Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Sahana Sridhar

A Survey of Quantum-safe Digital Signatures and their building blocks

Master's thesis in Communication Technology
Supervisor: Prof. Danilo Gligoroski
Co-supervisor: Mattia Veroni
March 2021

NTNU
Norwegian University of
Science and Technology

Sahana Sridhar

A Survey of Quantum-safe Digital Signatures and their building blocks

Master's thesis in Communication Technology
Supervisor: Prof. Danilo Gligoroski
Co-supervisor: Mattia Veroni
March 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

Title: A Survey of Quantum-safe Digital Signatures and their building blocks
Student: Sahana Sridhar

Problem description

The arrival of small-scale quantum computers has heightened the threat to existing internet security systems, specifically, the Public Key Infrastructure (PKI). The standards bodies such as the National Institute of Standards and Technology (NIST) are calling for quantum-resistant cryptosystems that can be considered for long-term replacement of current public-key cryptosystems. Digital signatures have been for decades, a very important part of critical sectors like governance and banking and common sectors like education, corporate organizations and so on. With the increase in usage of remote devices and Internet of Things (IoT) in these sectors, it would be prudent to consider the recent developments in communication networks and security, as it is becoming a growing concern with more and more applications moving to the cloud.

In this work, we aim to identify loopholes and mount possible attacks on current state-of-the-art post-quantum digital signature schemes, such as the recently designed Picnic. Picnic is based on symmetric key primitives and has been shown to be vulnerable to multi-target attacks. Moreover, various optimizations have been made to the LowMC block cipher implementation about its linear layers. Some of the other symmetric key candidates are the hash-based SPHINCS scheme, the 5-pass or 3-pass identification schemes based on multivariate quadratic equations, the lattice-based TESLA scheme. A detailed and comparative study of the available schemes and related attacks will give a comprehensive understanding of the post-quantum algorithms designed to withstand the quantum threat.

Note: The title of the thesis is updated based on feedback received during the presentation of the results at the faculty. The problem description and the scope remain unchanged.

Date approved: 2020-02-21
Supervisor: Prof. Danilo Gligoroski, IIK, NTNU
Co-supervisor: Mattia Veroni, IIK, NTNU

Abstract

Security and privacy in the internet as we know today, is under threat from the emerging technology of building larger and commercial quantum computers. Even though the quantum computers built are relatively small, and not very effective to disrupt the Public Key Infrastructure, nevertheless, it is important to recognize the imminent threat. Besides, the famous quantum algorithms due to Shor, Grover and Brassard-Høyer-Tapp, have shaken the classical foundations of public-key cryptography. These reasons have triggered NIST to spearhead the Post-Quantum Cryptography (PQC) Standardization project, to enable quality development and faster deployment of quantum-safe cryptosystems. This is currently in the third round with 7 finalists and 8 alternative candidates - both key-exchange mechanisms and digital signature schemes.

To understand the future of public-key cryptography in the post-quantum world, the natural approach would be to explore different mathematical problems that cannot be solved by quantum computers, yet. Built on this premise, this thesis presents a theoretical study of cryptographic hard problems that are allegedly quantum-resistant, and the NIST round 3 PQC signature schemes constructed from them. Additionally, in order to understand the practical implications of these schemes, a holistic evaluation in terms of security assurance and performance has been done as part of this work. Open source tools, such as the Open Quantum Safe (OQS) project and PQClean, have been used in conjunction with Docker containers to evaluate these schemes quantitatively.

To address the first part of the above research questions, the categories of quantum-resistant cryptographic systems have been studied, namely, *(i) Lattice-based (ii) Multivariate quadratic (MQ) polynomials based (iii) Hash-based (iv) Code-based and (v) Isogeny-based.* In the second part, the six signature schemes have been studied, which are (a) Hash/Symmetric-primitive based - *Picnic* and *Sphincs+* (b) MQ polynomials based - *GeMSS* and *Rainbow* (c) Lattice-based - *Crystals-Dilithium* and *Falcon*.

Initially inspired by the simple construction of Picnic, the 3 alternate candidates (i.e. Picnic, Sphincs+ and GeMSS) in NIST Round III have been studied in more detail compared to the finalists (i.e. Dilithium, Falcon and Rainbow) as part of this work. Later, this decision was also guided by the progression in the NIST PQC standardization, and the curiosity about why Picnic and the two other schemes did not make it as

finalists. Along with the cryptographic constructions, the robustness of the schemes through their proofs of security, and their vulnerabilities in terms of different attacks have also been discussed.

Furthermore, during the quantitative analysis of the schemes, it was realized that GeMSS is not currently in the Liboqs library of OQS. Therefore, integrating GeMSS into Liboqs, to allow for evaluation of all six schemes on a common platform and in cryptographic protocols such as TLS became one of the objectives of this work. However, due to a recent serious key recovery attack [Din20] on the underlying problem of GeMSS, this task has been deemed irrelevant for now. The partial implementation done upto the point of knowing about the attack has been presented. Nevertheless, a comprehensive comparison of all six schemes has been done taking into consideration, the software and hardware implementation complexities of these schemes.

The observation from the quantitative analysis of the schemes has been that, in software implementations, Dilithium is by far the best algorithm in terms of performance and sizes of the components (keys and signatures). However, Rainbow seems to have very good performance in high-speed hardware implementations. In this regard, despite being constructed from simpler cryptographic primitives, Picnic and Sphincs+ have average performances compared to other schemes in either platforms. As for the qualitative analysis, each scheme is vulnerable to various attacks, and the strength and security of a scheme is determined by its ability to counter powerful attacks. Though symmetric-based schemes are generally believed to be quantum-resistant, needing only to double their parameters to be secure, they are still vulnerable to multi-target attacks. The MQ-polynomial based schemes on the other hand, are subject to attacks that exploit their mathematical constructions. While the lattice-based schemes are robust in terms of construction, their implementations can be complicated and subject to generic fault and side-channel attacks like any other scheme.

Sammendrag

Sikkerhet og personvern på internett som vi kjenner i dag, er truet av den nye teknologien for å bygge større og kommersielle kvantecomputere. Selv om kvantecomputere som er bygget er relativt små, og ikke veldig effektive for å forstyrre Public Key Infrastructure, er det likevel viktig å gjenkjenne den forestående trusselen. Dessuten har de berømte kvantealgoritmene på grunn av Shor, Grover og Brassard-Høyer-Tapp, rystet de klassiske grunnlagene for kryptografi med offentlig nøkkel. Disse årsakene har utløst NIST til å stå i spissen for PQC Standardiseringsprosjektet, for å muliggjøre kvalitetsutvikling og raskere distribusjon av kvantesikre kryptosystemer. Dette er for tiden i tredje runde med 7 finalister og 8 alternative kandidater - både nøkkelutvekslingsmekanismer og ordninger for digital signatur.

For å forstå fremtiden for offentlig nøkkelkryptografi i post-quantumverdenen, ville den naturlige tilnærmingen være å utforske forskjellige matematiske problemer som ennå ikke kan løses av kvantecomputere. Bygget på denne forutsetningen presenterer denne avhandlingen en teoretisk studie av kryptografiske harde problemer som angivelig er kvantebestandige, og NIST runde 3 PQC signaturskjemaer konstruert av dem. For å forstå de praktiske implikasjonene av disse ordningene, er det i tillegg gjort en helhetlig evaluering av sikkerhetssikring og ytelse som en del av dette arbeidet. Open source-verktøy, som Open Quantum Safe (OQS) -prosjektet og PQClean, har blitt brukt i forbindelse med Dockercontainere for å evaluere disse ordningene kvantitativt.

For å adressere den første delen av de ovennevnte forskningsspørsmålene, har kategoriene av kvantebestandige kryptografiske systemer blitt studert, nemlig *(i) Latticebasert (ii) Multivariate kvadratiske (MQ) polynomer basert (iii) Hash-basert (iv) Kodebasert og (v) Isogeny-basert.* I den andre delen har de seks signaturskjemaene blitt studert, som er (a) Hash / Symmetric-primitive based - *Picnic* og *Sphincs +* (b) MQ polynomials based - *GeMSS* og *Rainbow* (c) Latticebasert - *Crystals-Dilithium* og *Falcon.*

Opprinnelig inspirert av den enkle konstruksjonen av Picnic, har de 3 alternative kandidatene (dvs. Picnic, Sphincs + og GeMSS) i NIST Round III blitt studert mer detaljert sammenlignet med finalistene (dvs. Dilithium, Falcon og Rainbow) som en del av dette arbeidet. Senere ble denne avgjørelsen også styrt av utviklingen i NIST PQC-standardiseringen, og nysgjerrigheten rundt hvorfor Picnic og de to andre ordningene ikke

gjorde det som finalister. I tillegg til de kryptografiske konstruksjonene, har robustheten til ordningene gjennom deres sikkerhetsbevis og deres sårbarhet i forhold til forskjellige angrep også blitt diskutert.

Videre ble det under den kvantitative analysen av ordningene innsett at GeMSS for øyeblikket ikke er i Liboqs-biblioteket til OQS. Derfor ble integrering av GeMSS i Liboqs, for å tillate evaluering av alle seks ordningene på en felles plattform og i kryptografiske protokoller som TLS, et av målene med dette arbeidet. På grunn av et nylig alvorlig nøkkelgjenopprettingsangrep [Din20] på det underliggende problemet med GeMSS, har denne oppgaven blitt ansett som irrelevant for nå. Den delvise implementeringen som er gjort til det punktet å vite om angrepet, har blitt presentert. Likevel er det gjort en omfattende sammenligning av alle de seks ordningene under hensyntagen til programvaren og maskinvarens implementeringskompleksitet i disse ordningene.

Observasjonen fra den kvantitative analysen av skjemaene har vært at Dilithium i programvareimplementeringer er den klart beste algoritmen når det gjelder ytelse og størrelse på komponentene (nøkler og signaturer). Imidlertid ser Rainbow ut til å ha veldig god ytelse i høyhastighets maskinvareimplementeringer. I denne forbindelse, til tross for at de er konstruert av enklere kryptografiske primitiver, har Picnic og Sphincs + gjennomsnittlig ytelse sammenlignet med andre ordninger i begge plattformene. Når det gjelder den kvalitative analysen, er hver ordning sårbar for forskjellige angrep, og styrken og sikkerheten til en ordning bestemmes av dens evne til å motvirke kraftige angrep. Selv om symmetrisk baserte ordninger generelt antas å være kvantebestandige, og bare trenger å doble parametrene for å være sikre, er de fremdeles sårbare for angrep med flere mål. MQ-polynombaserte ordninger er derimot utsatt for angrep som utnytter deres matematiske konstruksjoner. Mens de latticebaserte ordningene er robuste når det gjelder konstruksjon, kan implementeringene være kompliserte og utsatt for generiske feil og sidekanalangrep som alle andre ordninger.

Acknowledgement

This Master Thesis has been an incredible journey for me personally and professionally, with a lot of ups and downs but nevertheless, a very much fun-filled and exciting roller-coaster ride. It has changed me as a person in many ways, thankfully all for the good. I would like to acknowledge and thank all the people who have stood by me as pillars of strength during this wonderful journey.

First of all, I would like to express my sincere gratitude to my Supervisor, Prof. Danilo Gligoroski, for his constant encouragement, guidance and support, especially during crucial times. I thank him for believing in me and my capabilities, and giving me the academic freedom to explore higher and beyond.

I would like to wholeheartedly thank my Co-Supervisor, Mattia Veroni, for always being my support system and guiding me at every small step and big obstacles, and helping me to sail through the tough times. I also sincerely thank him for his invaluable, timely and critical comments on the Thesis draft versions, which have brought it to the present form.

I would also like to extend my sincere gratitude to the Department of Information Security and Communication Technology (IIK) at NTNU, especially Aud Bakken and the Study Faculty, for guiding and providing necessary support during times of need.

I would like to specially mention and wholeheartedly thank Line Sivertsen, for patiently listening to all my rantings and helping keep myself sane and focused during this entire period. I am very much indebted to her for all the help and support.

I would like to thank all my friends in Trondheim - Nayan, Paritosh, Divya, Anirudh, Tejali, Swapnil and Jeevith - for being my family away from home here. Particularly, I would like to thank Paritosh, Nayan and my other friends outside Norway - Sagar, Supreeth, Pradyumna and Donna - for their unconditional support and understanding in times of need. All of them have always made me smile and taught me to take things easily.

I would also like to thank my mother, my sister-in-law and my brother Sham Prasad, for their eternal belief in me to achieve my dreams, and wanting the best of things in the world for me to attain and enjoy the

fruits. I am also thankful to my parents-in-law for their belief and constant support for all my endeavours. It would be incomplete without explicitly mentioning my gratitude to my better half, my support system, my backbone - my husband Rohith - for being the way he is and constantly striving to make me a stronger, happier individual each day to the next.

Last but not the least, I would like to say a huge thanks to this beautiful life, which is so filled with different, vibrant colours and I wish to continuously explore and be awed by it.

Contents

List of Figures

List of Tables

List of Algorithms

Listings

Chapter 1
Introduction

Communication has evolved over time from the physical methods to analog communication through wired devices up to the digital age of computers. Along with the progress in telecommunication, the challenges in protecting the information from untrusted parties also became increasingly sophisticated. To address these challenges, mathematical problems were used to develop techniques that protected the leakage of information, giving rise to the field of *cryptography*.

In this regard, though *classical* digital computers were able to solve most of the computational problems in mathematics, there were some categories of problems that could not be solved efficiently (i.e. in terms of the time and resources required). It was this set of mathematical problems that classical cryptography relied on, to safeguard information and the identities of the communicating parties. However, the advances in the field of *quantum* computing have brought in newer but exciting challenges to the world of communication and cryptography. Thus, protecting individual and collective privacy and integrity, notwithstanding the quantum threat is the goal of the cryptographic community for the next decade.

Quantum computing is an entirely new way of computation that is different from the traditionally known binary classical computing. It uses the principles of quantum mechanics to perform efficient computations, even at large sizes, impossible by largest/fastest available classical computers. Quantum mechanics deals with the duality of matter i.e. wave-like nature and particle-like nature. Based on these two fundamental principles, the main concepts of quantum computing "superposition" and "entanglement" are exploited to develop exponentially faster computing algorithms that solve many classical problems [Qis20b].

While classical computers operate on *bits* or binary states of 0 and 1, quantum computers operate on the *quantum bit* or *qubit*, the quantum state which represents both 0 and 1 or linear combinations of 0 and 1. This linear combination is called *superposition*, which allows for the simultaneous processing of exponentially many

logical states. The other concept *entanglement* refers to the combined state of multiple qubits, which contain more information than that contained in each individual qubit. Such multi-qubits that are well entangled, are also known to be useful for generating high-entropy pseudo-random strings [Qis20b].

All quantum algorithms are implemented on *quantum circuits*. A quantum circuit consists of *quantum gates* that are constructed differently than classical logic gates, but do perform similar logical operations as their classical counterparts. They also have a set of instructions and control logic as classical logic gates, but operate on a superposition of quantum states in the input and output *wires*. For the circuit evaluation, a superposition of all possible quantum states is created and given as input to the quantum circuit. Then, according to the defined instructions, the quantum circuit causes selective interference of the amplitudes and phases of the input states, giving the solution of the computation which can be measured at the output [Qis20b]. The final measurement of the result collapses all the information contained in the superposed output of the quantum circuit to a single classical output state.

IBM being one of the leaders in building larger and faster quantum computers, has also developed ***QisKit*** [AAB+19], an open source software development kit (SDK) based on python for quantum computing. It allows users to design quantum circuits, run optimized quantum algorithms on classical simulators, create high-level models of problems and also run optimized algorithms on real and different types of quantum hardware. It also provides support and has code packages for applications in finance, physical chemistry and machine learning algorithms.

Having had a glimpse of what quantum computing means and how it works at a very high level, it would be interesting to look at some of the quantum algorithms developed in the 1990's, and gain a deeper understanding of the problems that they can solve efficiently and to see why they are considered a threat to classical cryptography.

Shor's Algorithm

The most famous quantum algorithm that shook the foundations of classical (i.e. *public-key*) cryptography was Peter Shor's [Sho94] *Quantum Factoring* algorithm, developed in 1994. Shor showed that quantum computers can very effectively break integer factorization and discrete log problems in polynomial time unlike classical computers. The algorithm run time is $\mathcal{O}((\log N)^2(\log\log N)(\log\log\log N))$, where N is the number of input bits, against $\mathcal{O}(\exp[c\ (\log N)^{1/3}(\log\log N)^{2/3}])$ of best known classical algorithms [Sho99]. This *Period Finding* algorithm of Shor is briefly explained below.

Period Finding Problem: ([Qis20c]) Given a periodic function $f(x) = a^x$

(mod N), where $0 < a < N$ and $gcd(a, N) = 1$, find the period r such that $a^r \equiv 1 \pmod{N}$. The period r is the smallest non-zero integer that satisfies the above relation. Using the *order* r, and the relation $a^r - 1 = 0 \pmod{N}$, the $gcd(a^{r/2} \pm 1, N)$ gives one of the factors of N.

The basic idea of Shor's algorithm is to use *quantum phase estimation* to solve the period finding problem. The quantum phase estimation is used on the unitary operator, given by,

$$U|y\rangle \equiv |ay \mod N\rangle \implies U|1\rangle = |a\rangle \rightarrow \cdots \rightarrow U^r|1\rangle = |1\rangle$$

where, every successive application of U, will multiply the state by $a \pmod{N}$, giving the state of $|1\rangle$ again, after r iterations, which gives the required period.

Grover's Algorithm

The other famous quantum algorithm is the equivalent of the classical brute force attack or exhaustive search, developed by Lov Grover [Gro96] in 1996. Although it was initially designed to be an algorithm for efficiently searching databases, it has found many other applications, the significant one being the *Quantum Exhaustive Search* problem, briefly explained below. However, this algorithm only gives a quadratic speed-up over similar classical algorithms, and does not solve NP-Complete problems in polynomial time. The algorithm run time is $\mathcal{O}(\sqrt{N})$ where N is the number of input bits, as opposed to $\mathcal{O}(N)$ for classical algorithms [Aar05].

Exhaustive Search Problem: ([Qis20a]) Given a function $f : K \rightarrow \{0, 1\}$ where

$$f(k) = \begin{cases} 1 & \text{, if } k = k_0 \\ 0 & \text{, otherwise} \end{cases}; \qquad \text{find } k_0 \text{ such that } f(k_0) = 1.$$

The core idea of Grover's algorithm is based on the concept of *amplitude amplification*. The algorithm solves the search problem by converting it into an oracle that adds negative phase to the marked solution in the system state.

$$U_\omega|x\rangle = \begin{cases} |x\rangle & \text{, if } x \neq \omega \\ -|x\rangle & \text{, if } x = \omega \end{cases}$$

Given a function f as below, an oracle $\mathcal{O}$ for f can be described as

$$f(x) = \begin{cases} 0 & \text{, if } x \neq \omega \\ 1 & \text{, if } x = \omega \end{cases}; \qquad \mathcal{O} \leftarrow U_\omega|x\rangle = (-1)^{f(x)}|x\rangle$$

thus, enabling easy identification of the search value. This can be used in applications where we need to find the input value for an output obtained as the result of evaluation of some function f.

Brassard-Høyer-Tapp (BHT) Algorithm

Another important quantum algorithm is the Brassard-Høyer-Tapp (BHT) algorithm [BHT97], developed in 1997 and named after its inventors. This algorithm uses the classical *birthday paradox* and the quantum search algorithm by Grover to achieve the combined cubic-root speed-up against equivalent classical algorithms. The *collision finding* problem is briefly explained below:

Collision Problem: ([BHT97]) Given a *many-to-one* function $f : X \to Y$ where $y = f(x) \in Y$, find $x_0, x_1 \in X$ such that $f(x_0) = f(x_1)$ and $x_0 \neq x_1$.

In general, the above collision finding problem implies that an *r-to-1* function has distinct r pre-images for every image i.e., for every y_i there exist r distinct x_j. Given that this function is modeled as a black-box, the BHT algorithm needs only $\mathcal{O}(\sqrt[3]{N/r})$ expected evaluations of the function. This is faster than the best known classical algorithms, which need $\mathcal{O}(\sqrt{N})$ evaluations [BHT97].

Research efforts on quantum algorithms have mostly focused on solving the so-called *Hidden Subgroup Problem (HSP)*, to which most other problems can be reduced to. For instance, problems based on (a) finite or countable *groups* - like factoring and discrete-log, and (b) uncountable groups - like Pell's equation, reduce to the HSP for *abelian* groups, and therefore, have been efficiently solved by quantum algorithms. On the other hand, for problems based on *non-abelian* groups, like graph isomorphism and unique shortest lattice vector problem, there are no known efficient quantum algorithms to solve them [HV09]. This is the vital point that post-quantum cryptography seeks to take advantage of, in order to construct cryptosystems that are resistant to attacks by quantum algorithms. In this regard, the following section outlines the current efforts taken by the cryptographic community towards moving in the direction of quantum-safe cryptography.

