give any information about the witness. Now, verifier can check the following;

- Each opened views are correct.
- Challenge is correct.
- y can be computed by using output of the views.

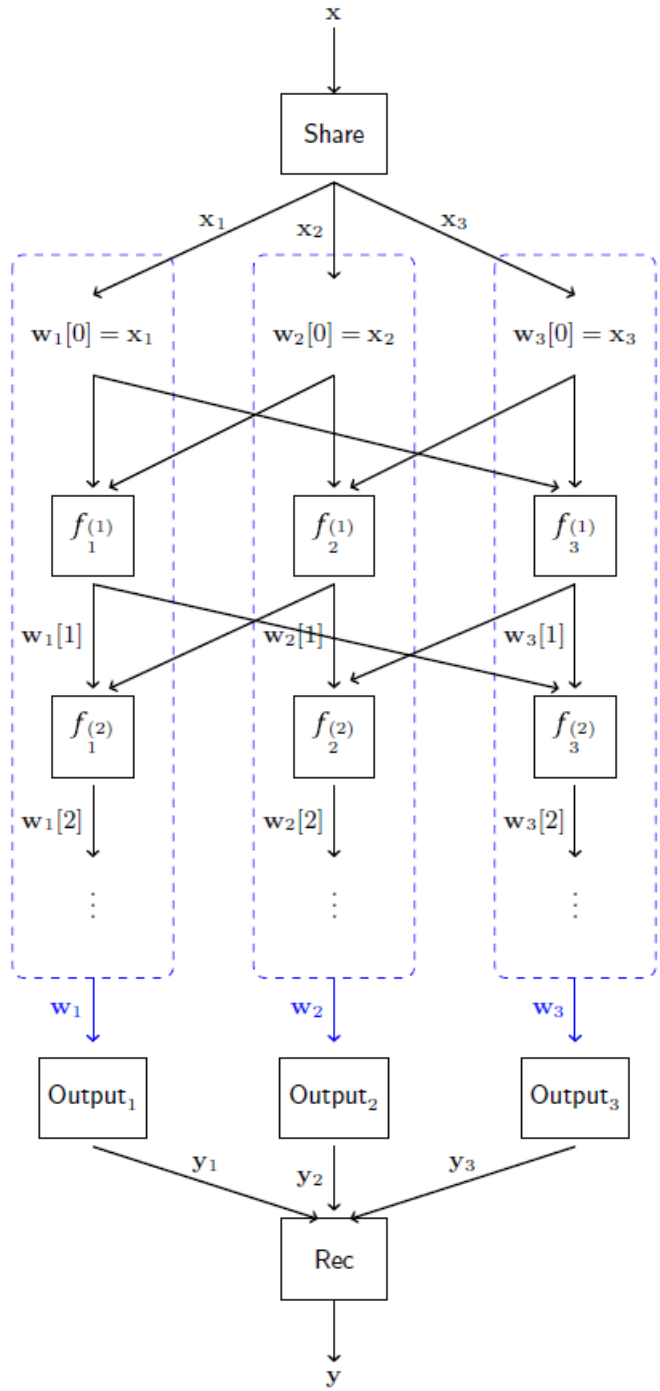


Figure 2.1: Representation of a circuit decomposition of the computation $y = f(x)$

- $(x_1, x_2, x_3) \leftarrow$ Share$(x, k_1, k_2, k_3)$ such that $x_1 + x_2 + x_3 = x$ where $k_1, k_2, k_3$ are random tapes corresponding to each player respectively.
- $y_i \leftarrow$ Output$_i(w_i)$ where $1 \leq i \leq 3$
- $y \leftarrow$ Reconstruct$(y_1, y_2, y_3) = y_1 + y_2 + y_3$

Figure 2.1 visualizes how circuit decomposition works. In this decomposition, even if two shares are exposed, witness $x$ will remain secret.

Mathematical representation of the ZKBoo is the following:

Verifier and the signer have $y$. The signer knows $x$ such that $y = f(x)$. Signer does the following:

- Sample random tapes $k_1, k_2, k_3$
- Run the protocol to obtain views $w_1, w_2, w_3$ and output shares $y_1, y_2, y_3$
- Commit to $c_i = Com(k_i, w_i)$ for $1 \leq i \leq 3$
- Using the FS heuristic, send $a = (y_1, y_2, y_3, c_1, c_2, c_3)$ to the random oracle to compute the challenge($e$) – the challenge tells the signer which two of the three views to open.
- Send $c_e, c_{e+1}$ to verifier. Thus revealing $z = (k_e, w_e, k_{e+1}, w_{e+1})$

Verifier does the following:

- If $Rec(y_1, y_2, y_3) \neq y$, reject
- If $y_i \neq Output_i(w_i)$ for $i \in \{e, e+1\}$, reject
- If $w_e[j] \neq f_e^{(j)}(w_e, w_{e+1}, k_e, k_{e+1})$, reject
- Output accept

Detailed information and security analysis about MPC protocol and ZKBoo is presented in [25].

### 2.3 ZKB++ Protocol

ZKB++ is an improved version of ZKBoo. Therefore, we will only mention about the ZKB++ optimizations over the ZKBoo protocol.

**Optimization 1:** Share function generates the pseudorandom shares as

$$(x_1, x_2, x_3) \leftarrow \text{Share}(x, k_1, k_2, k_3)$$
$$x_1 = R_1(0...|x-1|)$$
$$x_2 = R_2(0...|x-1|)$$
$$x_3 = x - x_1 - x_2$$

where $R_{i\in\{1,2\}}$ is the PRNG with the seed $k_{i\in\{1,2\}}$ and $(0..|x-1|)$ is the first $|x|$ bit string of $R_i$.

**Optimization 2:** When optimization 1 is applied, input shares computed pseudorandomly. Therefore we don't need to add the shares in the view if the challenge is 1. When challenge is 2 or 3, we need to add an input share for the third view.

**Optimization 3:** The commitments belong to the three views are sent to the verifier in the ZKBoo. However, verifier can calculate first two commitments by using opened views. Hence, it is enough to send only the third commitment.