1.1 Scope

The National Institute of Standards and Technology (NIST) [1] is a standards organization, with its headquarters located in Maryland, USA. It is a non-regulatory agency under the Department of Commerce of the US Government [NIS21]. An agency founded in 1901 and mainly meant to be a physical science laboratory has progressed over the years, today expanding its scope from physical sciences to information technology and beyond. The main focus areas of NIST are: "*measurement sciences, rigorous traceability, and development and use of standards*". Though NIST establishes standards for technology and measurements to be adopted across USA, it is also widely accepted internationally.

[1] originally called the National Bureau of Standards (NBS)

As part of its standardization efforts, NIST has initiated many projects over the years, to standardize information security products and services. These standards are published through a set of documents, such as, Federal Information Processing Standard(s) (FIPS), Special Publication(s) (SP), and NIST Interagency or Internal Report(s) (NIST-IR). The Information Technology Laboratory (ITL) under NIST is the body responsible for developing cryptography standards. Some of these projects to standardize different areas of cryptography are [Moo19]:

- Block-Ciphers — Advanced Encryption Standard (AES)
  15 candidates, 2 rounds, 5 finalists (1997 - 2000)
- Hash Functions — Secure Hash Algorithm 3 (SHA-3)
  14 candidates, 2 rounds, 5 finalists (2007 - 2012)
- Post-Quantum Cryptography (PQC) — General purpose quantum-resistant cryptosystems
  15 candidates, 3 rounds, 7 finalists and 8 alternate candidates
  (2016 - Present – 3^{rd} round ongoing)
- Lightweight Cryptography (LWC) — Light-weight cryptosystems
  32 candidates, 2 rounds
  (2018 - Present – 2^{nd} round ongoing)

1.1.1 PQC standardization

As mentioned above, NIST started the PQC standardization process with a formal call for proposals, in December 2016. The process focuses on standardizing *Key Exchange Mechanism (KEM)* or *Public Key Encryption (PKE)* schemes and *Digital Signature scheme (SIG)*. NIST received around 82 submissions from around the world, for all three categories of schemes, out of which 69 candidates (49 PKE/KEM and 20 SIG schemes) were accepted as they met the submission requirements.

Round 1

The criteria for minimum acceptance and evaluation in Round 1 were [AASA+19]:

1. *Reference and Optimized C/C++ implementations*
2. *Known-Answer Tests (KATs)*
3. *Algorithms to be implementable on a wide variety of hardware and software platforms*

4. *Written specification*
5. *Required Intellectual Property statements*

The shortlisted Round 1 candidates according to the post-quantum family of mathematical assumptions/constructions are given in Table 1.1. Out of the 69 shortlisted candidates from round I, five of the schemes were withdrawn from the competition and/or merged with other schemes submitted to the standardization process.

Round 1 Selected Candidates				
PQ family	**PKE/KEM**	**SIG**	**PKE/KEM & SIG**	**Total**
Lattices	21	5	–	26
Multi-variate	2	7	DME SRTPI*	11
Code-based	17 + 1*	2 + 1*	–	21
Hash/Symmetric-based	–	Gravity-Sphincs Sphincs+ Picnic	–	3
Isogeny-based	SIKE	–	–	1
Other	2 + 2*	WalnutDSA	pqRSA KEM pqRSA SIG	7

Table 1.1: Round I Selected Candidates (the starred (*) algorithms were withdrawn from the process and/or merged with other schemes)

Round 2

The evaluation criteria for the second round were [AASA+20]:

1. ***Security***
   – *PKE/KEM: Semantic security w.r.t adaptive chosen ciphertext attack ≡ IND-CCA2 and Semantic security w.r.t chosen plaintext attack ≡ IND-CPA*
   – *SIG: Existential Unforgeability w.r.t adaptive chosen message attack ≡ EUF-CMA*
   – *Classification of proposed parameter sets w.r.t NIST's security levels*
   – *Perfect Forward Secrecy, resistance to side-channel and multi-key attacks, and resistance to misuse*

2. ***Cost and Performance***
   – *Computational efficiency of operations, transmission costs of artifacts, and implementation costs in terms of RAM or gate counts*

3. ***Algorithm and Implementation Characteristics***
   – *Constant-time implementations, protection against power analysis attacks, performance in internet protocols, and simple, elegant and royalty-free designs for wide-spread adoption*

The Round 2 candidates are shown in Table 1.2. From Round 1, 26 candidates were selected to progress into Round 2, with 17 PKE/KEM schemes and 9 SIG schemes. A few algorithms submitted as separate schemes in round 1, were merged and re-submitted as a single scheme for round 2.

Round 2 Selected Candidates				
PQ family	**PKE/KEM**	**SIG**	**PKE/KEM & SIG**	**Total**
Lattices	9†	3	–	12
Multi-variate	–	4	–	4
Code-based	7†	–	–	7
Hash/Symmetric-based	–	Sphincs+ Picnic	–	2
Isogeny-based	SIKE	–	–	1

Table 1.2: Round 2 Selected Candidates († $\implies$ 2 or more schemes from the PQ family were merged into one scheme)

Round 3

The Round 3 finalists and alternate candidates were announced by NIST in July 2020, which are listed in Table 1.3:

Round 3 Finalists		
PQ Family	**PKE/KEM**	**SIG**
Lattices	Crystals-Kyber NTRU Saber	Crystals-Dilithium Falcon
Multi-variate	–	Rainbow
Code-based	Classic McEliece	–

Round 3 Alternate Candidates		
PQ Family	**PKE/KEM**	**SIG**
Lattices	FrodoKEM NTRU Prime	–
Multi-variate	–	GeMSS
Code-based	BIKE HQC	–
Hash/Symmetric-based	–	Sphincs+ Picnic
Isogeny-based	SIKE	–

Table 1.3: Round 3 Finalists and Alternatives

The alternate candidates have been announced to keep diverse constructions in the standardization process and possibly, NIST will declare more than one scheme as the standard for a variety of applications, such as general-purpose encryption and signature schemes, for ephemeral (i.e. one-time) use cases, and so on.

1.2 Motivation

Digital communication today happens over the internet using classical computing devices. However, the improved knowledge on how to build larger and commercially viable quantum computers has threatened the security and privacy of information as we know today on the internet. Although large enough quantum computers have not yet been built that can break, for instance a 3072-bit RSA modulus, the threat is real and impending. Thus, recognizing this threat, NIST has started efforts to standardize cryptosystems that are claimed to be quantum-resistant. Even so, there are pertinent challenges to standardizing and adopting post-quantum cryptosystems, i.e. *(i) efficiency (ii) confidence* and *(iii) usability.* While classical cryptosystems have stood the test of time and overcome these challenges to a significant extent, the same evaluation needs to be done for the post-quantum cryptosystems before they can be deployed and widely used in the internet.

Thus, it is important to first understand what "*post-quantum cryptosystems*" are and how are they able to resist quantum attacks. Based on this understanding, it becomes much easier to gain more confidence in their security, evaluate how efficient they are in terms of performance and memory usage, and also to ascertain their usability for different applications. Therefore, this forms the motivation for the study and survey of quantum-safe cryptosystems in this thesis. Based on this, the below research questions have been formulated along with a brief explanation of the specific motivation behind each question.

Research Question 1:
What are the different cryptographic hard problems that are believed to be resistant to quantum attacks?

As stated previously, there are certain sets of hard mathematical problems that are allegedly quantum-resistant and many cryptosystems have been constructed based on these problems. Therefore, a study of these mathematical problems and the different baskets into which they are categorized forms the first objective of this thesis. In this regard, Chapter 2 addresses the same and also contains the formal definition of a digital signature, which lays the foundation for all further discussions in this thesis.

Research Question 2:
Which are the signature candidates considered for the NIST standardization so far? What are the different quantum-safe constructions they are based on? Are these schemes secure enough?

Although NIST is standardizing both Key Exchange Mechanisms (KEMs) and Digital Signature schemes (SIGs), the community has mainly focused on the evaluation of KEMs for deployment considerations in the *Public Key Infrastructure (PKI)*. Several evaluations have been made across literature in different settings and environments for the PKE/KEM schemes [DFA+20]. However, there has not been much evaluation of the SIG schemes as it is not considered to be an immediate need compared to the PKE/KEM schemes [SKD20]. Therefore, this thesis work focuses on the study and evaluation of signature schemes over PKE/KEM schemes. This question also becomes relevant in terms of fixing the scope of the current work.

Addressing this research question, the chapters 3 through 6 contain a detailed description of the six signature schemes that have progressed to the third round of the NIST PQC standardization, along with a discussion of their security and related attacks.

Research Question 3:
How do these schemes fare against each other qualitatively and quantitatively? What are the performance metrics of these schemes on different platforms? How well do these signatures fit-in into cryptographic protocols?

As a consequence of studying the mathematical constructions and security of the signature schemes as part of RQ2, the study naturally leads to the next step, that is, a qualitative and quantitative comparison of the schemes. A quantitative comparison includes performance evaluations across different platforms (software and hardware), and evaluation in networking and cryptographic protocols. For instance, GeMSS

was not included in the open source test-bed platform like OQS for evaluation in cryptographic protocols and therefore, integrating GeMSS into OQS for a more complete evaluation was also one of the objectives of this thesis. Chapters 7 and 8 contain the methodology followed by the performance results and a comprehensive discussion of all the six schemes.

Chapter 2
Background

Cryptography is a branch of mathematics and computer science, that deals with protecting data and communication between two or more parties without allowing any form of misuse or violation of privacy. Historically, the development of cryptography has been very interesting, and it mainly focused on *encryption*, that is protecting sensitive information from unwanted third parties or *eavesdroppers*. Today the definition has expanded to include many other applications such as *digital signatures*, *identity authentication*, *interactive proofs*, and so on.

Ancient cryptography mainly focused on traditional characters like letters and numbers, while the advent of computers allowed to encrypt any form of data that is representable in binary format. Some of the early forms of encryption include *Substitution* ciphers, *Transposition* ciphers, *Polyalphabetic* ciphers and so on. The famous German *Enigma* machines were very sophisticated rotor machines developed and used extensively for encrypting confidential information, during World War II. It is also well known that it took years of combined effort from mathematicians in the UK to break the *Enigma* cipher.

Early forms of cryptography were mostly based on a *symmetric* approach. In this approach, the sender and receiver both share a mutually-agreed secret key prior to actual communication, and the same secret key is used for both encryption and decryption. Hence, it becomes mandatory for the involved parties to have known each other and trust each other, to agree on an encryption scheme and a corresponding secret key.

However, this is not always possible, and there are many applications where two unknown parties who cannot trust each other, will need to communicate without a shared secret key. For instance, a bank is a trusted organization with which individuals need to communicate securely. It would be impractical and inefficient if the bank had to establish and maintain individual shared secret keys with each customer. This drawback of the symmetric approach was the motivation for Diffie-Hellman [DH76]

to propose the idea of *Asymmetric* or *Public-key* cryptography. The Diffie-Hellman key exchange protocol was a significant breakthrough in cryptography, as it very efficiently solved the biggest problem of secure key generation and management present in symmetric cryptography.

In the asymmetric approach, each party involved in the communication generates two related keys - *public key* and *private key* - using mathematical problems. As the names suggest, one of the keys is made known publicly while the other one is kept secret. Although they are related, it is *computationally infeasible* to obtain the secret private key only from the public key. In this method, the sender takes the public key of the receiver and applies his secret key to it, in order to generate a shared secret key known only to him and the receiver. Similarly, the receiver applies his secret key to the sender's public key, in order to generate the same shared secret key as the sender. Now both communicating parties have obtained the same secret key value, despite the use of an insecure channel. Using the newly computed shared secret key, the parties can now communicate using the symmetric method.

Symmetric and asymmetric approaches in cryptography have their own pros and cons. Symmetric cryptography is faster due to simpler operations and shorter key lengths, but have poor key management due to the need of having distinct secret keys between every pair of communicating parties. Asymmetric cryptography solves the key management problem efficiently but since it is based on computationally expensive mathematical operations, it is slower than symmetric cryptosystems. Therefore, a hybrid approach is often used, wherein the message itself is encrypted using symmetric ciphers, while the secret key for the symmetric ciphers is generated and encrypted using the asymmetric ciphers.

The CIA Triad is a well-known model for security design principles, which stands for *Confidentiality*, *Integrity* and *Availability*. (a) Confidentiality aims to protect sensitive data from unwanted parties, (b) Integrity provides assurance that the data has not been tampered with, and (c) Availability provides a guarantee that the information is accessible when required by authorized individuals. In general, the cryptosystems which offer confidentiality in the symmetric setting are called encryption schemes (ENC), and the ones offering data integrity are called Message Authentication Codes (MACs). Similarly, their asymmetric counterparts are called PKE schemes for confidentiality and SIGs for data integrity. An overview of these schemes using the traditional sender (*Alice*) and receiver (*Bob*) are shown in Figure 2.1 and Figure 2.2.

Symmetric and Asymmetric Cryptosystems

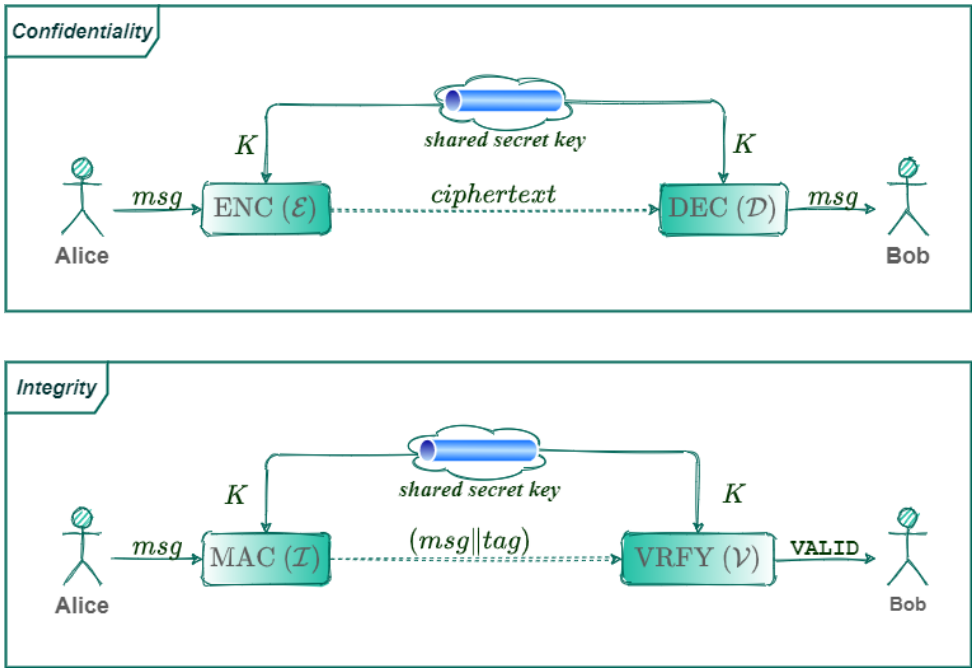


Figure 2.1: Symmetric Cryptosystems with confidentiality and integrity

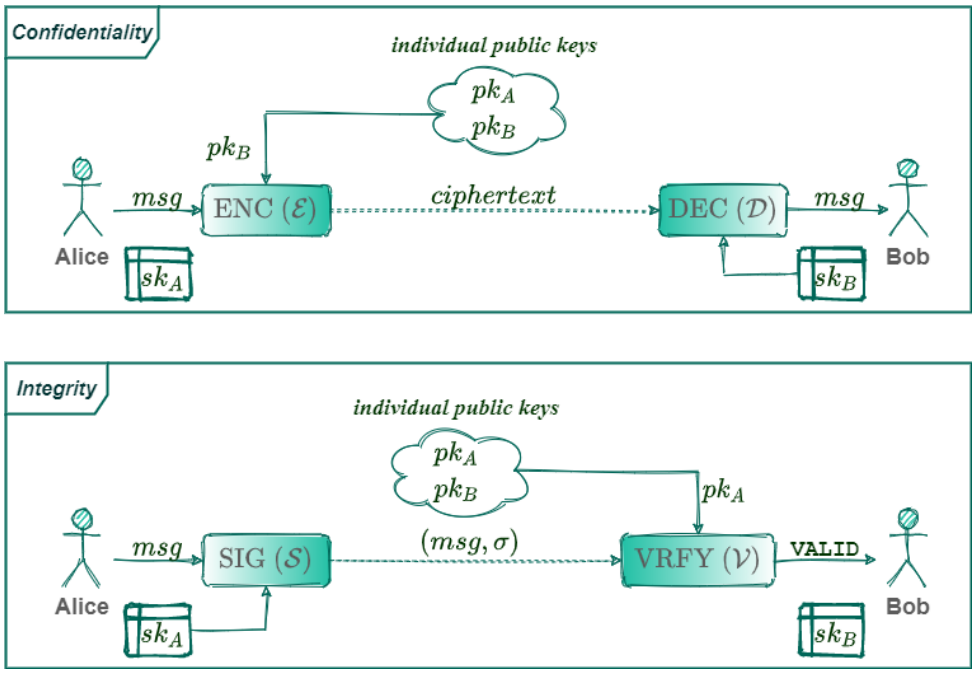


Figure 2.2: Asymmetric Cryptosystems with confidentiality and integrity

2.1 Mathematical concepts

In this section, some of the fundamental mathematical concepts necessary to construct cryptosystems are defined. This is followed by a discussion on modern cryptography and its principles.

Definition 2.1. ***Abelian Group***
An ***abelian (or commutative)*** **group** $(\mathbb{G}, *)$ is a set of elements $\mathbb{G}$ with a binary operation $*$, satisfying the below properties ([Sma16, Sho09]):

i) ***Associative:*** $\forall\ a, b, c \in \mathbb{G} : a * (b * c) = (a * b) * c$

ii) ***Identity:*** $\forall\ a \in \mathbb{G}\ \exists\ e \in \mathbb{G} : a * e = e * a = a$

iii) ***Inverse:*** $\forall\ a \in \mathbb{G},\ \exists\ a' \in \mathbb{G} : a * a' = a' * a = e$

iv) ***Commutative:*** $\forall\ a, b \in \mathbb{G} : a * b = b * a$

The abelian group $\mathbb{G}$ contains an *identity* and only one *inverse* for every element of $\mathbb{G}$. Some well-known examples for abelian groups are the set of integers under addition $(\mathbb{Z}, +)$ and the set of rationals under multiplication $(\mathbb{Q}, \times)$.

Definition 2.2. ***Commutative Ring***
A ***commutative ring*** $(\mathbf{R}, *, \circ)$ is a set $\mathbf{R}$ with two binary operations, $*$ and $\circ$, satisfying the below properties ([Sma16, Sho09]):

i) $(\mathbf{R}, *)$ *must be an abelian group*

ii) ***Associative under*** $\circ$***:*** $\forall\ a, b, c \in \mathbf{R} : a \circ (b \circ c) = (a \circ b) \circ c$

iii) ***Identity under*** $\circ$***:*** $\forall\ a \in \mathbf{R}\ \exists\ i \in \mathbf{R} : a \circ i = i \circ a = a$

iv) ***Distributive:*** $\forall\ a, b, c \in \mathbf{R} : a \circ (b * c) = (a \circ b) * (a \circ c)$

v) ***Commutative under*** $\circ$***:*** $\forall\ a, b \in \mathbf{R} : a \circ b = b \circ a$

Some well-known examples for commutative rings are the set of rationals under addition and multiplication $(\mathbb{Q}, +, \times)$ and the set of complex numbers under addition and multiplication $(\mathbb{C}, +, \times)$.

Definition 2.3. ***Field***
A ***Field*** $(\mathbb{F}, *, \circ)$ is a set $\mathbb{F}$ with two binary operations, $*$ and $\circ$, satisfying the below properties ([Sma16, Sho09]):

i) $(\mathbb{F}, *, \circ)$ *must be a commutative ring*

ii) ***Inverse under*** $\circ$***:*** $\forall\ a, a' \neq 0$ *and* $a \in \mathbb{F},\ \exists\ a' \in \mathbb{F} : a \circ a' = a' \circ a = i$
(i.e. every non-zero element must have a multiplicative inverse)

Some well-known examples for fields are the set of integers modulo a prime number p, under addition and multiplication $(\mathbb{Z}_p, +, \times)$ and the set of reals under addition and multiplication $(\mathbb{R}, +, \times)$.

Modern Cryptography

Traditional cryptography was mostly based on ad-hoc constructions that were designed as reactions to newer attacks mounted on existing ciphers. One of the assumptions was that information is protected when the ciphers are designed in such a way that no *adversary* is able to break them. In contrast, modern cryptography stresses on the importance of viewing cryptography as a rigorous science rather than an art of hiding information. It has been developed with the learnings and experience from classical cryptography, and ensures that no ad-hoc designs are either constructed or widely used. Modern cryptography does not assume that the cryptosystems are unbreakable, but relaxes the requirement to ensure that cryptosystems are *computationally* unbreakable. That is, an adversary does not attempt to break the cryptosystem because it is computationally expensive.