**Optimization 4:** We don't need to add any randomization value for commitments. Since the first input is the seed value to calculate the commitments, commitments will be randomized.

**Optimization 5:** We don't need to send the output shares to the verifier since they can be computed from the opened views by the verifier.

**Optimization 6:** In ZKBoo, verifier calculates every wire in $view_e$ and verifies if it is correct or not. However, verifier doesn't have to check that each wire in $view_e$ is calculated correctly. It is enough for verifier to calculate view and verify the commitments using calculated view.

Note that all the inputs to compute challenge is given to the verifier in the ZKBoo so

the verifier can compute the challenge. In ZKB++, the challenge itself is given to the verifier explicitly.

Detailed information and security analysis about ZKB++ is presented in [16].

# CHAPTER 3

# CRYPTOGRAPHIC COMPONENTS

In this section, the cryptographic components that Picnic algorithm uses is described. In Chapter 3.1, LowMC, a blockcipher family is explained. In Chapter 3.2, hash functions and their usage in Picnic is described while in Chapter 3.3, other supporting functions and their usage in Picnic is explained with their pseudocode.

## 3.1 LowMC Function

LowMC is a very parameterizable block cipher family by Albrecht et al [10, 8]. The reason LowMC is the best candidate for Picnic algorithm is that it provides low AND depth as well as low multiplicative complexity. Many different trade-offs between size of the signature and consumed time can be obtained by using the different parameter sets of the LowMC algorithm. Hence, LowMC is very flexible as besides modifying the security parameter of the construction as required.

LowMC is built on an substitution-permutation network(SPN) structure. Security expectations and the number of S-boxes per round can be chosen to decrease the AND depth and the number of ANDs.

Let $n$ be the block size, $k$ be the key size, $m$ be the number of Sboxes, $d$ be the data complexity and $r$ be the number of rounds to reach the security requirements.

In the beginning of the LowMC encryption scheme, some key whitening is applied to the key. It is basically multiplying key with a key matrix chosen randomly. After key whitening, $r$ rounds of encryption is performed. A single round contains 4 main

layers.

$$\text{LowMC Round} = \begin{cases} \text{Sbox Layer} \\ \text{Linear Layer} \\ \text{Constant Addition} \\ \text{Key Addition} \end{cases}$$

In the **Sbox Layer**, 3-bit Sbox is applied to the first $3m$ bits of the state. It is recommended to minimize m, number of parallel Sboxes, in order to decrease the multiplicative complexity. The definition of the Sbox algorithm is given in the Chapter 3.3.

In the **Linear Layer**, state is multiplied with an binary and invertible $n \times n$ linear layer matrix. Linear layer matrices $L_i \leftarrow F_2^{n \times n}$ where $1 \leq i \leq r$ are chosen randomly and kept unchanged during the algorithm. The multiplication algorithm is given in the Chapter 3.3.

In the **Constant Addition Layer**, the state is simply added with the length $n$ binary vector round constants. Round constants $C_i \leftarrow F_2^n$ where $1 \leq i \leq r$ are chosen randomly and kept unchanged during the algorithm. The addition algorithm is given in the Chapter 3.3.

In the **Key Addition Layer**, state is added with the binary round key. Round key is generated by multiplying the master key with a binary $n \times k$ key matrix. Key matrices $K_i \leftarrow F_2^{n \times k}$ where $1 \leq i \leq r$ are chosen randomly and kept unchanged during the algorithm. The multiplication and addition algorithms are given in the Chapter 3.3.

Decryption is the inverse of these steps.

LowMC encryption algorithm is provided in Algorithm 1 for the signature generation and verification.

LowMC has simple circuit design. However other block cipher families can also be used in Picnic signature schemes, such as AES. AES is also believed to be secure against attacks by quantum computers and it is suited for MPC-in-the-head paradigm. It is well-studied cipher compared to LowMC. However, since AES has more complex circuits than LowMC, results show that signature sizes and computation time are larger with AES compared to LowMC [20].

**Algorithm 1:** LowMC Encryption ($K_i \in F_2^{n \times k}$ key matrices, $L_i \in F_2^{n \times n}$ linear layer matrices and $C_i \in F_2^n$ round constants for $1 \leq i \leq r$)

**input:** Plaintext $p \in F_2^n$ and key $y \in F_2^k$

$s \leftarrow K_0 \cdot y + p$

**for** *i from 1 to r* **do**

  $s \leftarrow \text{Sbox}(s)$

  $s \leftarrow L_i \cdot s$

  $s \leftarrow s + C_i + K_i \cdot y$

**end**

**return** $s$

### 3.2 Hash Function

Hash functions are used to map arbitrary-length messages to fix-length output strings. This output which is called hash value or message digest works like fingerprints of the message. Since the message digests are short representation of the messages, there will be many different messages that have the same message digest. However, for the hash function to be cryptographically secure, it must be infeaseable to find a collision between message digests in practice. To ensure this property, hash function must be collision resistant, as well as preimage resistant(one way) [1, 32].

Cryptographic hash functions are used in many digital signatures. We will mainly focus on the Secure Hash Algorithm-3 (SHA-3) family that have been selected by the NIST on NIST Hash Function Competition in August 2015 [3]. The SHA-3 functions are based on the Keccak sponge function, by G. Bertoni, J. Daemen, M. Peeters, G. Van Assche. SHA-3 family includes six functions, called SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128 and SHAKE256. SHA3-224, SHA3-256, SHA3-384 and SHA3-512 functions are hash functions and the numbers after the dashes indicates the output length of the related function. SHAKE128 and SHAKE256 functions are extendable-output functions (XOF). The difference between hash functions and XOF is that we can extend the output length of the XOF to any length. Thus, SHAKE128 and SHAKE256 functions are adaptable to the requirements of different algorithms [32, 3].