The three main principles of modern cryptography are [KL20] :

1. *Formulating rigorous and precise definitions of security*
2. *Precisely stating minimal, unbroken assumptions*
3. *Rigorous proof of security w.r.t the definition and relative to the assumption*

Developing formal and exact definitions of security is very essential, as it gives clarity about the security goals to be achieved for the cryptosystem being designed. With a clear understanding of the need, design efforts can be focused towards achieving those goals in the most efficient way. Features that are not relevant or do not add security enhancements to the cryptosystem can be filtered out, once precise security definitions are outlined. Also, the cryptosystem must be easy to use, while giving the required security at the same time. If it has redundant processes, or does not apply to a particular use case, or has no scalability, then the scheme would be quite useless even if it is very secure. Having accurate definitions of security also gives the flexibility of a efficiency-security trade-off, and helps to achieve the best possible balance between the two in different scenarios.

Since modern cryptography relies on computational complexity theory for the security of cryptosystems, the security is based on mathematical assumptions on

problems that are so difficult to become almost unsolvable by computers. Therefore, the assumptions need to be clearly stated right at the beginning before providing the security proof. When these assumptions have been studied and tested for long enough with no provable contradictions, then we can claim some level of soundness on them. Stating the assumptions also helps to compare and evaluate two different cryptographic constructions on the basis of the assumptions. Security based on a weaker assumption which is simpler or well-studied, maybe preferred over a new but stronger assumption, as it may turn out to be false at a later point of time. Different security proofs can be based on the same assumption, allowing to study (i.e. cryptanalyse) the proof and the assumption to evaluate if the cryptosystem meets the security definitions.

With the definitions and assumptions stated explicitly, the final most important step is providing the proof of security. Proof of security and correctness are absolutely necessary in cryptography as unlike other software, cryptographic algorithms are targeted by attackers using sophisticated resources and techniques with the sole intent to break them. Thus, a rigorous security proof assures that it will not be possible to mount such attacks by definition, and also avoids the potential damage of using insecure systems. Security proofs in modern cryptography follow the ***reductionist approach***, where the proof is given by reducing the problem of the mathematical assumption to the problem of breaking the cryptographic construction. A security proof consists of a series of sound arguments showing that an adversary who can break the cryptosystem can be used as a sub-routine to break the underlying mathematical assumption.

2.2 Signatures and EUF-CMA

A digital signature is different from a physical signature in many aspects. For instance, while a physical signature is the same on all documents irrespective of their content, a digital signature is constructed based on the content of every message to be signed. The formal definition of a digital signature is given in Definition 2.4, while a list of the basic requirements that it must fulfill are stated below [McA16, KL20].

- ***Authentication***: assurance to the receiver/verifier $\mathcal{V}$ that the signed message has in fact come from the sender/signer $\mathcal{S}$
- ***Unforgeability***: assurance that the signature has been generated by the claimed signer $\mathcal{S}$
- ***Non-Reusability***: malicious actors cannot reuse the signature on other messages that are not originally signed by the signer $\mathcal{S}$

- ***Unalterability***: the signature assures that the message has not been altered in transit
- ***Non-Repudiation***: if the signature is valid, the signer $\mathcal{S}$ cannot later deny having signed the document
- ***Public Verifiability***: the signature can easily be verified by anyone who has access to the public key pk of the signer $\mathcal{S}$
- ***Transferability***: the signature is transferable to third parties for verification and/or for solving disputes.

Note ([Sho09]) An algorithm $\mathcal{A}$ is called ***polynomial-time*** if its running time is bounded by the polynomial $n^c + d$ in the input length n, where c and d are constants. The polynomial-time algorithm is ***probabilistic*** or ***randomized*** if its outputs are random variables that take values from a probability distribution.

Definition 2.4. ([KL20, Kat10]) A ***digital signature (SIG) scheme*** consists of a tuple of three probabilistic, polynomial-time (PPT) algorithms *(Gen, Sign, Vrfy)* that are defined as below:

Key Generation (*Gen*(1^λ)): Given a security parameter λ, *Gen* outputs a keypair $(sk, pk) \leftarrow$ *Gen*(1^λ), where sk is a private *signing* key and pk is public *verification* key.

Signature Generation (*Sign*(sk, m)): Given a message $m \in \{0,1\}^*$, i.e. an arbitrary length message or hash of the message, *Sign* takes the signing key sk as input and produces a valid signature $\sigma \leftarrow$ *Sign*(sk, m).

Verification (*Vrfy*(pk, m, σ)): *Vrfy* takes in input a message $m \in \{0,1\}^*$, a signature σ and a verification key pk and outputs a bit $b :=$ *Vrfy*(m, σ, pk) such that $b = 1$ if σ is VALID and $b = 0$ implies σ is INVALID.

Gen and *Sign* are randomized algorithms while *Vrfy* is usually a deterministic one. Additionally, the *Correctness* property must be satisfied, that is, for a given security parameter λ and $\forall\ (sk, pk) \leftarrow$ *Gen*(1^λ), and $\forall\ m \in \{0,1\}^*$

$$\textit{Vrfy}(pk, m, \textit{Sign}(sk, m)) = 1$$

The security of signature schemes is summarized through the notion of Existential Unforgeability under adaptive Chosen Message Attack (EUF-CMA), defined formally in Definition 2.5. The term adaptive means that an adversary can choose and adapt his queries to the *signing oracle* by analysing the previously obtained oracle responses. This is illustrated through a security game as shown in Figure 2.3.

Definition 2.5. ([KL20, Kat10]) A signature (SIG) scheme, $\Pi = (Gen, Sign, Vrfy)$ is ***Existentially Unforgeable under adaptive Chosen Message Attack***, if for all probabilistic, polynomial-time (PPT) adversaries $\mathcal{A}$, the probability that $\mathcal{A}$ can forge a valid signature (m', σ') is negligible, i.e.

$$Pr[\mathcal{A}_\Pi(Vrfy(pk, m', \sigma') = 1)] \leq \boldsymbol{negl}(\lambda)$$

Note ([KL20]) A function $\varepsilon : \mathbb{N} \to [0, 1]$ is ***negligible***, if $\forall\ c \geq 0,\ \exists\ k_c \geq 0$ such that

$$\varepsilon(k) < \frac{1}{k^c}; \quad \forall\ k > k_c$$

.

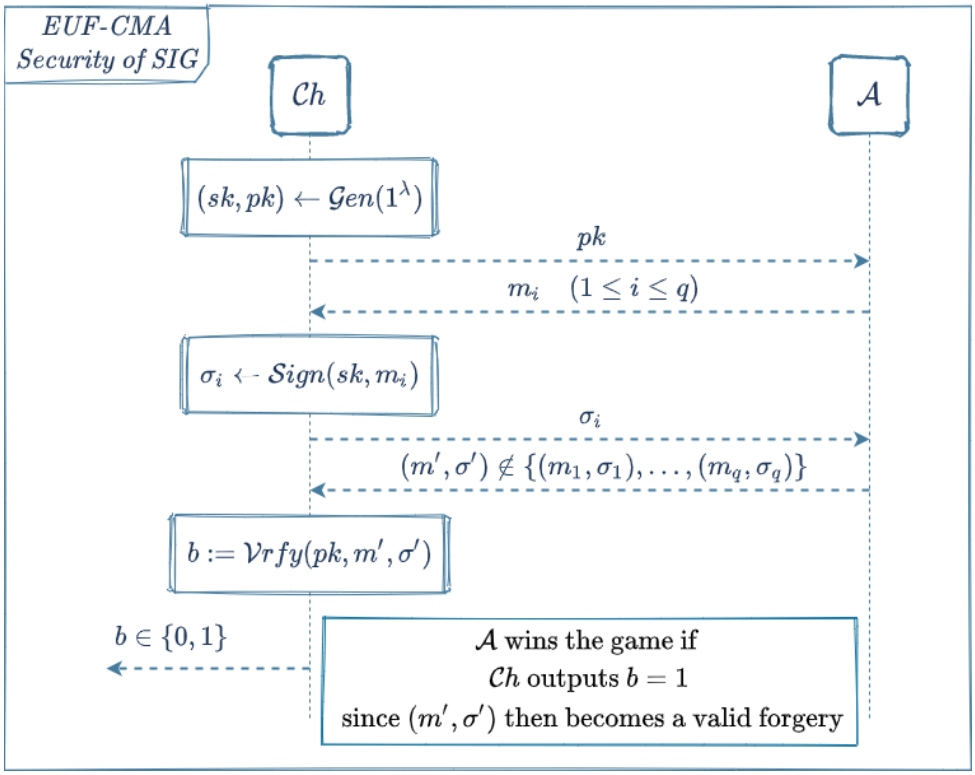


Figure 2.3: EUF-CMA Security of a Signature scheme

1. *The Challenger ($\mathcal{C}h$) simulates the SIG scheme, generating the key-pair, $(sk, pk) \leftarrow \mathcal{G}en(1^\lambda)$ and giving the public key (pk) to the adversary $\mathcal{A}$*
2. *The adversary ($\mathcal{A}$) queries the* ***Signing Oracle*** *(i.e. $\mathcal{C}h$) with chosen messages m_i and gets back corresponding signatures σ_i. The adversary is allowed at most* ***q*** *queries to the oracle.*

3. *Having obtained the set of signature-message pairs from the signing oracle,* $\mathcal{Q} = \{(m_1, \sigma_1), \ldots, (m_q, \sigma_q)\}$*, the adversary* $(\mathcal{A})$ *outputs a new message-signature pair* $(m', \sigma') \notin \mathcal{Q}$

4. *The adversary* $(\mathcal{A})$ *wins the game if and only if,* $\mathcal{C}h$ *outputs* $b := \mathcal{V}rfy(pk, m', \sigma') = 1$ *and* $(m', \sigma') \notin \mathcal{Q}$

2.3 Post-Quantum Cryptographic categories

The different families of mathematical hard problems that are believed to be resistant to quantum attacks, are discussed in this section, which also addresses one of the objectives of this thesis. There are mainly five categories that have been explored over the last decade w.r.t quantum-resistant hard mathematical problems. Each of these categories is briefly explained in the sub-sections below.

1. *Lattice-based cryptography*

2. *Multivariate-polynomials based cryptography*

3. *Hash-based cryptography*

4. *Code-based cryptography*

5. *Isogeny-based cryptography*

2.3.1 Lattice-based cryptography

A lattice can be visualized as a grid-like structure in an n-dimensional space. It is constructed using integer points that are usually elements of a group. A set of vectors called *basis* vectors are used to generate the lattice. Lattices have been studied extensively over the last decade, to explore new problems or problem instantiations that can be resistant to quantum attacks. This intractability of lattice-based problems, or absence of efficient algorithms to solve them, is what forms the basis of post-quantum cryptosystem constructions. A formal definition of a lattice and its basis vectors are given below [Pei16, Ant17].

Definition 2.6. An n-dimensional *lattice* $\mathcal{L}$ is any subset of $\mathbf{R}^n$ that is both [Pei16]:

1. *an additive subgroup: for every* $\mathbf{x}, \mathbf{y} \in \mathcal{L}$, $\exists\ -\mathbf{x} \in \mathcal{L}$ *such that* $\mathbf{x} + (-\mathbf{x}) = 0 \in \mathcal{L}$ *and also* $\mathbf{x} + \mathbf{y} \in \mathcal{L}$

2. *discrete: every* $\mathbf{x} \in \mathcal{L}$ *has a neighbourhood in* $\mathbf{R}^n$ *in which* $\mathbf{x}$ *is the only lattice point.*

Definition 2.7. Every non-trivial lattice $\mathcal{L}$, finite or infinite, is generated by the integer linear combinations of linearly independent *basis* vectors $\mathbf{B} = \{b_1, \ldots, b_k\}$ ([Pei16]) i.e.

$$\mathcal{L} = \mathcal{L}(\mathbf{B}) := \mathbf{B} \cdot \mathbb{Z}^k = \left\{ \sum_{i=1}^{k} z_i \boldsymbol{b}_i : z_i \in \mathbb{Z} \right\}$$

where k is the *rank* of the basis. If $k = n$, such lattices are called *full-rank* lattices.

Given the formal definitions of lattices, a few of the important problems in lattices that are conjectured to be resistant to quantum attacks are defined below. Post-quantum encryption and/or signature schemes are constructed based on different instantiations of these problems.

Problem 2.8. Shortest Vector Problem (SVP) ([Pei16])

- The *minimum distance* of a lattice $\mathcal{L}$ is the length of the shortest non-zero lattice vector:
$$\lambda_1(\mathcal{L}) := \min_{\mathbf{v} \in \mathcal{L} \setminus \{0\}} \|\mathbf{v}\|$$
- Given an arbitrary basis $\mathbf{B}$ of some lattice $\mathcal{L} = \mathcal{L}(\mathbf{B})$, find a shortest non-zero lattice vector $\mathbf{v} \in \mathcal{L}$, for which $\|\mathbf{v}\| = \lambda_1(\mathcal{L})$.

Lattice Basis Reduction (LBR) and *Closest Vector Problem (CVP)* are some of the problems that can be seen as variants or extensions of the SVP. A lattice has something called *"Good"* and *"Bad"* basis vectors, which determine how the lattice structure *"looks like"*, i.e. each lattice has many basis, and a good basis is formed by vectors that are close to be the shortest and orthogonal. The lattice structure defines how the key-pair is generated for cryptosystems. Therefore, this set of problems aim to find an efficient algorithm to find the shortest length vector that can reconstruct the lattice and thus recover the private key.

Problem 2.9. Short Integer Solution (SIS) ([Pei16])
Given m uniformly random vectors $\boldsymbol{a}_i \in \mathbb{Z}_q^n$, which are the columns of a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, find a non-zero integer vector $\mathbf{z} \in \mathbb{Z}^m$ of *norm* $\|\mathbf{z}\| \leq \beta$, where $\beta > 0$, such that

$$f_{\mathbf{A}}(\mathbf{z}) := \mathbf{A}\mathbf{z} = \sum_i \boldsymbol{a}_i \cdot z_i = 0 \in \mathbb{Z}_q^n$$

The SIS problem serves as the basis for one-way functions, collision-resistant hash functions, digital signatures and other cryptographic protocols.

Problem 2.10. Decision Ring-Learning With Errors (D-RLWE) ([Pei16])

- Given a (*cyclotomic*)[1] quotient ring $\mathbf{R}_q = \mathbf{R}/q\mathbf{R}$ and an *error* distribution χ over $\mathbf{R}$, for a *secret* $s \in \mathbf{R}_q$, the *R-LWE distribution* $A_{s,\chi}$, with $a \xleftarrow{\$} \mathbf{R}_q$ and $e \xleftarrow{\$} \chi$, is given by $A_{s,\chi} = \{(a, b = s \cdot a + e \pmod q)\}$.
- Given m independent samples $(a_i, b_i) \in \mathbf{R}_q \times \mathbf{R}_q$, the *Decision Ring-LWE* problem is to distinguish whether the samples have been taken from the R-LWE distribution $A_{s,\chi}$ or the Uniform distribution $\mathcal{U}$.

The D-RLWE problem serves as the basis for encryption schemes and acts as the dual of the SIS problem. This is a particular instantiation of the decision LWE problem over a ring, the other one being the search problem in Ring-LWE.

2.3.2 Multivariate-polynomial based cryptography

The linear system of equations is a well-known problem that can be solved efficiently, at least for some instances. However, it is also a fact that as the degree of the polynomials increase, and there are many variables, the solution to such a system is either not known to exist or the solution is hard to obtain. Based on this premise, the multivariate polynomial system of equations have been widely studied by the cryptographic community to exploit the hardness of this problem and construct quantum-resistant cryptosystems.

The definition of multivariate cryptography, specifically the structure of the public and private keys, is given below and summarized in Figure 2.4, where the parameters (i) m is the number of equations, (ii) n is the number of variables and (iii) d is the degree of the polynomials. To construct efficient cryptosystems, it is required that $m \gg n$ for encryption schemes and $m < n$ for signature schemes.

Definition 2.11. MPKC ([Pet17, DP17])
The *multivariate public-key cryptosystem (MPKC)* is characterized by a system of quadratic polynomials defined over a finite field $\mathbb{F}$ with q elements, given by,

$$
\begin{aligned}
p_1(x_1, \ldots, x_n) &= \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_{ij}^{(1)} x_i x_j + \sum_{i=1}^{n} \beta_i^{(1)} x_i + \gamma^{(1)} \\
p_2(x_1, \ldots, x_n) &= \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_{ij}^{(2)} x_i x_j + \sum_{i=1}^{n} \beta_i^{(2)} x_i + \gamma^{(2)} \\
&\vdots \\
p_m(x_1, \ldots, x_n) &= \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_{ij}^{(m)} x_i x_j + \sum_{i=1}^{n} \beta_i^{(m)} x_i + \gamma^{(m)}
\end{aligned}
$$

[1] n^{th} cyclotomic field is obtained by adjoining the primitive n^{th} root of unity ξ_n to the rationals $\mathbb{Q}$ [Hø13]

The public key $\mathcal{P}$ of an MPKC consists of an invertible quadratic map $\mathcal{F} : \mathbb{F}^n \to \mathbb{F}^m$, which is masked or scrambled using two linear or affine maps $\mathcal{S} : \mathbb{F}^m \to \mathbb{F}^m$ and $\mathcal{T} : \mathbb{F}^n \to \mathbb{F}^n$. Thus, the public key is the composition of all these three maps, i.e. $\mathcal{P} = \mathcal{S} \circ \mathcal{F} \circ \mathcal{T}$, and the private key is the set of individual maps $(\mathcal{S}, \mathcal{F}, \mathcal{T})$.

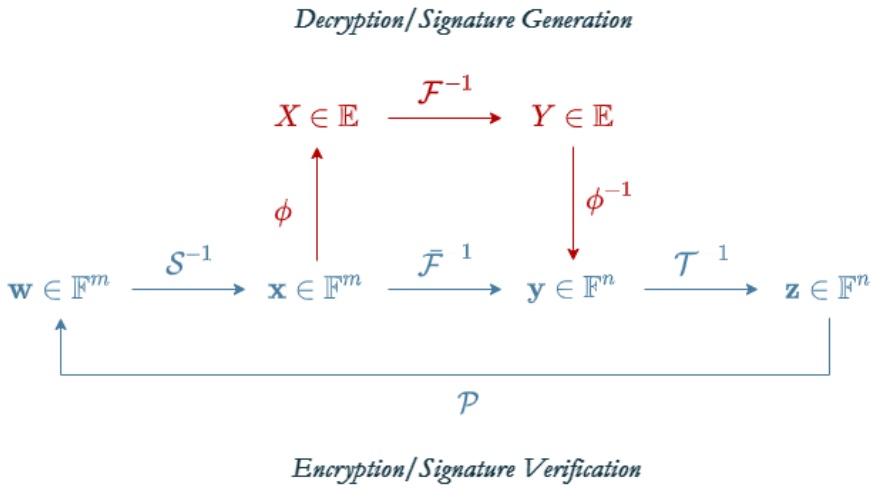


Figure 2.4: Cryptosystems constructed using MPKC [DP17, Pet17]

An MPKC is based on the Multi-variate Quadratic equations (MQ) problem which is believed to be hard to solve on average, for both classical and quantum computers. Since two of the signature schemes (GeMSS and Rainbow) discussed in this thesis are based on the MQ problem, it has been described in detail in Section 5.2. The MQ problem has been proven to be non-deterministic polynomial-time (NP)-hard over any field [DP17]. There are two instantiations of the MQ problem in MPKCs (a) *Unbalanced Oil and Vinegar (UOV)* schemes and (b) *Hidden Field Equation (HFE)* schemes. These are also called the *Small field* and *Big field* schemes, respectively, due to the fields over which they are defined and the level of operations involved. The UOV design can be represented by the blue portion in Figure 2.4 while the additional red portion represents the HFE design, where the operations are performed over a degree n extension field $\mathbb{E}$ of the base field $\mathbb{F}$.

2.3.3 Hash-based cryptography

One of the essential primitives in cryptography is the *hash* function. The output of a hash function computed on a message or plaintext is called the *digest*. The digest serves as a sort of a digital fingerprint of the message and therefore, specifically for digital signatures, signing the digest ensures message integrity instead of signing the

actual message. This also speeds up the signing and verification processes since hash functions are deterministic and usually have a fixed output length. The definition of a cryptographic hash function and important security notions related to hash functions are described below.

Definition 2.12. Hash-function families ([Hül19])
A cryptographic hash function family is a set of hash functions that take arbitrary length messages and a random key k as input, producing fixed length output, and is given by,

$$\mathbf{H}_n := \{\boldsymbol{h}_k : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n\}$$

where $m \geq n$ and $k \xleftarrow{\$} \{0,1\}^n$. $\boldsymbol{h}_k$ is called a *keyed* hash function and the value k is not secret, but a publicly known value such as an initialization vector.

In order for the hash function to be cryptographically secure, it must satisfy the following properties.

One-Wayness / Pre-Image Resistance (OW):
Given a hash function family $\mathbf{H}_n$ and the challenge y_c, a PPT adversary $\mathcal{A}$ cannot find a value $x' \leftarrow \{0,1\}^N$ such that, $y_c \leftarrow \boldsymbol{h}_k(x')$, where $\boldsymbol{h}_k \xleftarrow{\$} \mathbf{H}_n$ is a hash function sampled randomly from $\mathbf{H}_n$ for key k.