The hash functions mentioned in this thesis($H$) are SHAKE128 or SHAKE256. There are many hash computations when generating signatures. To differentiate the inputs, a fixed byte is prepended to the input of the related hash function as

$$H_i(x) = H(0\text{x}0i||x)$$

where $0 \leq i \leq 5$ [6].

### 3.3 Supporting Functions

MPC protocol and LowMC functions use supporting functions for signature generation and verification operations. In this section, these functions will be explained.

**First function is XOR operation($\oplus$).** This function takes shares $a[0...(m-1)][0...(L-1)]$, $b[0...(m-1)][0...(L-1)]$ as an input and calculates $c = a \oplus b$.

$\oplus$ function algorithm is provided in Algorithm 2 for the signature generation and verification. Note that, $\oplus$ operation can be performed without any necessity for interaction of players.

**Algorithm 2:** Multiparty computation - XOR($\oplus$) operation

**input:** m bit vectors $a[0...(m-1)][0...(L-1)]$ and
$b[0...(m-1)][0...(L-1)]$ of length $L$

**for** *i from 0 to* $m-1$ **do**
  $c[i] = a[i] \oplus b[i]$

**end**

**return** $c$

where $m = 3$ for signature generation and $m = 2$ for verification.

When $\oplus$ will be used with a constant, only one share will be XORed with this constant. During signature generation, the first secret share is XORed. During verification, the first share is XORed when $e_t = 0$, the second share is XORed when $e_t = 2$

**Second function is MPC AND operation($\wedge$).** This function takes shares $a[0...2]$, $b[0...2]$ as an input and calculates $c = a \wedge b$. To add some randomness, $\wedge$ function uses random tapes which are the other inputs. $transcript$ will also be updated by $\wedge$

function.

$\wedge$ function algorithm is provided in Algorithm 3 for the signature generation. Notice that, three shares for each of $a$ and $b$ are taken as input.

**Algorithm 3:** Multiparty computation - (signature generation) AND($\wedge$) operation

**input:** Random tapes $rand$, triple $views$ and secred-shared inputs $a[0...2], b[0...2]$

$r[0] = rand[0].nextbit()$

$r[1] = rand[1].nextbit()$

$r[2] = rand[2].nextbit()$

**for** *i from 0 to 2* **do**

$c[i] = (a[i] \wedge b[(i+1)\%3]) \oplus (a[(i+1)\%3] \wedge b[i]) \oplus (a[i] \wedge b[i]) \oplus r[i] \oplus r[(i+1)\%3]$

$views[i].transcript.append(c[i])$

**end**

**return** $c$ and $views$

where $.nextbit$ reads the tape's next bit.

$\wedge$ function algorithm is provided in Algorithm 4 for the verification. Notice that, two shares for each of $a$ and $b$ are taken as input.

**Algorithm 4:** Multiparty computation - (verification) AND($\wedge$) operation

**input:** Random tapes $rand$, pair of $views$ and secred-shared inputs $a[0...1], b[0...1]$

$r[0] = rand[0].nextbit()$

$r[1] = rand[1].nextbit()$

$c[0] = (a[0] \wedge b[1]) \oplus (a[1] \wedge b[0]) \oplus (a[0] \wedge b[0]) \oplus r[0] \oplus r[1]$

$views[0].transcript.append(c[0])$

$c[1] = views[1].transcript.nextbit()$

**return** $c$ and $views$

**Third function is LowMC Sbox algorithm.** This function takes $state$ as an input and calculates $S(a, b, c) = (a \oplus bc, a \oplus b \oplus ac, a \oplus b \oplus c \oplus ab)$. To add some randomness, LowMC Sbox algorithm uses random tapes which are the other inputs. MPC protocol

$transcript$ will also be updated by this function.

LowMC Sbox algorithm is provided in Algorithm 5 for the signature generation and verification. Notice that, algorithm uses zero-based and bitwise indexing.

**Algorithm 5:** LowMC Sbox Algorithm

**input:** LowMC shares $state$, random tapes $rand$ and triple $views$

**for** *i from 0 to* $3r - 1$ **do**

  **for** *j from 0 to 2* **do**

    $a[j] = state[j][i + 2]$

    $b[j] = state[j][i + 1]$

    $c[j] = state[j][i]$

  **end**

  $ab = a \wedge b \wedge rand \wedge views$

  $bc = b \wedge c \wedge rand \wedge views$

  $ca = c \wedge a \wedge rand \wedge views$

  **for** *j from 0 to 2* **do**

    $state[j][i + 2] = a[j] \oplus bc[j]$

    $state[j][i + 1] = a[j] \oplus b[j] \oplus ca[j]$

    $state[j][i] = a[j] \oplus b[j] \oplus c[j] \oplus ab[j]$

  **end**

**end**

**return** $state$

**The next function is vector-matrix multiplication algorithm.** This function takes three vectors $x, y, z$ in $GF(2)^n$ and a matrix $M$ in $GF(2)^{n \times n}$ as an input and calculates $xM, yM, zM$ in signature generation.

Vector-matrix multiplication algorithm is provided in Algorithm 6 for the signature generation. Note that, all the computations are calculated in $GF(2)$ and $M[i][j]$ shows the bit in the ($i$th row - $j$th column) of the matrix $M$.

Vector-matrix multiplication algorithm takes two vectors $x, y$ in $GF(2)^n$ and a matrix $M$ in $GF(2)^{n \times n}$ as an input and calculates $xM, yM$ in verification. This algorithm is provided in Algorithm 7 for the verification. Note that, all the computations are calculated in GF(2) and $M[i][j]$ shows the bit in the ($i$th row - $j$th column) of the

**Algorithm 6:** Binary Vector-Matrix Multiplication Algorithm - (signature generation)

**input:** $n$-bit vectors $x, y, z$ and $n \times n$ matrix $M$

**for** *i from 0 to* $n-1$ **do**

  **for** *j from 0 to* $n-1$ **do**

    $A[j] = x[j] \wedge M[i][j]$

    $B[j] = y[j] \wedge M[i][j]$

    $C[j] = z[j] \wedge M[i][j]$

  **end**

  $a[n-1-i] = A[0] \oplus A[1] \oplus ... \oplus A[n-1]$

  $b[n-1-i] = B[0] \oplus B[1] \oplus ... \oplus B[n-1]$

  $c[n-1-i] = C[0] \oplus C[1] \oplus ... \oplus C[n-1]$

**end**

**return** $a, b, c$

matrix $M$.