Collision Resistance (CR):
Given a hash function family $\mathbf{H}_n$, a PPT adversary $\mathcal{A}$ cannot find two distinct input values $x_1 \neq x_2$ to the function $\boldsymbol{h}_k$ such that, $\boldsymbol{h}_k(x_1) = \boldsymbol{h}_k(x_2)$.

Second Pre-Image Resistance (SPR):
Given a hash function family $\mathbf{H}_n$, an input value x_1 and corresponding challenge $y_1 = \boldsymbol{h}_k(x_1)$, a PPT adversary $\mathcal{A}$ cannot find another distinct value x_2, i.e. $x_2 \neq x_1$ such that, $\boldsymbol{h}_k(x_2) = \boldsymbol{h}_k(x_1)$.

Undetectability (UD):
Given a hash function family $\mathbf{H}_n$ and the challenge y_c, a PPT adversary $\mathcal{A}$ cannot distinguish between the digest $y_c \leftarrow \boldsymbol{h}_k(x)$, where $x \xleftarrow{\$} \{0,1\}^m$, and a random string $y_c \xleftarrow{\$} \{0,1\}^n$.

Pseudo-Randomness (PR):
Given access to a *random oracle* $\boldsymbol{g}$ (i.e. a black-box function), a PPT adversary $\mathcal{A}$ cannot distinguish the output of $\boldsymbol{g}$, i.e. the outputs between the pseudo-random function $\boldsymbol{g} \xleftarrow{\$} \boldsymbol{h}_k$ where $k \xleftarrow{\$} \{0,1\}^n$, and a truly random function $\boldsymbol{g} \xleftarrow{\$} \mathcal{F}_{N,n}$ for a queried input x to the oracle.

Table 2.1 lists the bounds (or limit) on the number of queries that can be allowed to an adversary to break the security properties of a hash function as defined above.

Generic Security	OW	SPR	CR	UD*	PRF*
Classical	$\Theta(2^n)$	$\Theta(2^n)$	$\Theta(2^{n/2})$	$\Theta(2^n)$	$\Theta(2^n)$
Quantum	$\Theta(2^{n/2})$	$\Theta(2^{n/2})$	$\Theta(2^{n/3})$	$\Theta(2^{n/2})$	$\Theta(2^{n/2})$

Table 2.1: Query bounds for the security notions in classical and quantum settings (* represents conjectured bounds) [Hül19]

Definition 2.13. Birthday Paradox ([KL20])
Let $N > 0$, and let q elements $y_1, \ldots, y_q$, where $q \leq \sqrt{2N}$, be chosen uniformly and independently at random from a set of size N. Then the probability that there exist distinct i, j with $y_i = y_j$ is at least $q(q-1)/4N$ and at most $q^2/2N$ i.e.

$$\frac{q(q-1)}{4N} \leq \mathsf{coll}(q, N) \leq \frac{q^2}{2N}$$

The *Birthday Paradox* defines the bounds on the collision resistance of a hash function. It is called the birthday problem because of the analogy to finding the number of people to be filled in a room (in this case, 23 people), to get two persons having matching birthdays with a probability $\geq 1/2$.

2.3.4 Code-based cryptography

Code-based cryptography is an interesting alternative that has been studied for a few decades now to provide secure cryptographic constructions. The exchange of data in the binary format of 0's and 1's in digital communication also made the detection and correction of transmission errors possible. Proper encoding and decoding of information at the sending and receiving sides respectively was made possible through the use of *error-correcting codes*. Using the same concept behind error-correcting codes, the difference in the complexities of decoding the received word for induced errors (especially for large parity check matrices) has been used to construct cryptosystems (i.e. encryption schemes). Some important concepts of information theory and code-based cryptography are discussed in this section ([Lan16]).

Entropy ($\mathsf{H}(\mathsf{X})$): ([McA16])
Entropy can be defined as the measure of uncertainty of an outcome or the amount of information obtained from an outcome, given by $\mathsf{H}(\mathsf{X}) = -\log_2 p$, where p is the probability of the outcome. The entropy associated with a random

variable X with k outcomes, each with different probabilities $p_1, p_2, \ldots, p_k$ is given by

$$\mathsf{H}(\mathsf{X}) = -\sum_{i=1}^{k} p_i \log_2 p_i \quad \text{and} \quad \mathsf{H}(\mathsf{X},\mathsf{Y}) \leq \mathsf{H}(\mathsf{X}) + \mathsf{H}(\mathsf{Y})$$

.

Hamming Weight ($wt(\boldsymbol{x})$):
The Hamming Weight of a codeword $\boldsymbol{x} \in \mathcal{C}$ is the number of non-zero elements in $\boldsymbol{x}$.

$$\text{e.g.} \quad wt(\boldsymbol{x} = (11010)) := 3$$

Hamming Distance ($d(\boldsymbol{x}, \boldsymbol{y})$):
The Hamming Distance between two code-words is the number of elements (or co-ordinates) by which they differ. The hamming distance between $\boldsymbol{x}$ and $\boldsymbol{y}$ is equal to the hamming weight of $\boldsymbol{x} + \boldsymbol{y}$ i.e.

$$d(\boldsymbol{x}, \boldsymbol{y}) = wt(\boldsymbol{x} + \boldsymbol{y})$$

$$\text{e.g.} \quad d(\boldsymbol{x}, \boldsymbol{y}) = ((11010), (10110)) = 2$$

Systematic Generator Matrix $\mathcal{G}$:
A matrix of the form $\mathcal{G} = (I_k|Q)$, where $I_k \to (k \times k)$ Identity matrix and $Q \to k \times (n-k)$ redundant matrix, is called the *Systematic Generator* matrix which is used to generate the codeword $\mathcal{C}$.

Parity Check Matrix $\mathcal{H}$:
A matrix given by $\mathcal{H} = (-Q^T|I_{n-k})$, where $Q^T \to (n-k) \times k$ is the transpose of Q and $I_{n-k} \to (n-k) \times (n-k)$ Identity matrix, is called the *Parity Check* matrix that, as the name suggests, checks the parity of the received word (crudely, how many bits have been flipped during transmission).

The above definitions set the premise to define the *Linear Codes* in coding theory.

Definition 2.14. Linear Codes ([Lan16])
A binary linear code $\mathcal{C}$ of length n and dimension k is defined as a k-dimensional subspace of $\mathbb{F}_2^n$. $\mathcal{C}$ is defined in one of two ways:

a) as the row space of the generating matrix $\mathcal{G} \in \mathbb{F}_2^{k \times n}$ where $\mathcal{C} = \{\boldsymbol{m}\mathcal{G} : \boldsymbol{m} \in \mathbb{F}_2^k\}$

b) as the kernel space of the parity-check matrix $\mathcal{H} \in \mathbb{F}_2^{(n-k) \times n}$ where $\mathcal{C} = \{\boldsymbol{c} : \mathcal{H}\boldsymbol{c}^T = 0, \boldsymbol{c} \in \mathbb{F}_2^n\}$.

Note ([Lan16]) The *minimum distance* of a linear code $\boldsymbol{b} \in \mathcal{C}$ is the smallest Hamming weight of a non-zero code-word $\boldsymbol{c} \in \mathcal{C}$. i.e.

$$d_{min} = \min_{\boldsymbol{c} \neq 0}\{wt(\boldsymbol{c})\} = \min_{\boldsymbol{b} \neq \boldsymbol{c}}\{d(\boldsymbol{b}, \boldsymbol{c})\}$$

Problem 2.15. ***Decoding Problem:*** ([Lan16]) Given a vector $\boldsymbol{x} = \boldsymbol{c} + \boldsymbol{e}$ with $wt(\boldsymbol{e}) \leq t$, find the closest code-word $\boldsymbol{c} \in \mathcal{C}$ that is unique, i.e. has *minimum distance* $d_{min} = 2t + 1$. In this process, finding the error vector $\boldsymbol{e}$ is an equivalent decoding problem.

The *Decoding problem* is the basis on which cryptographic constructions are developed in code-based cryptography. A number of other different complex codes like the *Reed-Solomon* codes and binary *Goppa* codes are also used in the code-based cryptographic constructions.

2.3.5 Isogeny-based cryptography

Isogeny-based cryptography (IBC) is a relatively recent area of cryptography, specifically a branch of *Elliptic Curve Cryptography* (ECC), that has gained popularity over the last decade. Compared to ECC and its other branches, IBC is conjectured to have quantum-resistant properties, and therefore has gained the attention of the cryptographic community. There are many key-exchange and public encryption schemes built using IBC, one of them being SIKE that has been submitted to the NIST PQC standardization. For signature schemes though, *zero-knowledge* protocols are used to construct signatures using *isogeny* assumptions, but even so, constructing signature schemes is an open problem. A few basic IBC concepts and problems are discussed in this section [DF17].

Note Let k be a field, with *characteristic* $p \neq 2$ or 3, and let $\bar{k}$ be its *algebraic closure* [2]. A *projective space* of dimension n, when there exists $\lambda \in \bar{k}$ with an equivalence relation $x_i \sim y_i$, is given by $\mathbf{P}^n(\bar{k}) := \{(x_0, \ldots, x_n) \mid x_i \neq 0 \ \wedge \ x_i = \lambda_i y_i \ \forall \ i\}$. The set of *k-rational points* is defined as $\mathbf{P}^n(k) := \{(x_0 : \cdots : x_n) \in \mathbf{P}^n \mid x_i \in k \ \forall \ i\}$, and fixing an arbitrary $x_n = 0$ gives the *space* and its *points at infinity*. ([DF17])

Elliptic Curves ($\mathcal{E}$): An elliptic curve over a finite field k, can be defined in an equivalent *affine* form as the *locus* (in $\mathbf{P}^2(\bar{k})$) of the equation $\mathcal{E} : y^2 = x^3 + ax + b$, with $a, b \in k$ and $4a^3 + 27b^2 \neq 0$, along with the point at infinity $\mathcal{O} = (0 : 1 : 0)$ for the elliptic curve.

[2] $GF(p^\infty)$ is the *algebraic closure* of $GF(p^e)$ for any $e \geq 1$. [MM+74]

The graphs in Figure 2.5 illustrate two types of elliptic curves, one defined over $\mathbb{R}$ and the other over a finite field k, along with the geometric representation of the group operations.

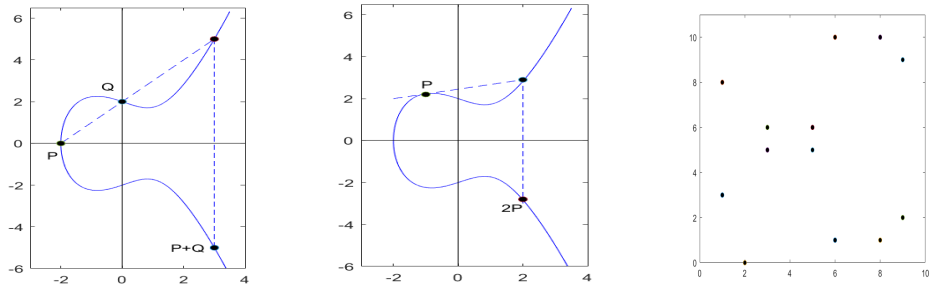


Figure 2.5: An elliptic curve defined over a finite field $GF(11)$ (right), along with the group operations of addition (left) and doubling (middle)

Supersingularity: For elliptic curves defined over a field of characteristic $p > 0$, the case where $\mathcal{E}[p] \simeq \mathbb{Z}/p\mathbb{Z}$ is called the *ordinary* p^{th}*-torsion* group, while $\mathcal{E}[p] \simeq \{\mathcal{O}\}$ is called the *supersingular* p^{th}*-torsion* group.

j-invariant ($j(\mathcal{E})$): The j-invariant of an elliptic curve $\mathcal{E} : y^2 = x^3 + ax + b$ is given by

$$j(\mathcal{E}) = 1728 \cdot \frac{4a^3}{4a^3 + 27b^2}$$

Two curves are said to be isomorphic over $\bar{k}$ if they have the same j-invariant.

Isogeny (φ): An *isogeny* of elliptic curves over the field k is a non-zero group morphism $\varphi : \mathcal{E} \to \mathcal{E}'$ (i.e. a map of curves preserving the group laws), that preserves the identity (i.e. the point of infinity $\mathcal{O}$ of an elliptic curve) and is given by rational maps. An isogeny is also a surjective algebraic map between two elliptic curves. Two curves are said to be *isogenous* if there exists an isogeny between them.

Isogeny Graphs: An *isogeny-graph* is a multi-graph whose nodes are the j-invariants of isogenous curves and the edges are the isogenies between them. The *expansion* property of large isogeny graphs are known to have special properties suitable for cryptography.

Problem 2.16. Explicit Isogeny ([MP, DF17]) Given two elliptic curves $\mathcal{E}, \mathcal{E}'$ defined over a finite field k and isogenous of known degree d, find such an isogeny $\varphi : \mathcal{E} \to \mathcal{E}'$ of degree d.

The problem of finding explicit (ordinary or supersingular) isogenies and/or isogeny paths, and variants thereof, is used to construct isogeny-based cryptosystems. Secure key exchange protocols and encryption schemes are built using the concept of *random walks* in an isogeny graph to provide different instantiations of the *Decisional* Diffie-Hellman problem [Mar17, MP, DF17].

There are a few other schemes that were submitted to the NIST PQC standardization, which cannot be strictly categorized into any of the above described post-quantum families, such as, WalnutDSA - based on what is called "*Braid* groups" and pqRSA - seems like a futile attempt to revive RSA cryptosystem in the post-quantum setting.

As part of addressing the first of the research questions, the current state-of-the-art post-quantum families of mathematical problems were explored, and a basic understanding of such problems was obtained in this chapter. Particularly, the post-quantum cryptographic categories that are used to construct the signature schemes described in the next part of this thesis, have been given preference over the other categories. With this, the second research question has been addressed in the next few chapters (Ch. 3 - Ch. 6), which describe the post-quantum signature schemes that have progressed into the third round of the NIST PQC standardization.

Chapter 3
Picnic

Picnic [CDG+17] is a novel signature scheme developed by Microsoft Research in collaboration with other research groups. It is entirely based on symmetric primitives and the interesting concept of Zero-knowledge proofs, which basically means that this signature scheme does not rely on any cryptographic hard problems for its security.

Well studied symmetric primitives like block ciphers and cryptographic hash functions with special properties, believed to make them secure against quantum attacks, are used to construct Picnic. The core of Picnic is what is called a Non-Interactive Zero-Knowledge (NIZK) protocol, which essentially means that no secret information is leaked during the protocol run to anybody, including the verifier of the signature, but provides enough information (*proof*) to check the validity of the generated signature. As part of the signature generation process, Picnic uses a variant of the Multi-Party Computation (MPC) [IKOS07] paradigm, which is a method of secretively[1] computing the result of a shared function between many parties. The NIZK proof is computed as an evaluation of a function on an underlying boolean circuit, based on the ZKB++ protocol [CDG+17, GMO16].

The signature scheme has two variants, Picnic-FS (Fish) and Picnic-UR (Picnic), based on Fiat-Shamir Transform (FST) [FS86] and Unruh Transform (URT) [Unr15] respectively, used to make the underlying protocols non-interactive. The signature scheme for both the variants is analysed and proved secure in the Random Oracle Model (ROM) and Quantum-accessible Random Oracle Model (QROM). These random oracles are mathematical abstractions of the cryptographic hash functions, that help to construct and provide rigorous security proofs for any cryptosystem.

The rest of the chapter is organized as follows: Section 3.1 describes the key concepts and building blocks of Picnic, Section 3.2 gives a detailed description of the signature scheme and finally, the Section 3.3 discusses the security of the scheme.

[1] i.e. without revealing the individual private inputs to the function

3.1 Key Concepts

The attractive feature of Picnic is that it is not constructed using any additional number-theoretic or algebraic hardness assumptions unlike other schemes [CDG+20]. This section gives a detailed description of the various building blocks of Picnic and the mathematical background for those building blocks.

LowMC Block Cipher

LowMC is a family of block ciphers that have low multiplicative complexity (MC) and low AND-depth, specifically designed for applications such as secure MPC, Fully Homomorphic Encryption (FHE) and Zero-Knowledge (ZK) protocols. The terms MC and AND-depth, mean the minimum number of AND gates in the underlying circuit required to represent a boolean function, and the number of multiplications (AND operations) that can be performed, respectively [ARS+16]. LowMC is used in Picnic as a one-way, pseudo-random function that generates the key-pair required for signing and verifying a given message.

LowMC is a Substitution-Permutation Network (SPN) based block cipher, which consists of 4 stages: $\texttt{LowMC} = \texttt{KeyAdd} \circ \texttt{ConstAdd} \circ \texttt{LinLayer} \circ \texttt{Sbox}$. The Sbox is a layer of m 3-bit substitution look-up tables whose entries are computed as: $\texttt{Sbox}(a, b, c) := (a \oplus bc, a \oplus b \oplus ac, a \oplus b \oplus c \oplus ab)$. After the initial key whitening step, the four layers of LowMC, as described in the Algorithm 3.1, are applied successively for each round r. The components used in each stage are a regular matrix L_i, a constant vector C_i and a key matrix K_i used to generate the round key $(K_i \cdot k_m)$ for each round.

Algorithm 3.1 LowMC block cipher $\mathsf{LowMC}(n, k, r, m)$ [ARS+16, CDG+17]

Require: plaintext $p \in \mathbb{F}_2^n$ and master key $k_m \in \mathbb{F}_2^k$ ▷ n - block size; k - key size
$L_i \in \mathbb{F}_2^{n \times n}$
Ensure: $C_i \in \mathbb{F}_2^n$ for $i \in [1, r]$ ▷ r - number of rounds
$K_i \in \mathbb{F}_2^{n \times k}$ for $i \in [0, r]$

1: $s \leftarrow K_0 \cdot k_m + p$ ▷ key whitening
2: **for** $i \in [1, r]$ **do**
3: $s \leftarrow \texttt{Sbox}(s)$ ▷ m 3-bit S-boxes
4: $s \leftarrow L_i \cdot s$ ▷ `LinLayer`
5: $s \leftarrow C_i + s$ ▷ `ConstAdd`
6: $s \leftarrow K_i \cdot k_m + s$ ▷ `KeyAdd`
7: **end for**
8: **return** s

Multi-Party Computation

Secure Multi-Party Computation (MPC) [2] is a method of computing a shared function by many parties with their secret inputs, producing a shared result without leaking any information about the secret values to anyone, including the other participants in the protocol. The so-called millionaires problem is a common example of MPC, where two or more players compute a (boolean) function to determine the richest person among the group, without revealing their actual wealth. There are two types of MPC protocols, one based on garbled binary circuits [Yao86] and the other based on secret sharing (e.g. [Sha79]) using arithmetic circuits. For the secret sharing mechanism using arithmetic circuits, the function is evaluated through a set of addition and multiplication gates defined over a finite field $\mathbb{F}_q$ [Sma16].

Specifically in Picnic, the *MPC-in-the-head* paradigm [IKOS07] is employed in the sub-routine $\textsc{Decomp}(x, k)$ as part of the signing process. This can be seen as a preprocessing step, where the prover simulates 3 parties in the protocol, that are each given their corresponding private shares x_i of the witness x. These shares are then used to compute the function $y = \phi(x)$ using the arithmetic circuit in the $\textsc{Decomp}(x, k)$ algorithm.

Commitments

Commitments are generally used in Σ-protocols by the *prover*, to prove the knowledge of a secret, by enabling him to produce a value that contains some information about the secret, which can be verified by the *verifier*. Specifically, it is used in the signing process in Picnic to generate the commitment in the non-interactive Σ-protocol.

Definition 3.1. Commitment scheme ([MWK18])
A commitment scheme is defined by the algorithms `Commit` and `Open` as described below:

`Commit`: $(C, D) := \texttt{Com}(m, r)$; given a message $m \in \{0,1\}^n$ and random value r, an output C, called *commitment*, is computed such that it masks the message m and it is hard to find m', r' such that $\texttt{Com}(m', r') = \texttt{Com}(m, r) = C$. And, D is the *de-commitment* string that contains the values (m, r) and is kept secret.

`Open`: $V := \texttt{Open}(C, D)$; given the values C and D, the message m and randomness r are obtained by opening D, and the algorithm returns true if and only if $C == \texttt{Com}(m, r)$ and outputs message m, else it returns false and outputs invalid or $\perp$.

[2] parts of the relevant definitions in this section are taken from the specialization project report [Sri19]

The commitment scheme must satisfy the following properties: (i) **Correctness:** a valid commitment always verifies correctly and outputs the message m (i.e. $\texttt{Open}(\texttt{Com}(m, r)) == m$) (ii) **Binding:** given a commitment C, it is hard to find different message-random pairs that give the same commitment value. This ensures that once committed to a message m, it cannot be opened to a different value m' (iii) **Hiding:** given only the commitment C, it should be hard to compute any information about m.

Sigma Protocols and Zero-Knowledge proofs

Sigma (Σ) Protocols and Zero-Knowledge (ZK) proofs are widely used in cryptography to conceal a secret while communicating, revealing only enough information so as to assure the validity of the information exchange and the authenticity of the communicating entities. Specifically, the Picnic signature scheme is built using the ZKB++ protocol ([CDG$^+$17]) which is essentially a Σ-protocol that gives a ZK proof of knowing the input to an arbitrary boolean circuit.