**Algorithm 7:** Binary Vector-Matrix Multiplication Algorithm - (verification)

**input:** $n$-bit vectors $x, y$ and $n \times n$ matrix $M$

**for** *i from 0 to* $n-1$ **do**

  **for** *j from 0 to* $n-1$ **do**

    $A[j] = x[j] \wedge M[i][j]$

    $B[j] = y[j] \wedge M[i][j]$

  **end**

  $a[n-1-i] = A[0] \oplus A[1] \oplus ... \oplus A[n-1]$

  $b[n-1-i] = B[0] \oplus B[1] \oplus ... \oplus B[n-1]$

**end**

**return** $a, b$

Since LowMC requires many matrix-vector multiplications, an efficient matrix-vector multiplication algorithm is needed to speed up the LowMC encryption scheme. Instead of this binary vector matrix multiplication algorithm, "Method of four Russians" algorithm seem to be the best approach for the LowMC matrix-vector multiplication operation [9, 12].

**The fifth function is the function to compute the challenge.** The function $H3$ takes a bit string of arbitrary length as an input and outputs a bit string of length $T$ which has the elements in $0, 1, 2$: $H3 : \{0,1\}^* \rightarrow \{0,1,2\}^t$

The algorithm to compute the challenge is provided in Algorithm 8 for the signature generation and verification [6]. Remember that, $H_1(x) = H(0\text{x}01||x)$ where $||$ denotes the concatenation.

**Algorithm 8:** Algorithm to compute the challenge

**input:** bit string $b$ of arbitrary length

$h = (h_0, h_1, ..., h_S) = H_1(b)$ where $h_0, h_1, ..., h_S \in \{0,1\}$.

flag

Create pairs $(h_0, h_1), (h_2, h_3), ...$

**if** *The pair* $(0,0)$ **then**
| append 0 to e,

**end**

**if** *The pair* $(0,1)$ **then**
| append 1 to e,

**end**

**if** *The pair* $(1,0)$ **then**
| append 2 to e,

**end**

**if** *The pair* $(1,1)$ **then**
| do nothing.

**end**

**if** $e$ *has length* $T$ **then**
| **return** $e$

**end**

**if** *All pairs are used up and* $e$ *has length that less than* $T$ **then**
| $h = H_1(h)$
| Return to flag

**end**

**The last function is the function $G$.** The function $G$ takes a $seed$ of length $S$ where $S$ is the security strength and a view $v$ of different lengths as an input. $G$ is computed with SHAKE function with the input

$$H_5(seed)||v||l_G$$

The output length is $l_G$ where $l_G =$ (length of the seed) + (length of the view) [6].

# CHAPTER 4

# THE PICNIC SIGNATURE ALGORITHM

In this section, Picnic signature algorithm is explained. In Chapter 4.1, different parameters for different security levels are given. In Chapter 4.2, private and public key generations are explained. In Chapter 4.3, signature generation algorithm is described in detail while Chapter 4.4 studies signature verification algorithm.

## 4.1 Parameters

Table 4.1 shows the parameter sets for different security levels. L1, L3 and L5 are three security levels as described in [4, 6]. For each security level, there are three algorithms.

- One based on zero knowledge proof system and Fiat-Shamir(FS) transform (picnic-L1-FS, picnic-L3-FS and picnic-L5-FS)
- One based on zero knowledge proof system and Unruh's(UR) transform (picnic-L1-UR, picnic-L3-UR and picnic-L5-UR)
- One based on the proof system described in [28] and Fiat-Shamir transform (picnic2-L1-FS, picnic2-L3-FS and picnic2-L5-FS)

FS transform and UN transform is used to make the proof non-interactive. FS transform provides security in random oracle model(ROM) [19] while UR transform provides security in quantum random oracle model(QROM) [15]. The differences between the FS and UR variants is described in detailed in [16]. Note that $S$-bit security

Table 4.1: Parameters by security level

| Parameter Set | $S$ | $n$ | $s$ | $r$ | Hash | $l_H$ | $T$ |
|---|---|---|---|---|---|---|---|
| picnic-L1-FS | | | | | | | 219 |
| picnic-L1-UR | 128 | 128 | 10 | 20 | SHAKE128 | 256 | 219 |
| picnic2-L1-FS | | | | | | | 343 |
| picnic-L3-FS | | | | | | | 329 |
| picnic-L3-UR | 192 | 192 | 10 | 30 | SHAKE256 | 384 | 329 |
| picnic2-L3-FS | | | | | | | 570 |
| picnic-L5-FS | | | | | | | 438 |
| picnic-L5-UR | 256 | 256 | 10 | 38 | SHAKE256 | 512 | 438 |
| picnic2-L5-FS | | | | | | | 803 |

level is for classical computers. For quantum computers, parameters provide $S/2$-bit security.

Notation used in the table is the following;

- $S$-bit is the security strength
- $n$-bit is LowMC key and blocksize
- $s$ is the number of s-boxes
- $r$ is the number of rounds in LowMC algorithm
- $l_H$-byte is the output length of hash function H
- $T$ is the number of parallel repetitions of the zero knowledge proof

In this thesis, we will mention about algorithms used with picnic parameter set and the leave the picnic2 parameter sets to [6].

## 4.2 Key Generation

A key pair consists of a secret key ($sk$) and a public key ($pk$). It is suggested that different key pairs should be used for multiple signature algorithms.

n-bit secret key is chosen randomly.

$$sk \in_R \{0,1\}^n$$

To compute the public key, an n-bit string $p$ is chosen randomly.

$$p \in_R \{0,1\}^n$$

Then, the encryption of $p$ with secret key is computed.

$$C = E(sk, p)$$

where $E$ is the LowMC encryption algorithm.