Definition 3.2. **Σ-protocol** ([Dam02, CDG$^+$17])
A Σ-protocol is a three-move interactive protocol between a PPT prover $\mathcal{P}$ and PPT verifier $\mathcal{V}$, where $\mathcal{P}$ proves the correctness of a binary relation R, such that for $(x, w) \in R$, w is a *witness* for an instance x of some computational problem. It consists of the 3-moves commit-challenge-response, described by the algorithms below

Commit: The prover selects a random value r and computes a commitment $a = Com(r, m)$ that is sent to the verifier.

Challenge: The verifier randomly selects a challenge $e \xleftarrow{\$} \{0, 1\}^t$, which is sent to the prover.

Response: The prover responds to the challenge by computing a value z, which contains the proof of knowing the secret witness w, and sends it to the verifier. The verifier checks if the commitment is valid based on the received transcript (a, e, z), accepts the proof if valid and rejects otherwise.

A Zero-Knowledge (ZK) proof protocol is one, where a prover who knows a secret w convinces the verifier that he knows w without actually revealing it. Generally, since Σ-protocols do not reveal any information about the secret value or witness, they also become ZK proofs. ZK proofs (Σ-protocols) must satisfy the following properties: (i) **Completeness:** an honest verifier will always accept a proof produced by an honest prover. (i.e. $Pr[\mathcal{V}(x) == 1 | \mathcal{P}(x, w)] = 1$) (ii) **Soundness:** A cheating prover cannot convince an honest verifier about a false statement, and the cheating probability, called *soundness error*, must be negligible (because probability to cheat in one round is 0.5). This error is reduced to a negligible value through

multiple iterations of the protocol. (iii) **Zero-knowledge:** The honest verifier must not gain any knowledge about the secret value except for the fact that the prover is aware of the secret value. (iv) s**-Special Soundness:** Given a set of s valid transcripts $\{(a, e_i, z_i)\}\forall i, j \in [s], i \neq j : e_i \neq e_j$ with respect to input x, there exists a PPT extractor $\mathcal{E}$ who can efficiently recover a valid witness w with non-negligible probability. (v) **Special Honest-Verifier Zero-Knowledge (HVZK):** There exists a PPT simulator $\mathcal{S}$ with input x and challenge e, which produces a valid transcript (a, e, z) that is computationally indistinguishable from the transcripts generated from an honest protocol run. Specifically, the last 2 properties are of higher significance for the Picnic scheme, as they ensure correctness and security of the signature scheme.

Non-Interactive Zero-Knowledge Proofs

The term *non-interactive* in a ZK proof implies that the prover publishes the statements to be proved along with the proofs and anybody can verify it, without the need of any interaction with a verifier. Two well-known techniques (described subsequently in this section): (a) Fiat-Shamir Transform (FST) (b) Unruh Transform (URT), are used to convert any Σ-protocol into a NIZK proof. In the Picnic scheme, the signing and verifying processes instantiate a NIZK protocol using both the FST and URT methods, producing two variants of the signature scheme, that is, `Picnic-FS` and `Picnic-UR` respectively.

NIZK proof schemes are defined by three functions, Setup, Prove and Verify, described as [MWK18]: (i) **Setup:** $\{\texttt{params}\} \leftarrow \texttt{Setup}(1^\lambda)$ generates the parameters required for the proof system, where λ is the security level (i.e. the number of required bits of classical or quantum security). (ii) **Prove:** $\boldsymbol{p} \leftarrow \texttt{Prove}(x, w)$ is the proof of knowledge of a secret w, which is, for example, the solution for a computationally hard problem instance x. (iii) **Verify:** $b \leftarrow \texttt{Ver}(x)$ outputs 1 if the proof is valid, and 0 if it is invalid.

(a) Fiat-Shamir Transform

Fiat-Shamir Transform (FST) converts any Σ-protocol into a non-interactive proof system [FS86]. The algorithm works similarly to the Σ-protocol, except that the challenge is computed by the prover itself, instead of receiving it from the verifier. The challenge e is computed as the hash of the commitment a, that is $e \xleftarrow{\$} H(a, (m))$ (and the message m in case of a signature scheme). Based on the value of the challenge, the corresponding response z is computed and the tuple (a, e, z) is published. The verifier extracts the commitment a' from the proof (e, z), re-computes the challenge e' as the hash of the recovered commitment and checks if $e' == e$ and $a' == a$ to either accept or reject the proof. FST is illustrated in the Figure 3.1 below.

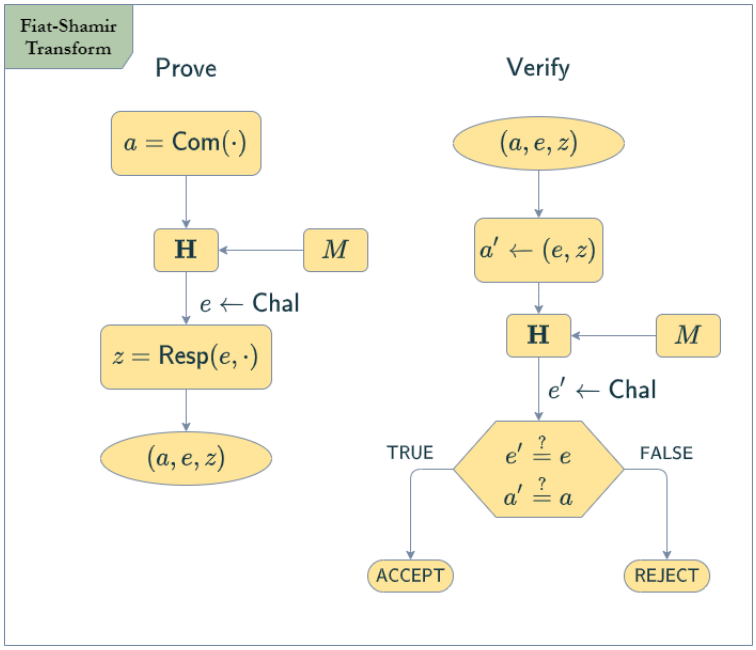


Figure 3.1: The Fiat-Shamir Transform

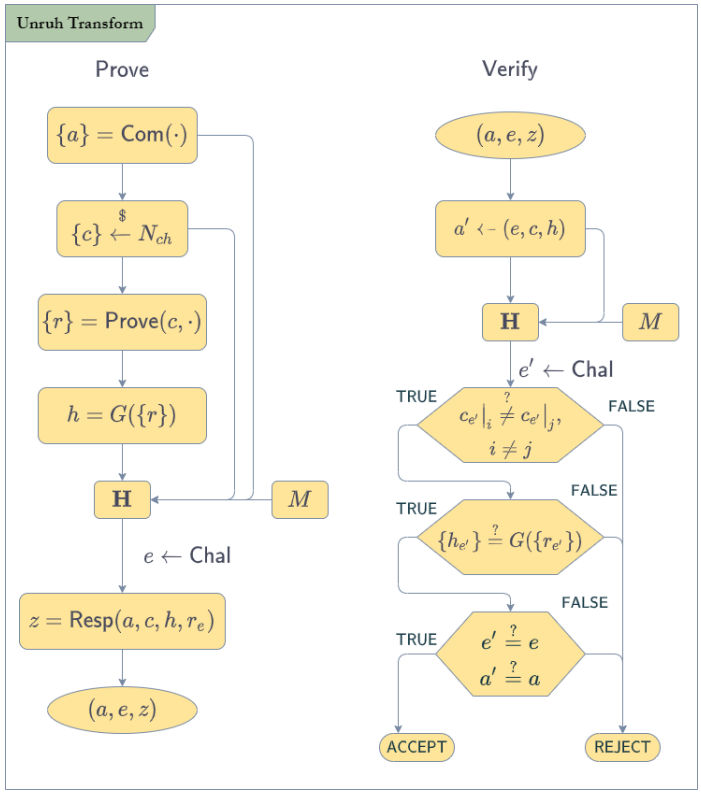


Figure 3.2: The Unruh Transform

(b) Unruh Transform

Unruh Transform (URT) is another method to convert a Σ-protocol into a non-interactive proof system [Unr15]. Similar to FST, URT also generates the challenge as the hash of the commitment but in a slightly different manner. The prover creates a number of proofs (i.e. $p := \{(\mathsf{com}, \mathsf{ch}, \mathsf{rsp})\}$) for a set of commitments $\{\mathsf{com}\}$, selects distinct challenges $\{\mathsf{ch}\}$ and computes corresponding responses $\{\mathsf{rsp}\}$. All such obtained responses are given as input to a random permutation G (i.e. $h := G(\mathsf{rsp})$) to obtain the commitments of these responses. Next, the challenge e is obtained by hashing these committed responses ($\{h\}$), individual challenges ($\{\mathsf{ch}\}$), original commitments ($\{\mathsf{com}\}$) and the message (m), to obtain and publish the final proof set $z := (\{\mathsf{com}\}, \{\mathsf{ch}\}, \{\mathsf{rsp}_e\}, \{h\})$, which contains a subset of the responses ($\{\mathsf{rsp}\}_e$) that will be revealed. During the verification process, the challenge value e' is recalculated as the hash of the message m and the response set z (excluding $\{\mathsf{rsp}_e\}$). Additionally, each of the individual extracted challenge values $\{\mathsf{ch}\}$ are verified if they are mutually distinct, the committed responses $\{h\}$ are re-computed and verified (i.e. $h \stackrel{?}{=} G(\mathsf{rsp}_{e'})$), and finally, the verification algorithm checks if $e' == e$. The $\mathsf{Ver}(m, (p)_e)$ outputs 1 if it accepts the proofs and outputs 0 otherwise. URT is illustrated in the Figure 3.2 above.

3.2 Scheme

By definition, a signature scheme consists of three parts: key generation, signing and verifying. This section outlines the details of these three algorithms as defined in the Picnic signature scheme. The algorithms also give an overview of how the various building blocks, defined in the previous section, are used to compose the entire scheme.

Key Generation

The key generation (Alg. (3.2)) in Picnic is different from general public-key cryptographic schemes because the public key is obtained through a symmetric primitive, that is, a block cipher. The block cipher used, is the LowMC block cipher (Sec. (3.1)) that has been chosen due to its special properties.

Random tapes (values) $\kappa \in \{0,1\}^\lambda$ and a secret value $x \in \{0,1\}^{c \cdot \lambda}$ are chosen randomly, where c is a constant such that $c = 1$ for classical security and $c = 2$ for quantum security. The *witness* x, from which $y = f_\kappa(x)$ is computed, becomes the pre-image of a one-way (pseudorandom) function f_κ, where $\{f_\kappa\} \stackrel{\$}{\leftarrow} F(\kappa, \cdot)$ such that $\kappa \in K_\lambda$. Or in other words, f_κ is chosen randomly from a family of one-way functions based on the random key κ, and $F(\kappa, \cdot)$ is a family of one-way functions that are instantiated as a block cipher. The LowMC block cipher is used to instantiate the

function f_κ, where the secret x is used as the key for encrypting a single block of the random tape κ giving the image y.

$$y \leftarrow f_\kappa(x) := Enc(x, \kappa) \tag{3.1}$$

Here, the public or signing key, $pk = y$ and the secret or verification key $sk = x$.

Algorithm 3.2 Key Generation $\mathsf{Gen}(1^\lambda)$ [CDG+17]

procedure KeyGen(λ) ▷ λ - security parameter
  $\kappa \xleftarrow{\$} \mathcal{K}_\lambda$ ▷ public random tapes
  $x \xleftarrow{\$} \mathcal{D}_\lambda$ ▷ secret witness
  $f_\kappa \xleftarrow{\$} F(\kappa, \cdot)$ ▷ sampled uniformly from family of OWFs
  function LowMC(x, κ) ▷ refer Algorithm 3.1
    $y \leftarrow f_\kappa(x) := Enc(x, \kappa)$
  end function
  $pk \leftarrow (y, \kappa)$ ▷ public key
  $sk \leftarrow (pk, x)$ ▷ private/secret key
  return (pk, sk)
end procedure

Signature Generation

The signature generation is based on the ZKB++ protocol [CDG+17], an improved version of the Zero-knowledge for Boolean circuits (ZKBoo) protocol [GMO16]. The ZKB++ protocol essentially generates a NIZK proof of the secret key, that is a proof where the verifier does not learn anything about the witness x, except for the correctness of the function $y = \phi(x)$. The main component of the ZKB++ protocol is the function decomposition, which is realized through an n-gate arithmetic circuit that consists of *addition-by-constant*, *multiplication-by-constant*, *binary addition* and *binary multiplication* gates [GCZ16].

The *(2,3)-Decomposition* (illustrated in Figure 3.3) simulates the MPC protocol (*MPC-in-the-head* paradigm [IKOS07]) to generate the secret shares for 3 players, while computing the decomposition of the function $\phi(x) = y$. The (2,3) in the name implies the properties of *2-privacy* and *3-special soundness*. The *2-privacy* property ensures that revealing 2 out of 3 secret shares does not reveal the witness, while the *3-special soundness* property ensures that without some information about all the 3 secret shares, it will not be possible to recover the witness. The witness x and random tapes k_j for each player P_j, are given to the `Share` function that generates input shares x_j for the 3 players. The `Update` function recursively updates the views w_j of each player at every gate in the circuit ϕ. The `Output` function takes the final views of the circuit to generate the output shares y_j, while the `Reconstruct` function combines all the output shares to reconstruct the image y such that $y = \phi(x)$.

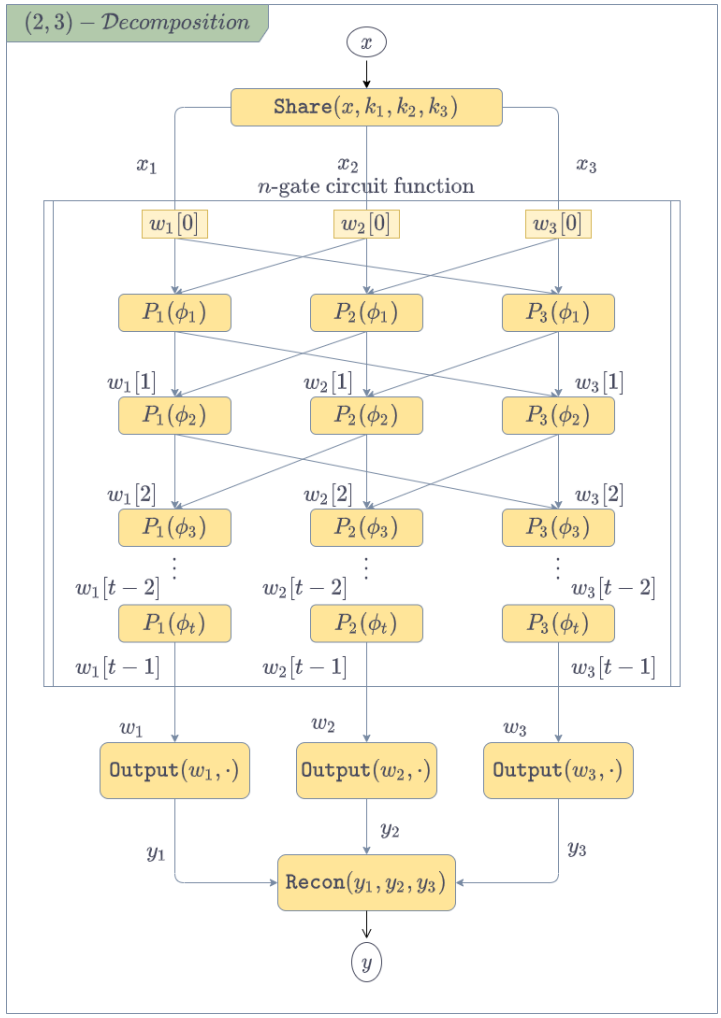


Figure 3.3: The *(2,3)-Decomposition* function [GMO16]

Coming to the Algorithm 3.3, the Prove_H function has two main components: an adapted version of the *MPC-in-the-head* paradigm, that is, the *(2,3)-Decomposition* and the non-interactive Σ-protocol (NIZK), with commitment, challenge and response messages. The Σ-protocol is made non-interactive using FST in the ROM, and using URT in the QROM.

Once the decomposition function generates the respective output shares y_j of the 3 simulated MPC players, the non-interactive Σ-protocol is instantiated by applying the Fiat-Shamir transform where the standard hash function ($\mathcal{H} : \{0,1\}^* \to \{0,1\}^n$)

is modeled as a random oracle. Here, the *prover* $\mathcal{P}$ generates commitments (C_j) as the hash ($H_C \equiv$ SHA-256) of the random tapes k_j and the views w_j of the player P_j. The challenge e is obtained by querying the random oracle, giving the commitments and output shares of the players as inputs. Based on the challenge e, the prover opens 2 out of 3 views and sends the challenge and the 2 opened views as part of the proof p. While computing the challenge, the message m to be signed is also given as input along with a random value $salt$[3], thus incorporating the message into the eventual proof. This proof p is itself the signature, which is essentially zero-knowledge, not revealing any information about the signing key x. At the end of the signing process, the prover outputs the signature in transcripts of the form $(\boldsymbol{a}, e, \boldsymbol{z})$.

Algorithm 3.3 Signature Generation $\mathsf{Sign}(sk, m)$ [CDG+17]

procedure $\textsc{Prove}_H(x)$
  require: $x \xleftarrow{\$} \mathcal{D}_\lambda$
  for $1 \le i \le t$ **do** ▷ number of iterations
    $(k_1, k_2, k_3) \xleftarrow{\$} \mathcal{K}_\lambda$
    function $(2,3) - \textsc{Decomp}(x, k_j)$ ▷ refer Figure 3.3
      ▷ MPC protocol simulated for 3 players
      $(x_1, x_2, x_3) \leftarrow \mathtt{Share}(x, k_1, k_2, k_3)$
      for $j \in [1, 2, 3]$ **do**
        $w_j \leftarrow \mathtt{Update}(\ldots \mathtt{Update}(x_j, k_j, x_{j+1}, k_{j+1}) \ldots)$ ▷ $w_j \equiv \mathtt{View}_j$
        $y_j \leftarrow \mathtt{Output}(w_j)$
        $[C_j, D_j] \leftarrow [H_C(x_j, k_j, w_j), k_j \| w_j]$ ▷ $H_C \equiv$ SHA-256
        $y \leftarrow \mathtt{Reconstruct}(y_1, y_2, y_3)$
      end for
      $\boldsymbol{a}_i \leftarrow (y_1, y_2, y_3, C_1, C_2, C_3)$
      return $(\boldsymbol{a}_i, y)$
    end function
  end for
  $\boldsymbol{a} := \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_t\}$
  $e \leftarrow H(\boldsymbol{a}, m, salt)$ ▷ challenge
  for $1 \le i \le t$ **do**
    $e_i \in [1, 2, 3]$
    $\boldsymbol{b}_i \leftarrow (y_{e_i+2}, C_{e_i+2})$
    $\boldsymbol{z}_i \leftarrow \begin{cases} (k_{e_i}, k_{e_i+1}, w_{e_i+1}), & \text{if } e_i = 1 \\ (k_{e_i}, k_{e_i+1}, w_{e_i+1}, x_3), & \text{otherwise} \end{cases}$ ▷ response
  end for
  $p := [e, (\boldsymbol{b}_1, \boldsymbol{z}_1), \ldots, (\boldsymbol{b}_t, \boldsymbol{z}_t)]$ ▷ zero-knowledge proof
  return $\sigma \leftarrow \mathtt{Ser}(p)$ ▷ signature (serialized)
end procedure

[3] the addition of *salt* is to mitigate multi-target attacks

Verification

The verifying process (as given by Algorithm 3.4) is composed of similar operations as in the signing process, as some of the values are computed again at this end. This does not additionally increase the computational cost but makes the signature size smaller.