$sk$ is the picnic algorithm secret key and $pk = (C, p)$ is the public key.

Note that $view$ calculated in signature generation and verification has three components which are $view.iShare$(input key share), $view.transcript$(transcript of all communication) and $view.oShare$(output share).

### 4.3 Signature Generation

The signature generation takes $(sk, pk)$ and byte array $M$ as an input where $M \in [1, 2^{55}]$ is the message desired to be signed. Note that when the Unruh transform is applied, it is shown as UR in the context.

1. Let $T$ be the number of iterations of the signature generation algorithm and $seed[0...(T-1)][0..2]$ be the list of seeds. First, $seed$ are set $3T$ random seeds. Each seed is $S$ bits, where $S$ is the security level(One of $128$, $192$ or $256$).

2. Next, 256-bit $salt$ value are chosen randomly.

$$salt \in_R \{0,1\}^{256}$$

3. For each iteration $t$ where $0 \leq t \leq T-1$, following parameters are computed respectively.

   - Random tapes $rand[0..2]$ are computed using the hash function $H$.

$$rand[j] = H(\, H_2(seed[t][j]) \,||\, salt \,||\, t \,||\, j \,||\, \text{output\_length}\,)$$

   where $||$ denotes the concatenation and $0 \leq j \leq 2$.

- Let $x[0..2]$ be the $n$-bit shares of secret key. These three shares are computed using random tapes.

$$x[0] = \text{ first } n\text{-bits of } rand[0]$$
$$x[1] = \text{ first } n\text{-bits of } rand[1]$$
$$x[2] = sk \oplus x[0] \oplus x[1]$$

  where $\oplus$ denotes the multiparty computation binary exclusive or (XOR) operation.

- $key$ parameter is computed using the first key matrix.

$$key = x \cdot Kmatrix[0]$$

- Let $state[0..2]$ be the $n$-bit vectors. These three vectors are computed using $key$ parameter.

$$state = key \oplus p$$

- For each LowMC round $i$ where $1 \leq i \leq r$, following parameters are computed respectively.
  - First, the key shares are computed using the key matrices.

$$key = x \cdot Kmatrix[i]$$

  - Next, Sbox layer is computed using random tapes and views.

$$state = sbox(state, rand, views[t])$$

  - Affine layer is computed using linear layer matrices.

$$state = state \cdot Lmatrix[i-1]$$

  - Round constant addition is computed using round constants chosen randomly before the signature generation.

$$state = state \oplus roundconstant[i-1]$$

  - Round key addition is computed using key shares.

$$state = state \oplus key$$

- After computing all LowMC rounds, the output shares are updated using $state$ vectors.

$$views[t][i].oShare = state[i]$$

where $0 \leq i \leq 2$.

- Commitments are computed using $seed$ and $view$.

$$C[t][i] = H_0(H_4(seed[i]), view[i])$$

where $0 \leq i \leq 2$. If UR is applied,

$$G[t][i] = G(H_4(seed[i]), view[i])$$

where $0 \leq i \leq 2$.

4. After computing all $T$ iterations, the challenge $e$ can be written as

$$\begin{aligned} e = & H_3(view[0][0].oShare, view[0][1].oShare, view[0][2].oShare, \\ & ... \\ & view[t-1][0].oShare, view[t-1][1].oShare, view[t-1][2].oShare, \\ & C[0][0], C[0][1], C[0][2], \\ & ... \\ & C[t-1][0], C[t-1][1], C[t-1][2], \\ & [G[0][0], G[0][1], G[0][2], \\ & ... \\ & G[t-1][0], G[t-1][1], G[t-1][2], ] \\ & salt, pk, M) \end{aligned}$$

Note that "$G[i][j]$" is the commitments that is added when UR is applied and omitted otherwise. $e$ must be of the form

$$e = (e_0, ..., e_{t-1})$$

where $0 \leq e_k \leq 2$ and $0 \leq k \leq t-1$.

5. $i = e_t + 2(mod 3)$ is calculated for each $0 \leq t \leq T-1$ and $0 \leq e_t \leq 2$.
$b_t = C[t][i], [G[t][i]]$ is set.
Note that "$G[i][j]$" is added when UR is applied and omitted otherwise.

- When $e_t = 0$, $z_t$ is set to
  $view[t][1].transcript, seed[t][0], seed[t][1]$

- When $e_t = 1$, $z_t$ is set to
  $view[t][2].transcript, seed[t][1], seed[t][2], view[t][2].iShare$

- When $e_t = 2$, $z_t$ is set to
  $view[t][0].transcript, seed[t][2], seed[t][0], view[t][2].iShare$

6. The output is formed as $(e, salt, b_0, ..., b_t, z_0, ..., z_t)$. To minimize the signature space, this output must be serialized.
   - Let's say $B$ is a byte array. First, $B$ is set to first $2T$ bits of the challenge $e$. A minimum number of zeros is appended to $B$ such that the length of $B$ reaches the nearest byte.
   - $salt$ is appended to the next 32 bytes of $B$.
   - When it comes to appending $(b_t, z_t)$ where $0 \leq t \leq T - 1$, first zeros are padded to the $b_t$ and $z_t$ values that don't use even number of bytes such that the length of $b_t$ and $z_t$ reaches the nearest byte.
   - $b_t$ is appended which is the commitment that uses $l_H$ bytes where $l_H$ is the output length of hash function. The second commitment ($G[t][i]$) is appended to $B$ if UR is applied.
   - $z_i$ is appended. This operation must be in the order mentioned step 5 (first transcript, second the two seed values and third the input share if $e_t \neq 0$)
7. Finally, B is the picnic signature.

The multiplication, addition and Sbox algorithms are given in the Chapter 3.3.

### 4.4 Signature Verification

The signature verification takes $pk$, byte array signature $B$ and byte array $M$ as an input where $M \in [1, 2^{55}]$ is the message signed. Note that when the Unruh transform

is applied, it is shown as UR in the context.

1. First, let's start with the deserialization of $B$.

   - To do this, the first $(2T + 7)/8$ bytes of $B$ are read. Remember all the bit pairs must be in $\{0, 1, 2\}$ and all padding bits must be $0$. These bytes are set to $e$.
   - Next, set $salt$ by writing the next $32$ bytes of $B$.
   - The commitment which is the next $l_H$ bytes of $B$ is set to $b_t$ where $0 \leq t \leq T - 1$ and $l_H$ is the output length of hash function. The second commitment ($G[t][i]$) is read from $B$ if UR is applied. If $e_t = 0$, the length of this commitment is $3rs + n$ bits and $3rs$ bits otherwise.
   - Transcript ($3rs$ bits) is set to the first component of $z_t$.
   - The first seed value ($S$ bits) is appended to $z_t$.
   - The second seed value ($S$ bits) is appended to $z_t$.
   - The input share ($S$ bits) is appended to $z_t$ when $e_t \neq 0$.
   - Finally, deserialization is completed and we have $(e, salt, b_0, ..., b_t, z_0, ..., z_t)$. If any of the above steps fail, reject the signature and return "invalid".

2. For $0 \leq t \leq T - 1$, compute the following operations.

   - First of all, if $e_t = 0$, random tapes $rand[0]$ and $rand[1]$ is computed using the $seed[t][0]$ and $seed[t][1]$ respectively. $view[0].iShare$ and $x[0]$ is assigned to the first $n$ bits of $rand[0]$. Similarly, $view[1].iShare$ and $x[1]$ is assigned to the first $n$ bits of $rand[1]$.
   - If $e_t = 1$, random tapes $rand[0]$ and $rand[1]$ is computed using the $seed[t][1]$ and $seed[t][2]$ respectively. $view[0].iShare$ and $x[0]$ is assigned to the first $n$ bits of $rand[0]$. Similarly, $view[1].iShare$ and $x[1]$ is assigned to the input share in $z_t$.
   - If $e_t = 2$, random tapes $rand[0]$ and $rand[1]$ is computed using the $seed[t][2]$ and $seed[t][0]$ respectively. $view[0].iShare$ and $x[0]$ is assigned to the input share in $z_t$. Similarly, $view[1].iShare$ and $x[1]$ is assigned to the first $n$ bits of $rand[1]$.

- $key$ parameter is computed using key matrix.

$$key = x \cdot Kmatrix[0]$$

- Next, $state$ is computed using the $key$.

$$state = key \oplus p \oplus e_t$$

- For each LowMC round $i$ where $1 \leq i \leq r$, the following parameters are computed respectively.
  - First, the key shares are computed using key matrices.

$$key = x \cdot Kmatrix[i]$$

  - Next, sbox layer is computed using rand and views.

$$state = sbox(state, rand, view[t])$$

  - Affine layer is computed using linear layer matrices.

$$state = state \cdot Lmatrix[i-1]$$

  - Round constant addition is computed using round constants chosen randomly before the signature generation.

$$state = state \oplus roundconstant[i-1] \oplus e_t$$

  - Round key addition is computed using key shares.

$$state = state \oplus key$$

- After computing all LowMC rounds, the output shares are updated using $state$ vectors.

$$views[i].oShare = state[i]$$

  where $0 \leq i \leq 1$.

- Commitments are computed using hash functions.

$$\begin{aligned} C[t][e_t] &= H_0(H_4(seed[0]), view[0]) \\ C[t][(e_t+1) mod 3] &= H_0(H_4(seed[1]), view[1]) \\ C[t][(e_t+2) mod 3] &= c \end{aligned}$$

where $c$ is the commitment, the first element in $b_t$. If UR is applied,

$$\begin{aligned} G[t][e_t] &= G(H_4(seed[0]), view[0]) \\ G[t][(e_t+1) mod 3] &= G(H_4(seed[1]), view[1]) \\ G[t][(e_t+2) mod 3] &= c^{'} \end{aligned}$$

where $c^{'}$ is the commitment, second element in $b_t$.

- The output shares are computed using views.

$$\begin{aligned} outputs[t][e_t] &= views[0].oShare \\ outputs[t][e_t+1] &= views[1].oShare \\ outputs[t][e_t+2] &= (views[0].oShare \oplus views[1].oShare \oplus C) mod 3 \end{aligned}$$

Note that, $C$ is provided in public key $(C, p)$.

3. After computing all $T$ iterations, the challenge $e^{'}$ can be written as

$$\begin{aligned} e^{'} = H_3( & outputs[0][0], outputs[0][1], outputs[0][2], \\ & ... \\ & outputs[T-1][0], outputs[T-1][1]., outputs[T-1][2]., \\ & C[0][0], C[0][1], C[0][2], \\ & ... \\ & C[T-1][0], C[T-1][1], C[T-1][2], \\ & [G[0][0], G[0][1], G[0][2], \\ & ... \\ & G[T-1][0], G[T-1][1], G[T-1][2], ] \\ & salt, pk, M) \end{aligned}$$

Note that "$G[i][j]$" is the commitments that is added when UR is applied and omitted otherwise.

4. If $e = e^{'}$, the signature is accepted. If $e \neq e^{'}$, the signature is rejected.

The multiplication, addition and Sbox algorithms are given in the Chapter 3.3.

# CHAPTER 5

# COMPARISON WITH RELATED WORK

In this section, we give a brief information on other post-quantum digital signature candidates and compare these candidates with Picnic digital signature scheme.

In Table 5.1, public key, private key and signature sizes of the related algorithms are given in bytes, as well as key generation, signature generation and verification time are given in miliseconds. These algorithms are in different categories:

- Hash-based (SPHINCS-256 [14])
- Code-based (FS-Véron [35])
- Lattice-based (TESLA [11], Ring-TESLA [7], BLISS-I [21])
- MQ-based (MQ 5pass [17])
- Supersingular Isogeny (SIDHp751 [36])

As seen in the Table 5.1, Picnic has small public and private key sizes. However, Picnic takes longer to sign a message and verify a signature. The complete efficiency and security analysis of the Picnic algorithm is described in [5].