Algorithm 3.4 Verification $\mathsf{Verify}(pk, m, \sigma)$ [CDG+17]

procedure VERIFY(y, p) ▷ $p \leftarrow (\boldsymbol{a}, e, \boldsymbol{z})$
  for $1 \leq i \leq t$ **do** ▷ t - number of iterations
    $[e, \{(\boldsymbol{b}_i, \boldsymbol{z}_i)\}] \leftarrow \mathtt{DeSer}(p)$ ▷ proof/signature (de-serialized)
    $e_i \in [1, 2, 3]$
    ▷ run MPC protocol to recompute explicitly unsent values
    $x_{e_i} \leftarrow \begin{cases} x_3, & \text{if } e_i = 3 \\ G(k_{e_i}), & \text{otherwise} \end{cases}$
    $x_{e_i+1} \leftarrow \begin{cases} x_3, & \text{if } e_i = 2 \\ G(k_{e_i+1}), & \text{otherwise} \end{cases}$
    $w_{e_i} \stackrel{?}{=} \mathtt{Update}(\ldots \mathtt{Update}(x_{e_i}, k_{e_i}, x_{e_i+1}, k_{e_i+1}) \ldots)$ ▷ $w_{()} \equiv \mathtt{View}_{()}$
    $y_{e_i} \stackrel{?}{=} \mathtt{Output}(w_{e_i}),\ y_{e_i+1} \stackrel{?}{=} \mathtt{Output}(w_{e_i+1})$ ▷ $w_{e_i+1} \leftarrow \boldsymbol{z}_i$
    $y_{e_i+2} \stackrel{?}{=} y \oplus y_{e_i} \oplus y_{e_i+1}$ ▷ check if y is indeed sum of y_j
    for $j \in [e_i, e_i + 1]$ **do**
      $[C_j, D_j] \leftarrow [H_C(x_j, k_j, w_j), k_j \| w_j]$ ▷ $(y_{e_i+2}, C_{e_i+2}) \leftarrow \boldsymbol{b}_i$
    end for
    $\boldsymbol{a}'_i \leftarrow (y_1, y_2, y_3, C_1, C_2, C_3)$
  end for
  $\boldsymbol{a}' := \{\boldsymbol{a}'_1, \ldots, \boldsymbol{a}'_t\}$
  $e' \leftarrow H(\boldsymbol{a}')$
  if $e' == e$ **then**
    return ACCEPT
  else
    return REJECT
  end if
end procedure

The *verifier* $\mathcal{V}$ obtains the transcript $(\boldsymbol{a}, e, \boldsymbol{z})$ and simulates the MPC protocol to generate the missing shares (x_e, x_{e+1}), not given in the response $\boldsymbol{z}$. The shares thus generated, are verified whether they satisfy the decomposition function when reconstructed. Once this is verified, the challenge is recomputed using the obtained shares and commitments, and verified if it matches with the challenge computed by the *prover* $\mathcal{P}$. If the multiple steps of verification within the DECOMP(x, k)

succeed, then the signature is accepted as valid, otherwise it is rejected. Accepting the signature as valid also implies that the signature is a valid zero-knowledge proof of the witness known to $\mathcal{P}$, and that $\mathcal{V}$ knows nothing about it except that it is a witness to the statement being proved by $\mathcal{P}$.

3.3 Security

Some important notions necessary to argue the post-quantum security of Picnic and any signature scheme in general are briefly discussed here ([KL20]).

Pseudo Random Generator (PRG):

A *Pseudo Random Generator* is a deterministic polynomial-time algorithm G that, for a given security parameter λ, takes as input a truly random *seed* $s \xleftarrow{\$} \{0,1\}^n$ and produces a pseudo-random output string $x := G(s) \in \{0,1\}^{\lambda \cdot n}$, where $\lambda \cdot n \gg n$. Specifically, G is a PRG if for all PPT *distinguishers* D, the below holds:

$$\big|Pr[D(r) = 1] - Pr[D(G(s)) = 1]\big| \leq \boldsymbol{negl}(n),$$

where $r \xleftarrow{\$} \{0,1\}^{\lambda \cdot n}$. A PRG must also satisfy the property of *unpredictability* where a distinguisher is not able to predict the next output bits of a given PRG.

Pseudo Random Function (PRF):

Let Func_n be a set of all functions mapping n-bit strings to n-bit strings. Let $F \xleftarrow{\$} \mathsf{Func}_n$ be a randomly chosen function from this set. A function $f_k : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ for a randomly chosen *key* k ($k \xleftarrow{\$} \{0,1\}^n$), which is *length-preserving* (i.e. $|f_k| = |k| = |x|$) is *efficient* if there exists a deterministic PPT algorithm that computes $f_k(x)$ given k and x. Such a function f_k is called a *Pseudo Random Function* if for all PPT distinguishers D, the below holds:

$$\big|Pr[D^{f_k(\cdot)}(1^n) = 1] - Pr[D^{F(\cdot)}(1^n) = 1]\big| \leq \boldsymbol{negl}(n).$$

One-Way Function (OWF):

A function $f : \{0,1\}^* \to \{0,1\}^*$ is a *One-Way Function* if there exists a polynomial-time algorithm that computes $f(x)\ \forall\ x$ and for every PPT algorithm $\mathcal{A}$, computing the inverse of f is hard or

$$Pr[\mathcal{A}^{f^{-1}}(n) = 1] \leq \boldsymbol{negl}(n).$$

Security proof in the ROM

A *random oracle* can be defined as a black-box that accepts any arbitrary-length input and provides a fixed-length random output. An adversary can query the random oracle with any input string and obtain a completely random output. Since the oracle models a truly random function, the output is chosen uniformly at random and is independent of the input string. Hence, it is also not possible to predict the output for a different input string other than the ones already queried. This also means that the random oracle is deterministic, that is, every time it is queried with the same input, it produces the same output. In practical applications, cryptographic hash functions are usually modeled as random oracles, also called *idealized* hash functions, to argue their security based on this assumption. Such a model for defining security is called a Random Oracle Model (ROM) [Dam02].

Since the hash functions in Picnic are modeled as random oracles, the security of the key generation process is evaluated in the ROM. Thus, to prove the security of the key generation process, we use the definition of a hard instance generator.

Hard Instance Generator: An algorithm G is a Hard Instance Generator (HIG) for a relation R, if it satisfies the below properties:

1. for negligible function $\varepsilon_1(\cdot)$ and security parameter λ

$$Pr[\ (y,x) \leftarrow G(1^\lambda) : (y,x) \in R\ ] \geq 1 - \varepsilon_1(\lambda)$$

   - the PRG G outputs the values (i.e. keys) (x,y) such that $(y,x) \in R$ with non-negligible or high probability

2. for every PPT algorithm $\mathcal{A}$ and negligible function $\varepsilon_2(\cdot)$

$$Pr[\ (y,x) \leftarrow G(1^\lambda), x' \leftarrow \mathcal{A}(y) : (y,x') \in R\ ] \leq \varepsilon_2(\lambda)$$

   - given that G outputs (x,y), the probability that $\mathcal{A}$ takes input y and produces a pre-image x' such that $(y,x') \in R$ is negligible

The relation between the keys (x,y) is established as a one-way function through the LowMC block cipher, such that $(y,x) \in R \iff y = f_k(x)$, which instantiates a HIG suitable for key generation. Below is the proof of security for the key generation process in ROM.

Let $\{f_k\}$ where $k \in K_\lambda$, be a family of PRFs, and hence $y = f_k(x)$ is efficiently computable $\forall k \in K_\lambda$ and $\forall x \in M_\lambda$. Then $\{f_k\}$ is also a family of OWFs, that is,

$$\begin{aligned} P := Pr[\ & k \xleftarrow{\$} K_\lambda, \\ & k^* \leftarrow \mathcal{A}(1^\lambda, f_k(x)) : f_k(x) = f_{k^*}(x)\ ] \leq \varepsilon(\lambda) \end{aligned} \tag{3.2}$$

In essence, we have to prove that, given a security parameter λ, and a PRF f_k, the probability (P) that an adversary $\mathcal{A}$ can output a pre-image (or key) k^* for that function such that $f_k(x) = f_{k^*}(x)$ is negligible. For this, we consider the probabilities P_1, P_2 and P_3 as stated below, and show that they are all negligible (probability P can be re-written as P_1).

P_1: Probability that $f_k(x) = f_{k^*}(x)$ and $k^* \neq k$

$$P = P_1 := \sum_y \frac{|\texttt{keyset}(y)|}{|K_\lambda|} \cdot t_y \tag{3.3}$$

where (i) $\mathcal{B} := \texttt{keyset}(y)$ is the set of keys such that $\forall\ k \in \mathcal{B}, f_k(r) = y$ for random r (ii) t_y is the probability that given y, $\mathcal{A}$ will output k^* such that $f_{k^*}(r) = y$ (iii) K_λ is the keyspace

P_2: Probability that $f_k(x) = f_{k^*}(x)$ and $k^* = k$

$$P_2 := \frac{1}{|K_\lambda|} \sum_y t_y \tag{3.4}$$

Figure 3.4 shows a distinguisher to prove that the probability P_2 is negligible.

P_3: Probability that when $y \leftarrow \mathcal{M}_\lambda$, adversary $\mathcal{A}$ outputs k^* such that $F_{k^*}(r) = y$

$$P_3 := \frac{1}{|\mathcal{M}_\lambda|} \sum_y t_y \tag{3.5}$$

where (i) F is a truly random function (ii) $\mathcal{M}_\lambda$ is the message space

Since $|\mathcal{M}_\lambda| \geq |\mathcal{K}_\lambda|$, it can be shown that $|P_1 - P_3|$ is negligible. Also, $P_2 \leq P_3$ and therefore, $|P_1 - P_2|$ is negligible. Since P_2 and $|P_1 - P_2|$ are both negligible, P_1 is also negligible [CDG+17]. This proves that the function family $\{f_k\}$ is one-way since it is also pseudorandom. This ensures the security of the key generation process because

$$f_k(x) := Enc(x, k) \tag{3.6}$$

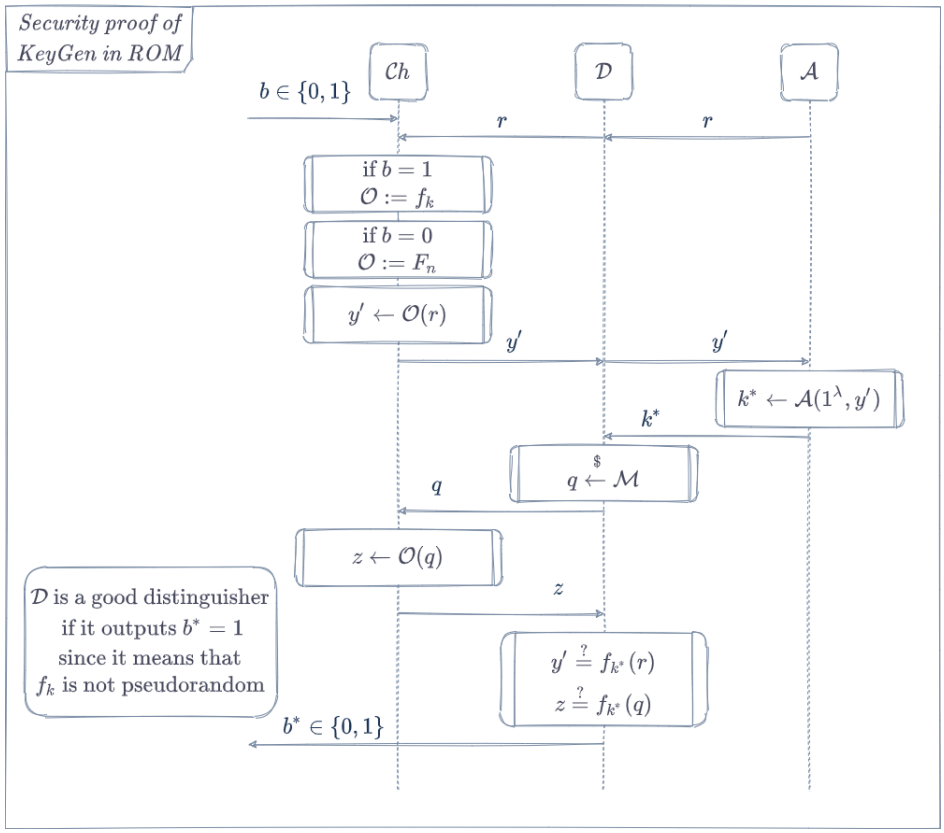


Figure 3.4: Security Proof of the Key Generation in the ROM

Quantum-accessible Random Oracle Model (QROM)

Quantum-accessible Random Oracle Model (QROM) is a random oracle model defined in the quantum setting, that is, it accepts input queries in the form of quantum bits or *qubits*. This means it allows an adversary to query the oracle on a superposition of input values, and the black-box function is computed on this superposition producing an output that is also a superposition of the resultant values. However, unlike the ROM, this property of the QROM does not allow to observe the adversary's inputs without disturbing them (because measuring the input will collapse the state to a single value which may not be the actual input value queried by the adversary). Hence, recording or reprogramming the quantum random oracle is not possible. Due to this, schemes that are deemed secure in the ROM need not be secure in the QROM. Nevertheless, the schemes are not insecure unless the underlying construction itself has been broken [BDF+11, DFG13, DFMS19].

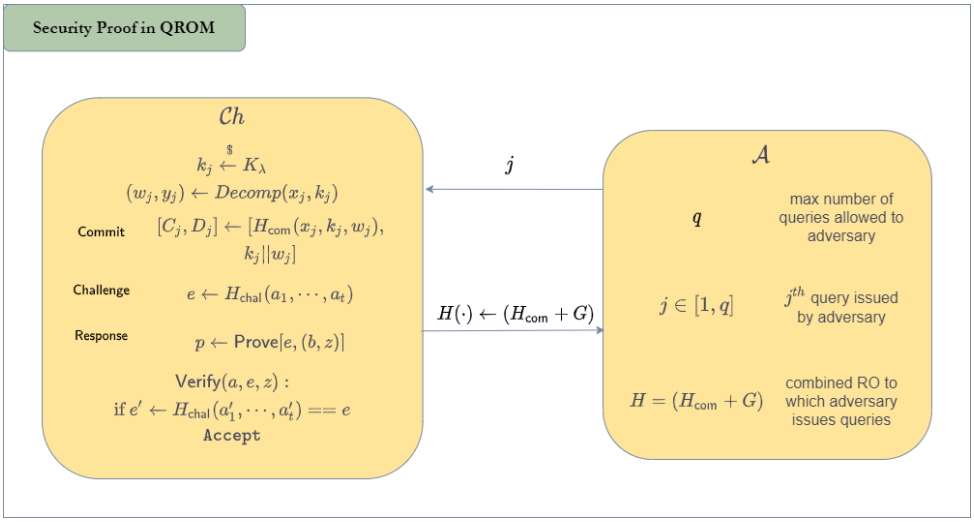


Figure 3.5: Security Proof of the scheme in the QROM

Game 1:	honest protocol run using random oracles
Game 2:	$e^* \xleftarrow{\$} \{1,2,3\}$ $H_{chal}(a,h) := e^*$ $g_j = G(k_j, w_j)$ $h = (g_1, g_2, g_3)$
Game 3:	$H_{com}(k_{e^*}, w_{e^*}) \xleftarrow{\$} \{0,1\}^*$ $G(k_{e^*}, w_{e^*}) \xleftarrow{\$} \{0,1\}^*$
Game 4:	$k_i \xleftarrow{\$} \{0,1\}^\lambda$
Game 5:	replace honest protocol by simulator $SIM(j \neq e_i^*) = \{w\}$
Game 6:	random oracle replaced by random polynomials of $d \geq 2q - 1$

Table 3.1: Security Games for Picnic scheme in the QROM

The Figure 3.5 represents the first part of a collection of security games that are used to give a non-tight security proof of the signature scheme in the QROM. Each of these games are mutually, computationally indistinguishable thereby giving

the necessary proof of security of the scheme. These games are summarized in the Table 3.1.

For the scheme to be secure in the QROM, we need to ensure that an adversary cannot produce a valid forgery on a new message. This is given in terms of the advantage (i.e. probability of succeeding) of the adversary $\mathcal{A}$ against the scheme Π, which must be negligible [DFMS19].

$$Adv[\mathcal{A}, \Pi] := \Big| \; Pr[b = 1 : b \leftarrow \mathsf{Vrfy}_{\Pi}((a, e, z))] \\ - Pr[b = 1 : b \leftarrow \mathsf{Vrfy}_{\mathsf{Sim},\mathcal{A}}((a', e', z'))] \; \Big| \leq \mathbf{negl}(\lambda)$$

The security is proved through a series of games, as described here. (i) ***Game 1:*** *the game is in the real model, where all the hash functions* (H_{com}, H_{chal}, G) *in the scheme* Π *are modeled as quantum random oracles.* Since, cryptographic hash functions are modeled as random oracles, this change makes Game 1 indistinguishable from the actual scheme Π. (ii) ***Game 2:*** *the prover is modified to randomly choose* $e^* := \{e_i^*\} \xleftarrow{\$} \{1, 2, 3\}, i \in [1, t]$ *and the oracle* H_{chal} *is programmed to return the value* e^*. Game 1 and Game 2 are indistinguishable since the method of choosing e does not alter the success probability of the adversary. (iii) ***Game 3:*** *the outputs of* H_{com} *and* G *are replaced by random strings.* The probability that a random j^{th} query made by $\mathcal{A}$ to the oracle H, measures to a value $x' = x$ is negligible, leading to Game 2 and Game 3 being indistinguishable. (iv) ***Game 4:*** *instead of randomly choosing* k_i *as the seed and expanding it via the PRG, the value* k_i *is chosen uniformly at random.* Game 3 and Game 4 are indistinguishable by the pseudo-randomness property of the PRG function. (v) ***Game 5:*** *a PPT simulator* Sim *is now used to generate without a witness, the views (for* $j \neq e^*$*) which will be opened.* The perfect privacy property of the circuit decomposition function ensures that Game 4 and Game 5 are identical. (vi) ***Game 6:*** *the random oracles are replaced by random polynomials of degree* d *greater than the maximum number of queries* q *allowed to the adversary.* Since random polynomials are indistinguishable from a random function, the probability that the adversary $\mathcal{A}$ can produce a valid transcript (a', e', z') accepted by the verifier, such that an extractor E cannot extract a valid witness is negligible. In each of these games, subsequent changes made to the real scheme Π, are proved to be computationally indistinguishable from instances of random functions and/or random values [CDG+17, Unr15]. This proves the security of the scheme in the QROM.

Chapter 4
Sphincs+

A *S*tateless, *P*ractical, *H*ash-based, *I*ncredibly *N*ice *C*ryptographic *S*ignature (SPHINCS+) is described in detail in this chapter. The name is quite attractive and suggests some of the most important features of the scheme. However, before getting into the details of SPHINCS+ or any hash-based signatures in general, it is important to know the significance of *Stateful* and *Stateless* schemes.

All hash-based signatures are built from the primitive of a One-Time Signature (OTS) scheme. OTS is secure, similar to a one-time-pad, as it can be used only once to sign a single message. Any other approaches, like Few-Time Signature (FTS), Many-Time Signature (MTS), Merkle trees and so on, are built on this fundamental block. When used multiple times as in FTS or MTS, it becomes necessary to remember the key-pairs that have already been used to generate valid signatures. Due to this reason, most of the early designs of hash-based signature schemes were stateful, requiring efficient storage management to store the state.

As a workaround to this, and to circumvent the storage constraints, stateless schemes have been developed in recent times. Such schemes employ two methods to achieve statelessness. One possible solution is to utilize a huge-enough keyspace, such that the probability of signing 2 or more messages with the same key is negligible. Another way is to randomize the process of selecting a key-pair to sign a given message, which ensures that, even though the same key-pair is used to sign more than one message, the randomness used with the key-pair ensures perfect secrecy [BDS09].

The Sphincs+ signature scheme is significant because it is stateless and practical enough for applications in the internet. It uses OTS to sign and certify the key-pairs of MTS, which form the nodes of a certification tree of MTS signatures. The certification tree (or *hypertree*) consists of a set of binary hash trees at its bottom layer. This layer of hash trees at the bottom consist of FTS key-pairs that are used to sign the actual message digests. Sphincs+ is able to use the advantages of all the three types of signatures to achieve a good trade-off between security and efficiency,

for a practical hash-based signature scheme.

4.1 Key Concepts

Sphincs+ and any signature scheme in general makes use of a secure cryptographic hash function to produce unforgeable signatures. In this regard, it is important to understand some basic constructions of such hash functions and the mathematics behind them.

Merkle - Damgård Construction

One such significant construction is the Merkle-Damgård [BS17] construction, which is shown in Figure 4.1. It is an algorithm that uses a compression function h iteratively, to compute the hash of long messages. The message is split into L blocks as per the block-size of the compression function h. The message m is padded with a padding block if it is not a multiple of the block-size. Even if the message is a multiple of the block-size, the padding block is added as a dummy block to facilitate proper decoding at the receiver.

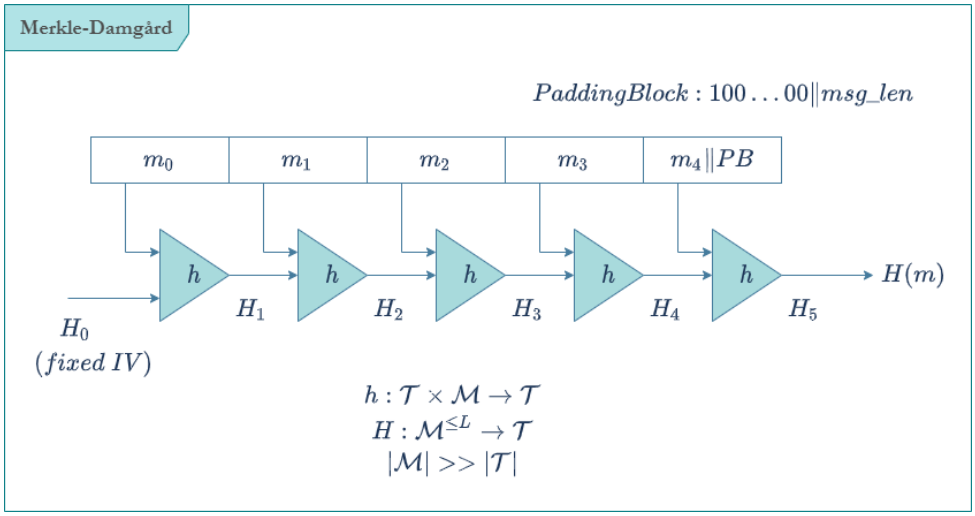


Figure 4.1: Cryptographic Hash functions: Merkle - Damgård Construction

The function takes a fixed *Initialization Vector (IV)* H_0 and the message block m_0, producing the output H_1 for the next stage. H_i are called the *chaining* variables. The compression function maps the arbitrary length input from the message space $\mathcal{M}$ to a fixed-length output in the tag (or digest) space $\mathcal{T}$.

There are different ways in which the compression function h can be instantiated. For instance, Davies-Meyer [BS17] compression function, given by $h(H, m) := E(m, H) \oplus H$ using a block cipher E, is of common use.

Sponge Construction

The Sponge construction [BDPVA07] is another iterated hash function like the Merkle-Damgård. This is shown in Figure 4.2. The construction has two phases, namely, the *Absorbing* and *Squeezing* phase. The system state consists of a block of $b = r + c$ bits initialized to 0, where r is the *bit-rate*, c is the *capacity* and $r \gtrapprox c$. A padding function is used to appropriately extend the input message to be a multiple of the r-bits of the state.

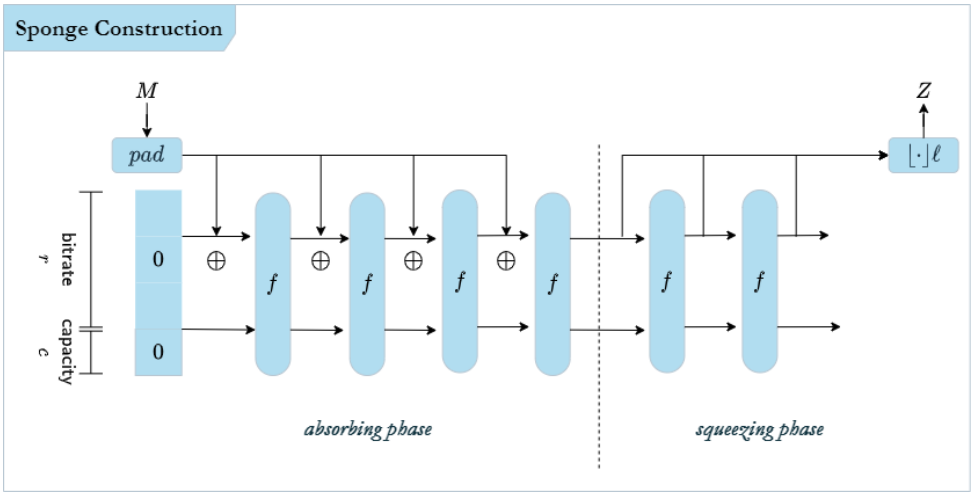


Figure 4.2: Cryptographic Hash functions: Sponge Construction; Source: [BDH+08]

In the absorption phase, the r-bits of the state and successively r-bits of the padded message are XOR-ed and given as input to function f. The capacity c is not directly affected by the input message, which is fed as-is to function f. It is different from the compression function h used in the Merkle-Damgård construction discussed above. The function f is a pseudo-random permutation (or a transformation) applied many times to the system state in multiple rounds. This is done until the complete message has been "*absorbed*" or transformed to an intermediate output block of b-bits.

In the squeezing phase, an output of desired length is chosen from the upper part of r-bits of the state successively until the required length of the digest is obtained.

This makes the Sponge construction an eXtendable Output Function (XOF) as the output is "*squeezed*" out of the system state. The capacity c, though does not play a direct role in the generation of the message hash, it plays a very important role in ensuring the security of the construction. Depending on the distribution of bits for bitrate r and capacity c, in order to avoid internal collisions in the state, the capacity [1] should never take the same value twice (limits to $2^{c/2}$ blocks). Therefore, the sponge construction claims to provide a security of c-bits due to the bound by the birthday paradox.

Merkle Tree

One-time signature schemes are the most fundamental form of digital signatures, whose security is based on secure one-way functions. However, the disadvantage of OTS is that they can be used to sign only a single message, which may not be desired in some applications. The Merkle Binary Hash Tree [BDS09] (shown in Figure 4.3) is an efficient way of extending the OTS to sign more than one message, while balancing the security vs efficiency trade-off. The Merkle Tree is built upon and agnostic to, the one-time signature (e.g. Lamport OTS, Winternitz OTS) and the cryptographic hash function used (e.g. based on Merkle-Damgård or Sponge).

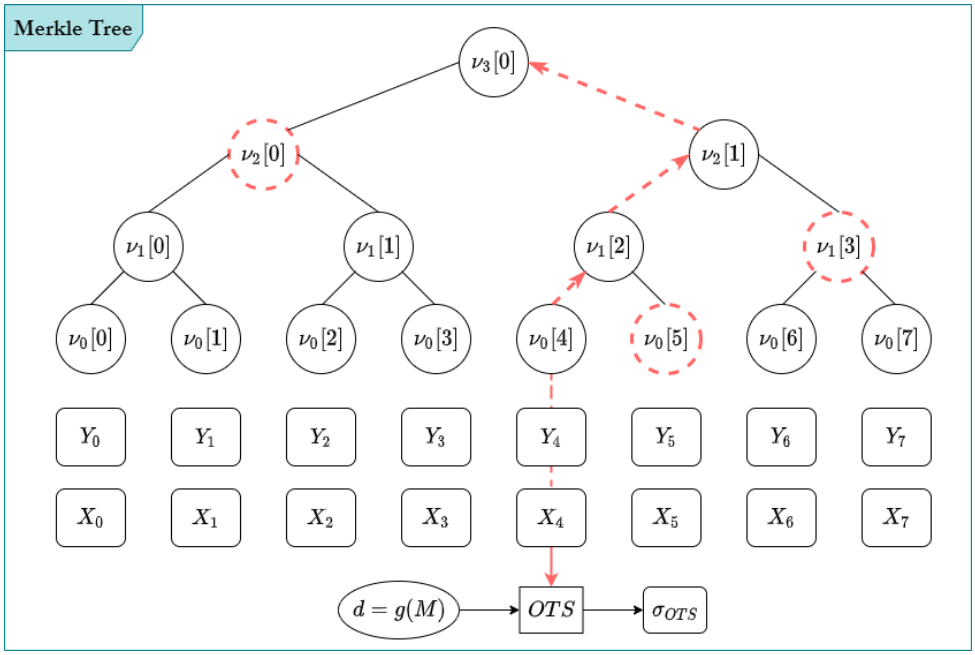


Figure 4.3: Merkle Binary Hash Tree (of height $H = 3$) with signature and authentication path [BDS09]

[1] [Exc14]

Let $f : \{0,1\}^n \to \{0,1\}^n$ be a secure one-way function and $g : \{0,1\}^* \to \{0,1\}^n$ be a secure cryptographic hash function. The secret and public keys of a given OTS, respectively, may be computed as follows:

$$X = \{x_i : x_i \xleftarrow{\$} \{0,1\}^n\} \quad \text{and} \quad Y = \{y_i : y_i \leftarrow f(x_i)\};$$

$$2^H \text{ key-pairs } (X_j, Y_j),\ 0 \le j < 2^H \implies 2^H \text{ one-time signatures}$$

The nodes of the Merkle tree are denoted by $\nu_h[j]$ where $0 \le h \le H$ and $0 \le j < 2^{H-h}$. Specifically, $\nu_0[j] = g(Y_j)$ and recursively, the other nodes are computed as follows:

$$\nu_h[j] = g(\nu_{h-1}[2j] \parallel \nu_{h-1}[2j+1])$$

For example,

$$\nu_1[1] = g(\nu_{1-1}[2 \cdot 1] \parallel \nu_{1-1}[2 \cdot 1 + 1]) \implies \nu_1[1] = g(\nu_0[2] \parallel \nu_0[3])$$

The tree shown has height $H = 3$ and consists of 2^H leaves which are the digests of the OTS public keys Y_j. Each pair of leaves are concatenated together and hashed to obtain the parent node in the tree. This is continued until the entire tree is computed upto the root node of the tree. The root node of the Merkle tree forms the public-key (pk) of the Merkle Tree signature scheme, while the OTS key-pairs form the secret-key (sk).

In the Merkle signature generation, the signer computes the n-bit digest, $d = g(M)$, for the message M. Then he picks up one of the OTS signing key X_{idx} (i.e. one of the leaves of the tree with index $0 \le idx \le 2^H - 1$) to generate the OTS signature σ_{OTS} on the digest.

$$\sigma_{OTS} \leftarrow OTS(X_{idx}, d)$$

Along with this, the Merkle signature also includes the corresponding one-time verification key Y_{idx} and the *Authentication Path* (`AUTH`).

$$A_{idx} = \{a_0, \ldots, a_{H-1}\} \quad \text{where} \quad a_h = \begin{cases} \nu_h[c-1] \text{ , if } \lfloor c \rfloor \equiv 1 \mod 2 \\ \nu_h[c+1] \text{ , if } \lfloor c \rfloor \equiv 0 \mod 2 \end{cases} \text{ and } c = idx/2^h$$

The `AUTH` (A_{idx}) consists of a set of *sibling* nodes that are used to re-construct the tree from the leaf ($g(Y_{idx})$) upto the root node ($\nu_H[0]$) of the tree. These nodes uniquely prove the authenticity of Y_{idx} and are shown by the red dotted nodes in the tree of Figure 4.3.

4.2 Scheme

An overview of the SPHINCS+ [BHK+19] signature scheme is shown in Figure 4.4. It consists of a *hyper-tree*, which is a chain of binary hash trees that is used to certify

the Sphincs+ key pairs. The required parameters for the scheme are given in the table below:

$n \in \mathbb{N}$	security parameter
w	Winternitz parameter
h	height of HyperTree
d	number of layers in the hypertree
k	number of trees in FORS
a	height of each tree in FORS
$t = 2^a$	number of leaves in a FORS tree
m	message digest

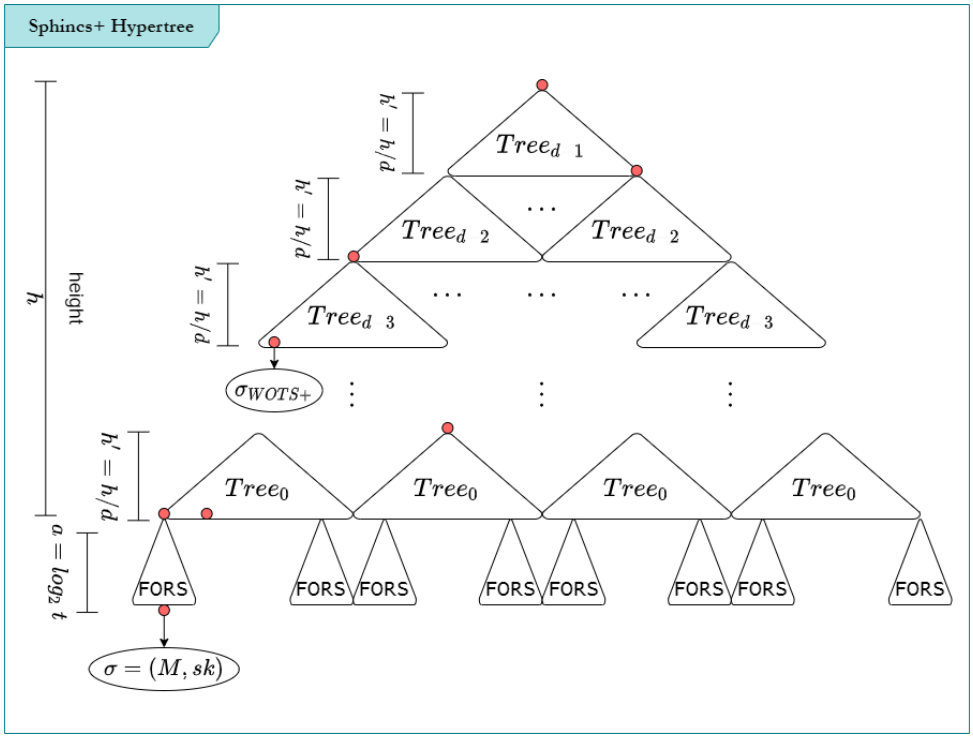


Figure 4.4: SPHINCS+ Hyper-Tree (HT) [BHK+19, ABD+20]

As seen in Figure 4.4, the Hyper-Tree (HT) is a huge N-ary hash tree of total height h with d layers. Each sub-tree in the hypertree has height $h' = h/d$, and it is a simple instantiation of the Merkle Binary hash tree, which uses the Winternitz

One-Time Signature (WOTS+) as the underlying OTS scheme. The lowest layer of the hypertree is used to generate signatures for the Forest of Random Subsets (FORS) key-pairs. The FORS key-pairs again form a set of binary hash trees of height $a = log_2\ t$ and there are k such trees. Here, the hypertree is basically a set of N^d OTS signatures for the MTS key-pairs that in turn sign the FTS key-pairs.

WOTS+

The OTS scheme used in Sphincs+ is the Winternitz One-Time Signature+ (WOTS+). The WOTS+ signature is obtained by using the chaining function f which is applied iteratively many times to the secret key element $\texttt{seed}_{\texttt{sk}}$. The hash chain output is sampled randomly and concatenated to produce the WOTS+ signature, as shown in Figure 4.5.

The WOTS scheme consisting of key generation, signature generation and verification algorithms can be summarized as below:

$$\text{KeyGen}(n, w) : (sk, pk) \leftarrow ((x_i \xleftarrow{\$} \{0,1\}^n), (y_i := f^{2^w-1}(x_i)))$$

$$\text{Sign}(sk, m) : \begin{cases} d = g(m) \implies d' = 0 \| d; & c = \text{checksum}(d) \implies c' = 0 \| c; \\ b = d' \| c' = \{b_{t-1}, \ldots, b_0\}; & W = 2^w - 1 \\ f^{b_{t-1}}(x_{t-1}), \ldots, f^{b_0}(x_0) & \sigma \leftarrow \{f^{b_i}(x_i)\} \end{cases}$$

$$\text{Verify}(pk, m, \sigma) : \begin{cases} v_i = f^{W-b_i}(x_i); & W = 2^w - 1;\ 0 \leq i \leq t-1 \\ \text{if } (\boldsymbol{\nu} := \{v_i\}) == (Y := \{y_i\}) & \text{accept else reject} \end{cases}$$

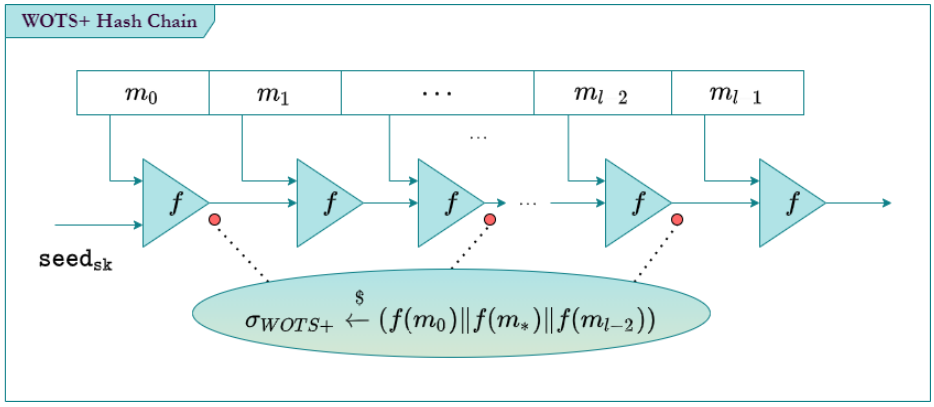


Figure 4.5: WOTS+ Chaining function F [BDS09, ABD+20]

FORS

The Forest of Random Subsets (FORS) is the FTS scheme used by Sphincs+ to instantiate binary hash trees and sign the message digests. There are k binary hash trees that are used to sign a few messages (of the order of tens of messages). The FORS instances are shown in Figure 4.6.

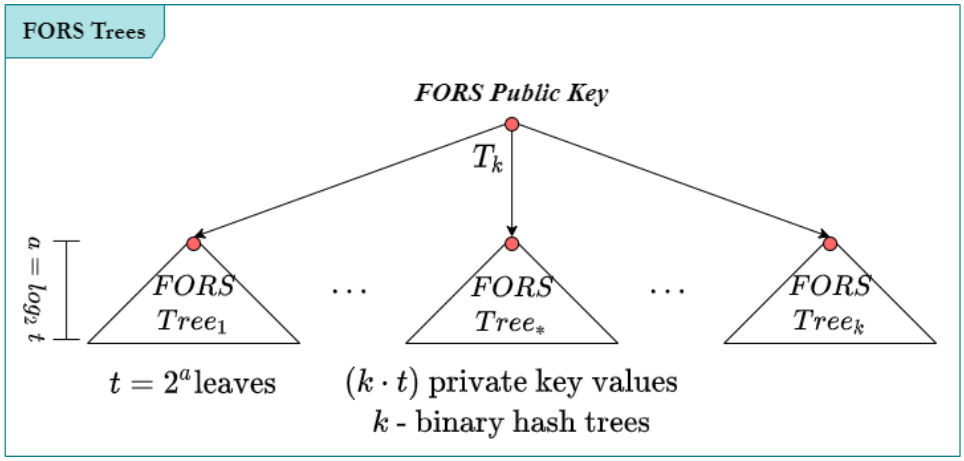


Figure 4.6: FORS Tree Instances [ABD+20]

Key Generation

Key generation (Algorithm 4.1) in Sphincs+ is described below.

Algorithm 4.1 Key Generation $\mathsf{Gen}(1^n)$ [BHK+19]

1: **procedure** KeyGen(n) ▷ n - security parameter
2: $\mathtt{seed}_{\mathtt{sk}} \xleftarrow{\$} \{0,1\}^n$ ▷ private seed of n-bytes
3: $\mathtt{prf}_{\mathtt{sk}} \xleftarrow{\$} \{0,1\}^n$ ▷ PRF key, n-byte string
4: $\mathtt{seed}_{\mathtt{pk}} \xleftarrow{\$} \{0,1\}^n$ ▷ public seed of n-bytes
5: $\mathtt{root}_{\mathtt{pk}} := \mathtt{PKGen}_{\mathtt{HT}}(\mathtt{seed}_{\mathtt{sk}}, \mathtt{seed}_{\mathtt{pk}})$ ▷ Hypertree public key
6: $pk \leftarrow (\mathtt{seed}_{\mathtt{pk}}, \mathtt{root}_{\mathtt{pk}})$
7: $sk \leftarrow (pk, \mathtt{seed}_{\mathtt{sk}}, \mathtt{prf}_{\mathtt{sk}})$
8: **return** (pk, sk)
9: **end procedure**

Given the security parameter n, the public key pk consists of an n-byte public seed ($\mathtt{seed}_{\mathtt{pk}}$) along with the root node ($\mathtt{root}_{\mathtt{pk}}$) of the top most layer sub-tree of the hypertree. The secret key sk consists of a secret seed ($\mathtt{seed}_{\mathtt{sk}}$), chosen uniformly at random, and another "PRF key" that is used later to generate a random value

during computation of the message digest. The secret $\texttt{seed}_{\texttt{sk}}$ is used to generate all the private key (and/or random) elements required for the other building blocks (i.e. WOTS+ and FORS) in the scheme.

Signature Generation

The Sphincs+ signature generation (Algorithm 4.2) has 3 main steps: (1) Random value generation, (2) Message Digest and Context Address generation, and (3) FORS and HT Signature generation with implicit verification.

In the first part, the n-byte string `opt` is initialized to 0 or set to a random value (optionally). If this value is always set to 0, then the signature generation in Sphincs+ will become deterministic and this is not desirable as it can lead to side-channel attacks. If `opt` has high entropy, then the randomization value R also has high entropy; otherwise, with `opt` set to 0, R becomes a pseudorandom value.

In the second part, the randomness R and the public key pk are used to hash the message. The resulting digest is split into 3 parts: msg_digest (md), index of the tree ($\texttt{tree}_{\texttt{idx}}$) and index of the leaf node ($\texttt{leaf}_{\texttt{idx}}$) in the tree.

One of the most important features of Sphincs+ is the notion of *Tweakable Hash Functions*, which ensures that every hash function call is unique/randomized. This is done using the address of the call context for every hash function call. The `ADRS` has a specific structure and is set according to the context in which a call is being made to a cryptographic hash function. The `ADRS` is a 32-byte value with a defined structure as shown below:

$$\texttt{ADRS} := \texttt{Layer\_Address} \,||\, \texttt{Tree\_Address} \,||\, \texttt{TYPE} \,||\, \texttt{Word}_1 \,||\, \texttt{Word}_2 \,||\, \texttt{Word}_3$$

where the `TYPE` field determines what type of context address is to be appended to the `ADRS` variable. The different values of `TYPE` and their corresponding address values [2] are shown in the table below for convenience.