**Hash-Based Digital Signature Schemes:** The earliest post quantum digital signature scheme is based on hash functions which is called "Lamport-Diffie One-Time Signature Scheme" proposed in 1979 [13]. A key pair consist of secret key and public key can be used only once for a signature with this scheme. We can use Lamport-Diffie one-time signature scheme with Merkle trees to obtain stateful signatures [13]. State counts the one-time signature key pairs used so that these key pairs will not be used

Table 5.1: Comparison with related work

| Scheme | Public key | Private key | SignSize | KeyGen | Sign | Verify |
|---|---|---|---|---|---|---|
| Picnic-L1-FS | 32 | 16 | 34032 | 0.05 | 38.31 | 25.18 |
| Picnic-L1-UR | 32 | 16 | 53961 | 0.04 | 47.93 | 32.36 |
| Picnic2-L1-FS | 32 | 16 | 13802 | 0.04 | 851.85 | 515.93 |
| Picnic-L3-FS | 48 | 24 | 76772 | 0.12 | 128.30 | 84.70 |
| Picnic-L3-UR | 48 | 24 | 121845 | 0.11 | 152.51 | 102.36 |
| Picnic2-L3-FS | 48 | 24 | 29750 | 0.10 | 2830.60 | 1538.25 |
| Picnic-L5-FS | 64 | 32 | 132856 | 0.21 | 308.96 | 204.53 |
| Picnic-L5-UR | 64 | 32 | 209506 | 0.21 | 342.98 | 230.12 |
| Picnic2-L5-FS | 64 | 32 | 54732 | 0.19 | 7080.01 | 3595.40 |
| MQ 5pass | 74 | 32 | 40952 | 0.96 | 7.21 | 5.17 |
| SPHINCS-256 | 1056 | 1088 | 41000 | 0.82 | 13.44 | 0.58 |
| BLISS-I | 7168 | 2048 | 5732 | 44.16 | 0.12 | 0.02 |
| Ring-TESLA | 8192 | 12288 | 1568 | 16k | 0.06 | 0.03 |
| TESLA-768 | 4128k | 3216k | 2336 | 48k | 0.65 | 0.36 |
| FS-Véron | 160 | 32 | 129024 | n/a | n/a | n/a |
| SIDHp751 | 768 | 48 | 141312 | 16.41 | 7.3k | 5.0k |

again. If we keep the tree large and choose the key pair randomly, we can obtain stateless signature schemes such as Sphics [14]. These signature schemes are desirable because their security depends on the collision resistance of the hash function, not hard algorithmic problems. However, they have increased signature sizes. As an example, Sphincs have about 41 kB signatures and 1 kB keys. [16]

**Code-Based Digital Signature Schemes:** The idea in code-based signature schemes is to convert identification scheme to signature scheme by using Fiat-Shamir transform. However, we can get about 129 kB signatures for 128 bit post quantum security. There are also other code-based signature schemes which is proven to be insecure [35]. [16]

**Lattice-Based Digital Signature Schemes:** Some lattice-based signature schemes depends on the worst or average case problems in standard lattices. We can count them as secure, however their key sizes are quite large, around 10 mB. Tesla with

about 1 mB public key size is seem to be the best option with respect to key sizes. There are other lattice-based signature schemes relies on the ring analogues of classical lattice problems. Their security reductions relies on the ideal lattice hardness assumptions such as Bliss. Their key sizes are small. On the negative side of these schemes, ideal lattices are not well-studied enough [11, 7, 21]. [16]

**MQ-Based Digital Signature Schemes:** Their securities rely on the problem of quadratic equation in multivariate system whose security relies on 5-pass identification scheme. They also use the Fiat-Shamir transform. However, their signature sizes are about 40 kB. There are also other multivariate system based signatures that have shorter signatures. However their security is not provable [17]. [16]

**Supersingular Isogeny Signature Schemes:** The idea is to apply the identification scheme and the Unruh transform. However, signature size is about 140 kB for the 128-bit post quantum security. There are also other supersingular isogeny signatures based on conceptually identical constructions which is isogeny based and endomorphism rings. Their signature sizes are relatively small [36]. [16]

# CHAPTER 6

# CONCLUSION

In this thesis, quantum safe digital signatures from symmetric key primitives is studied, mainly Picnic digital signature algorithm. Picnic has the security of symmetric-key primitives(hash functions and block ciphers) that is considered to be secure against quantum attacks. Picnic has small key pairs, parameterizable structure and it does not require hard algebraic problems.

In Picnic signature algorithm, public key($y$) can be derived from secret key($x$) over a secure one way function($f$) as $y = f(x)$. Signature is the secret key's non-interactive zero knowledge proof. For this purpose, zero knowledge proof systems are used. Zero knowledge proof systems uses circuits to compute the protocol. This technique is based on the "Multiparty computation-in-the-head" paradigm to zero-knowledge.

To make the proof non-interactive, Fiat-Shamir(FS) transform and Unruh's(UR) transform can be used. FS transform provides security in random oracle model(ROM) while UR transform provides security in quantum random oracle model(QROM).

# REFERENCES


[1] Cryptographic hash function, wikipedia.org/wiki/Cryptographic_hash_function, accessed: May 2016.

[2] Nist post-quantum cryptography standardization process, csrc.nist.gov/projects/post-quantum-cryptography, accessed: 2019-06-08.

[3] Fips pub 202. sha-3 standard: Permutation-based hash and extendable-output functions, NIST, 2015, nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf.

[4] National institute of standards and technology. submission requirements and evaluation criteria for the post-quantum cryptography standardization process, 2016, csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf.

[5] The picnic signature algorithm, design document version 2,1, 2019, https://github.com/microsoft/Picnic/blob/master/spec/design-v2.1.pdf.

[6] The picnic signature algorithm, specification version 2,1, 2019, github.com/microsoft/Picnic/blob/master/spec/spec-v2.1.pdf.

[7] S. Akleylek, N. Bindel, J. Buchmann, J. Krämer, and G. A. Marson, An efficient lattice-based signature scheme with provably secure instantiation, in D. Pointcheval, A. Nitaj, and T. Rachidi, editors, *Progress in Cryptology – AFRICACRYPT 2016*, pp. 44–60, Springer International Publishing, Cham, 2016, ISBN 978-3-319-31517-1.

[8] M. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner, Ciphers for mpc and fhe, Cryptology ePrint Archive, Report 2016/687, 2016, eprint.iacr.org/2016/687.

[9] M. R. Albrecht, G. V. Bard, and W. Hart, Efficient multiplication of dense matrices over GF(2), CoRR, abs/0811.1714, 2008.

[10] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner, *Ciphers for MPC and FHE. In Advances in Cryptology – EUROCRYPT 2015*, Springer Berlin Heidelberg, 2015.

[11] E. Alkim, N. Bindel, J. Buchmann, O. Dagdelen, E. Eaton, G. Gutoski, J. Krämer, and F. Pawlega, Revisiting tesla in the quantum random oracle model, Cryptology ePrint Archive, Report 2015/755, 2015, eprint.iacr.org/2015/755.

[12] G. V. Bard, *The Method of Four Russians. In Algebraic Cryptanalysis*, Springer Publishing Company, Incorporated, 1st edition, 2009, ISBN 0387887563, 9780387887562.

[13] D. J. Bernstein, J. Buchmann, and E. Dahmen, *Hash-based Digital Signature Schemes. In Post-Quantum Cryptography*, Springer-Verlag Berlin Heidelberg, 2009.

[14] D. J. Bernstein, D. Hopwood, A. Hülsing, T. Lange, R. Niederhagen, L. Papachristodoulou, M. Schneider, P. Schwabe, and Z. Wilcox-O'Hearn, Sphincs: Practical stateless hash-based signatures, in E. Oswald and M. Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pp. 368–397, Springer Berlin Heidelberg, Berlin, Heidelberg, 2015, ISBN 978-3-662-46800-5.

[15] D. Boneh, O. Dagdelen, M. Fischlin, A. Lehmann, C. Schaffner, and M. Zhandry, Random oracles in a quantum world, Springer, Berlin, Heidelberg, 7073, 2011.

[16] M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha, Post-quantum zero-knowledge and signatures from symmetric-key primitives, ACM, 2017.

[17] M.-S. Chen, A. Hülsing, J. Rijneveld, S. Samardjiska, and P. Schwabe, From 5-pass mq-based identification to mq-based signatures, Cryptology ePrint Archive, Report 2016/708, 2016, eprint.iacr.org/2016/708.

[18] J. Daemen and V. Rijmen, *AES Proposal: Rijndael*, 1999.

[19] O. Dagdelen, M. Fischlin, and T. Gagliardoni, The fiat–shamir transformation in a quantum world, Springer, Berlin, Heidelberg, 8270, 2013.

[20] C. D. de Saint Guilhem, L. D. Meyer, E. Orsini, and N. P. Smart, Bbq: Using aes in picnic signatures, Cryptology ePrint Archive, Report 2019/781, 2019, https://eprint.iacr.org/2019/781.

[21] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky, Lattice signatures and bimodal gaussians, Cryptology ePrint Archive, Report 2013/383, 2013, https://eprint.iacr.org/2013/383.

[22] A. Fiat and A. Shamir, *How To Prove Yourself: Practical Solutions to Identification and Signature Problems*, Springer Berlin Heidelberg, 1987.

[23] S. D. Galbraith, *Mathematics of Public Key Cryptography*, Cambridge University Press, 2012.

[24] S. D. Galbraith, *The RSA and Rabin cryptosystems. In Mathematics of Public Key Cryptography*, Cambridge University Press, 2012.

[25] I. Giacomelli, J. Madsen, and C. Orlandi, Zkboo: Faster zero-knowledge for boolean circuits, Cryptology ePrint Archive, Report, 2016/163, 2016.

[26] Y. Ishai, R. Ostrovsky, E. Kushilevitz, and A. Sahai, *Zero-knowledge from secure multiparty computation. In Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computingy*, ACM Press, 2007.

[27] Y. Ishai, R. Ostrovsky, E. Kushilevitz, and A. Sahai, *Zero-knowledge from secure multiparty computation*, volume 39,3, SIAM Journal on Computing, 2009.

[28] J. Katz, V. Kolesnikov, and X. Wang, Improved non-interactive zero knowledge with applications to post-quantum signatures, in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pp. 525–537, ACM, New York, NY, USA, 2018, ISBN 978-1-4503-5693-0.

[29] M. Lauridsen, *Design and Analysis of Symmetric Primitives*, Ph.D. thesis, 2016.

[30] D. Luciano and G. Prichett, Cryptology: From caesar ciphers to public-key cryptosystems, The College Mathematics Journal, 18(1), pp. 2–17, 1987.

[31] C. Paar and J. Pelzl, *Understanding cryptography: a textbook for students and practitioners*, Spinger Science and Business Media, 2009.

[32] J. L. G. Pardo and C. Gómez-Rodríguez, *The SHA-3 Family of Cryptographic Hash Functions and Extendable-Output Functions*, Maple, 2015.

[33] P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM Journal on Scientific Computing, 26, 1996.

[34] S. Singh, *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography*, Anchor, 2000.

[35] P. Véron, Improved identification schemes based on error-correcting codes, Applicable Algebra in Engineering, Communication and Computing, 8(1), pp. 57–69, Jan 1997, ISSN 1432-0622.

[36] Y. Yoo, R. Azarderakhsh, A. Jalali, D. Jao, and V. Soukharev, A post-quantum digital signature scheme based on supersingular isogenies, in A. Kiayias, editor, *Financial Cryptography and Data Security*, pp. 163–181, Springer International Publishing, Cham, 2017, ISBN 978-3-319-70972-7.