TYPE	$\texttt{Word}_1$	$\texttt{Word}_2$	$\texttt{Word}_3$	Comments
0	Key-pair address	Chain address	Hash address	WOTS+ Hash Address
1	Key-pair address	Zero Padding		WOTS+ Public key Compression Address
2	Zero Padding	Tree Height	Tree Index	Hash Tree Address (hypertree)
3	Key-pair address	Tree Height	Tree Index	FORS Tree Address
4	Key-pair address	Zero Padding		FORS Tree Roots Compression Address

[2] Each $\texttt{Word}_i$ is a 32-bit value or word. The Layer address takes one such 32-bit word and the Tree address takes three such words.

Algorithm 4.2 Signature Generation $\mathsf{Sign}(sk, M)$ [BHK+19]

procedure SIGN(sk, M)
  Step 1
  Start
    $\mathtt{opt}^* \xleftarrow{\$} \{0,1\}^n$ ▷ $^*\mathtt{opt}$ is initialized to 0, but can be a random value
    $R := \mathbf{PRF}_{msg}(\mathtt{prf_{sk}}, \mathtt{opt}, M)$
  End
  Step 2
  Start
    $\mathtt{digest} := \mathbf{H}_{msg}(R, M, pk)$
    $(md, \mathtt{tree_{idx}}, \mathtt{leaf_{idx}}) \leftarrow \mathtt{digest}$
    $\mathtt{ADRS} \leftarrow (0||\mathtt{tree_{idx}}||\mathtt{FORS\_tree}||\mathtt{leaf_{idx}})$ ▷ address type is FORS tree
    $\mathtt{SIG_{fors}} := \mathtt{Sign_{fors}}(md, \mathtt{seed_{sk}}, \mathtt{seed_{pk}}, \mathtt{ADRS})$
  End
  Step 3
  Start
    $\mathtt{PK_{fors}} := \mathtt{PKGen_{FORS}}(\mathtt{SIG_{fors}}, M, \mathtt{seed_{pk}}, \mathtt{ADRS})$
    $\mathtt{ADRS} \leftarrow (0||\mathtt{tree_{idx}}||\mathtt{TREE}||\mathtt{leaf_{idx}})$ ▷ address type is hypertree
    $\mathtt{SIG_{ht}} := \mathtt{Sign_{ht}}(\mathtt{PK_{fors}}, \mathtt{seed_{sk}}, \mathtt{seed_{pk}}, \mathtt{tree_{idx}}, \mathtt{leaf_{idx}})$
  End
  $\mathtt{SIG} \leftarrow (R||\mathtt{SIG_{fors}}||\mathtt{SIG_{ht}})$
  return SIG
end procedure

In the third part, the message digest, the tree and leaf indexes, and the ADRS are used to generate the FORS FTS signature along with the public and secret seed values. This FTS signature consists of the signature on the message and also the authentication path from the leaf node to the root node of the FORS tree. The FTS signature is implicitly verified in the signing process by using the FORS signature to compute the corresponding FTS public key. This FORS public key is then used, with the updated ADRS type pointing to the hypertree, to compute the HT MTS on the FTS key-pairs. The final Sphincs+ signature is then a concatenation of the randomness R, the FORS signature and the HT signature.

Verification

The verification algorithm (Algorithm 4.3) is similar to the signing algorithm.

Algorithm 4.3 Verification Verify$(pk, M, \texttt{SIG})$ [BHK+19]

procedure VERIFY$(pk, M, \texttt{SIG})$
  Step 1
  Start
    $(R||\texttt{SIG}_{\texttt{fors}}||\texttt{SIG}_{\texttt{ht}}) \leftarrow \texttt{SIG}$
    $\texttt{digest} := \mathbf{H}_{msg}(R, M, pk)$
    $(md, \texttt{tree}_{\texttt{idx}}, \texttt{leaf}_{\texttt{idx}}) \leftarrow \texttt{digest}$
  End
  Step 2
  Start
    $\texttt{ADRS} \leftarrow (0||\texttt{tree}_{\texttt{idx}}||\texttt{FORS\_tree}||\texttt{leaf}_{\texttt{idx}})$ ▷ address type is FORS tree
    $\texttt{PK}_{\texttt{fors}} := \texttt{PKGen}_{\texttt{FORS}}(\texttt{SIG}_{\texttt{fors}}, md, \texttt{seed}_{\texttt{pk}}, \texttt{ADRS})$
  End
  Step 3
  Start
    $\texttt{ADRS} \leftarrow (0||\texttt{tree}_{\texttt{idx}}||\texttt{TREE}||\texttt{leaf}_{\texttt{idx}})$ ▷ address type is hypertree
    $\texttt{VER}_{\texttt{ht}} := \texttt{Verify}_{\texttt{ht}}(\texttt{PK}_{\texttt{fors}}, \texttt{SIG}_{\texttt{ht}}, pk, \texttt{tree}_{\texttt{idx}}, \texttt{leaf}_{\texttt{idx}})$
  End
  if $\texttt{VER}_{\texttt{ht}} == 1$ **then**
    return ACCEPT
  else
    return REJECT
  end if
end procedure

The Sphincs+ signature is parsed into its components, i.e. the randomness R, the FORS signature and the HT signature. The message digest and the ADRS values are computed as in the signing process. The FORS signature is implicitly verified by computing the corresponding public key that matches the secret key used in the FORS signature. The HT signature is similarly used to verify the FTS key-pairs through the certification hypertree of Merkle-type signatures.

4.3 Security

The Sphincs+ scheme bases its security on the properties of cryptographic hash functions such as Collision Resistance (CR), One-Wayness or Pre-image Resistance (OW) and Second Pre-image Resistance (SPR) [3] (refer Section 2.3.3). Along with these properties, the designers of Sphincs+ have introduced new notions of security for hash functions, in order to prove the security of their scheme against classical and quantum adversaries. Two of these new security notions are defined below (the other

[3] additionally, variant security notions of OW and SPR in the multi-target setting have been described in Appendix A

security notions of EUF-CMA and PRF have been described in detail in Section 2.2 and Section 3.3).

Let

$$\mathcal{H}_n = \{H_K : \{0,1\}^m \to \{0,1\}^n\} \text{ where } K \in \{0,1\}^\kappa$$

be a family of hash functions. Let

$$\texttt{MAP} : \{0,1\}^n \to \{0,1\}^h \times [0, t-1]^k := \{(I, i, J_i)\}_{i=1}^k$$

be the function that maps n-bit strings to a set of k indexes $\{(I, i, J_i)\}$ and let

$$G = \texttt{MAP} \circ \mathcal{H}_n$$

be defined as a random oracle $\mathcal{O}(\cdot)$ that takes input M_i and outputs K_i and $G(K_i, M_i)$.

Definition 4.1. PQ-DM-SPR ([BHK+19, ABD+20])
Given a set of randomly chosen messages M_i, a set of distinct pre-defined functions H_{K_i} (i.e. p targets as seen by the adversary), the Post-Quantum Distinct-function Multi-target Second Pre-image Resistance (PQ-DM-SPR) is defined as the success probability of an adversary $\mathcal{A}$ (allowed at most q queries) against the scheme Π, to output an index j and another pre-image M', such that M' is not in the set of queried messages $\{M_j\}$, but $H_{K_j}(M')$ belongs to the set of received responses $\{Y_j = H_{K_j}(M_j)\}$ (for the produced index j).

$$Succ_{\mathcal{H}_n,p}^{\text{(PQ-DM-SPR)}}(\mathcal{A}_\Pi) = Pr\Big[(\forall\{K_i\}_1^q \subset (\{0,1\}^\kappa)^q), M_i \xleftarrow{\$} \{0,1\}^m, 0 < i \leq p;$$
$$(j, M') \xleftarrow{\$} \mathcal{A}((K_1, M_1), \ldots, (K_p, M_p))$$
$$: M' \neq M_j \wedge H_{K_j}(M') = H_{K_j}(M_j)\Big]$$

For the scheme to be secure, this success probability (or advantage) must be negligible.

Definition 4.2. PQ-ITSR ([BHK+19, ABD+20])
Given access to a random oracle $\mathcal{O}(\cdot)$ as defined above, the Post-Quantum Interleaved Target Subset Resilience (PQ-ITSR) is defined as the success probability of the adversary $\mathcal{A}$ (allowed at most q queries) against the scheme Π, to find different (K, M) such that it does not belong to the already received responses from the oracle, but $G(K, M)$ belongs to the output set $\{G(K_j, M_j)\}$ of the oracle.

$$Succ_{\mathcal{H},q}^{\text{(PQ-ITSR)}}(\mathcal{A}_\Pi) = Pr\Big[(K, M) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(1^\lambda)$$
$$: G(K, M) \subseteq \bigcup_{j=1}^{q} G(K_j, M_j) \wedge (K, M) \notin \{(K_j, M_j)\}_1^q\Big]$$

Again, this probability must be negligible for the scheme to be secure.

The Theorem 4.3 states the EUF-CMA property of Sphincs+, which essentially defines an upper bound on the success probability (or advantage) of an adversary to produce a valid forgery.

Theorem 4.3. *The Sphincs+ scheme* Π *is (Post-Quantum) Existentially Unforgeable under adaptive Chosen Message Attack (PQ-EU-CMA)[BHK+19] if the below assumptions hold:*

$\mathbf{PRF}$ $\mathbf{PRF}_{msg}$	are post-quantum pseudo-random function families (PQ-PRF)
$\mathbf{PRF}_{BM}$	is modeled as a quantum accessible random oracle (Q-RO)
$\mathbf{H}_{msg}$	is a post-quantum interleaved target subset resilience hash function family (PQ-ITSR)
F, H, T	are post-quantum distinct-function multi-target second pre-image resistant function families (PQ-DM-SPR)
$\forall\ y \in \{\mathtt{IMG}(F_k)\}_{k\in\{0,1\}^n}$ $\exists\ x, x' \in \{0,1\}^n$ $:\ x \neq x' \wedge F_k(x) = F_k(x')$	every element in the image of F has at least two pre-images

specifically, the insecurity function $InSec_{(\mathsf{PQ\text{-}EU\text{-}CMA})}$ *of* Π *is the maximum success probability over all q-query adversaries running in time* $\leq \xi$ *and is bounded by*

$$
\begin{aligned}
InSec_{(\mathsf{PQ\text{-}EU\text{-}CMA})}(\Pi, \xi, q) \leq \\
2 \cdot \Big\{ & InSec_{(\mathsf{PQ\text{-}PRF})}(\mathbf{PRF}, \xi) + InSec_{(\mathsf{PQ\text{-}PRF})}(\mathbf{PRF}_{msg}, \xi) \\
& + InSec_{(\mathsf{PQ\text{-}ITSR})}(\mathbf{H}_{msg}, \xi) + InSec_{(\mathsf{PQ\text{-}DM\text{-}SPR})}(F, \xi) \\
& + InSec_{(\mathsf{PQ\text{-}DM\text{-}SPR})}(H, \xi) + InSec_{(\mathsf{PQ\text{-}DM\text{-}SPR})}(T, \xi) \Big\} \quad (4.1)
\end{aligned}
$$

The insecurity functions $InSec_{(\mathsf{PQ\text{-}*})}$, define the probability in terms of the insecurity of the scheme against an adversary (which is equivalent to the maximum success probability of the adversary), who can break the defined security properties of PRF, PQ-DM-SPR, PQ-ITSR and thus EUF-CMA. The scheme Π is secure, if the success probability or the advantage of the adversary $\mathcal{A}$, as bounded by Eq. (4.1), is negligible.

Game 0:	PQ_EU_CMA experiment in the QROM
Game 1:	**PRF** outputs replaced by truly random strings
Game 2:	Signing Oracle $\mathbf{PRF}_{msg}$ replaced by a truly random function
Game 3:	$\mathcal{A}$ wins if $Ver(SIG, M) \neq \perp$ for FORS secret values in previous SIG responses from SIG Oracle
Game 4:	$\mathcal{A}$ wins if $Ver(SIG, M) \neq \perp$ if SIG contains second pre-image for input to tweakable hash function as part of query response (observed during verification of SIG)

Table 4.1: Security Games for SPHINCS+ scheme in the QROM [BHK+19]

The security proof is given through a series of games (5 games) described in Table 4.1 to establish the security of this scheme. The games are described as follows: (i) ***Game 0:*** this simulates the EUF-CMA game for Sphincs+ in the QROM, where a quantum enabled adversary is allowed access to a classical random oracle (i.e. limited to classical queries). (ii) ***Game 1:*** all the WOTS+ and FORS secret key values (i.e. **PRF** outputs) are replaced by random strings, which implies indistinguishability by the pseudo-randomness property of the **PRF** (else $\mathcal{A}$ can be used to build a distinguisher to break the pseudo-randomness of **PRF**). (iii) ***Game 2:*** the signing oracle $\mathbf{PRF}_{msg}$ used to generate the message digest is replaced by a truly random function. The pseudo-randomness property of the **PRF** again ensures indistinguishability. (iv) ***Game 3:*** the adversary wins the game if he outputs a valid forgery (M, SIG), containing only the secret values in the $\mathtt{SIG}_{\mathtt{FORS}}$ part of SIG, as obtained from previous responses from the signing oracle. (v) ***Game 4:*** the adversary wins the game if he outputs a valid forgery (M, SIG), containing the same secret values as in the honestly generated $\mathtt{SIG}_{\mathtt{FORS}}$ part of SIG, but which was not obtained from previous responses from the signing oracle. These games are mutually indistinguishable by the cryptographic properties of the hash functions used and thus, the success probability of the adversary in each of these games is bound to a negligible value.

Chapter 5
GeMSS

Great Multi-variate Signature Scheme (GeMSS) is a signature scheme based on the concept of solving non-linear multi-variate polynomial systems and belongs to the class of multi-variate public key cryptosystems (MPKC). It is constructed using the *Hidden Field Equation (HFE)* paradigm and comes under the so-called *Big Field* family of multivariate schemes. The HFE paradigm involves masking a central non-linear polynomial ($\mathcal{F}$) with multiple variables, using two linear or affine transformations ($\mathcal{S}$ and $\mathcal{T}$). Specifically, the big field family makes use of a degree n extension field $\mathbb{E}$ of a base finite field $\mathbb{F}_q$ with q elements, to define the secret quadratic map $\mathcal{F}$.

The signature generation requires finding solutions to the system of quadratic equations, by inverting the individual maps $(\mathcal{S}, \mathcal{F}, \mathcal{T})$ and applying them to the hash of the message to produce a signature. The verification process is quite simple and evaluates the public-key (or the composition of the individual maps $\mathcal{P} = \mathcal{T} \circ \mathcal{F} \circ \mathcal{S}$), at the signature point to verify if it satisfies the set of equations, i.e. $\mathsf{Verify}()$ evaluates if $\mathcal{P}(sig) == H(msg)$.

This chapter is structured as follows: (a) Section 5.1 and Section 5.2 describe the scheme and the underlying hard problem of the construction (b) Section 5.3 discusses the security of the scheme and related definitions (c) The additional Section 5.4 describes generic attacks on the SIG schemes discussed so far, including a recent serious attack on GeMSS that indicates that the scheme is probably not secure enough as per the NIST criteria.

5.1 Scheme

The GeMSS scheme is constructed using multi-variate quadratic (MQ) polynomial equations having special algebraic structure. The complexity of solving such a system of quadratic equations is the core idea to argue the post-quantum security of this scheme.

Key Generation

The parameters used in the key generation process are given in the below table.

λ	security parameter
$D = 2^i$, $i \geq 0$	
$D = 2^i + 2^j$,	degree of secret polynomial
$i \neq j \wedge i, j \geq 0$	
K	digest (hash output) length in bits
n	degree of a field extension of $\mathbb{F}_2$
m	number of equations in public key
v	number of vinegar variables
$a = n - m$	number of minus equations
$\kappa > 0$	number of iterations of $\mathsf{Sign}(\cdot)$ and $\mathsf{Verify}(\cdot)$

The key generation process is described in Algorithm 5.1. This process involves generating two linear maps $\boldsymbol{S}$ and $\boldsymbol{T}$, which are used to conceal the central, quadratic secret map $\boldsymbol{F}$, a multi-variate polynomial in $\mathbb{F}_{2^n}$. The structure of the secret polynomial $\boldsymbol{F}$, called the `HFEv-` shape is as shown below:

$$\boldsymbol{F} := \sum_{\substack{0 \leq j < i < n \\ 2^i + 2^j \leq D}} A_{i,j} X^{2^i + 2^j} + \sum_{\substack{0 \leq i < n \\ 2^i \leq D}} \beta_i(v_1, \ldots, v_v) X^{2^i} + \gamma(v_1, \ldots, v_v) \tag{5.1}$$

where (i) $\boldsymbol{v} := (v_1, \ldots, v_v)$ are the *vinegar* variables (ii) $\beta_i(\boldsymbol{v}) : \mathbb{F}_2^v \to \mathbb{F}_{2^n}$ is the set of all linear terms in $\boldsymbol{v}$ variables (iii) $\gamma(\boldsymbol{v}) : \mathbb{F}_2^v \to \mathbb{F}_{2^n}$ is the set of all quadratic terms in $\boldsymbol{v}$ variables.

By fixing the *vinegar* variables, the Eq. (5.1) reduces to a univariate polynomial with `HFE` shape as shown below:

$$\sum_{\substack{0 \leq j < i < n \\ 2^i + 2^j \leq D}} A_{i,j} X^{2^i + 2^j} + \sum_{\substack{0 \leq i < n \\ 2^i \leq D}} B_i X^{2^i} + C \qquad \in \mathbb{F}_{2^n}[X] \tag{5.2}$$

This property of the `HFEv-` polynomial $\mathcal{F}$ is used as a *trapdoor* that allows to easily compute the roots of $\mathcal{F}$ and thus, generate a signature.

Let $(\theta_1, \ldots, \theta_n)$ be a basis of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$. The extension field $\mathbb{E} : \mathbb{F}_{2^n} \cong \mathbb{F}_2[X]/\langle g(X) \rangle$[1] and the isomorphism φ between $\mathbb{F}_{2^n}$ and $\mathbb{F}_2^n$ is defined as

$$\varphi : \mathbb{F}_{2^n} \to \mathbb{F}_2^n; \qquad \mathbb{E} = \sum_{k=1}^{n} e_k \cdot \theta_k \to \varphi(\mathbb{E}) = (e_1, \ldots, e_n)$$

[1] where $g(X)$ is an irreducible polynomial of degree n

The central map $\mathcal{F}$ of HFEv- is defined as a polynomial in the quotient ring $\mathbb{F}_{2^n}[X, x_1, \ldots, x_v]/\langle x_1^2 - x_1, \ldots, x_v^2 - x_v\rangle$, giving the quadratic multivariate map $\bar{\mathcal{F}}$ due to the special structure of $\mathcal{F}$ and the isomorphism φ defined above.

$$\bar{\mathcal{F}} := \varphi^{-1} \circ \mathcal{F} \circ \varphi$$

A set of multi-variate polynomials $\boldsymbol{f} := (f_1, \ldots, f_n)$ derived from $\boldsymbol{F}$ (given by Eq. (5.3)), are called the *components* of $\boldsymbol{F}$ over $\mathbb{F}_2$. The public key $\boldsymbol{p}$ (given by Eq. (5.4)) is derived as the composition of the functions $\boldsymbol{S}$, $\boldsymbol{f}$ and $\boldsymbol{T}$, reduced modulo the field equations $\langle x_j^2 - x_j\rangle$, selecting only the first m of the generated polynomials.

Algorithm 5.1 Key Generation $\mathsf{Gen}(1^\lambda)$ [CFMR+20]

procedure KEYGEN(λ) ▷ λ - security parameter
  $\boldsymbol{S} \xleftarrow{\$} GL_{n+v}(\mathbb{F}_2)$
  $\boldsymbol{T} \xleftarrow{\$} GL_n(\mathbb{F}_2)$
  $\boldsymbol{F} \xleftarrow{\$} \mathbb{F}_{2^n}[X, v_1, \ldots, v_v]$ ▷ HFEv- shape multivariate polynomial of degree D
  $\boldsymbol{f} = (f_1, \ldots, f_n) \in \mathbb{F}_2[x_1, \ldots, x_{n+v}]^n$

$$\boldsymbol{f} \leftarrow \boldsymbol{F}\left(\sum_{k=1}^{n} \theta_k x_k, v_1, \ldots, v_v\right) = \sum_{k=1}^{n} \theta_k f_k \tag{5.3}$$

  ▷ $\theta_k \implies$ basis of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$
  $\boldsymbol{p} = (p_1, \ldots, p_n) \in \mathbb{F}_2[x_1, \ldots, x_{n+v}]^n$

$$\boldsymbol{p} := \left(f_1\Big((x_1, \ldots, x_{n+v})\boldsymbol{S}\Big), \ldots, f_n\Big((x_1, \ldots, x_{n+v})\boldsymbol{S}\Big)\right)\boldsymbol{T} \mod \langle x_1^2 - x_1, \ldots, x_{n+v}^2 - x_{n+v}\rangle \tag{5.4}$$

  ▷ $\mathcal{P} = \boldsymbol{T} \circ \varphi \circ \boldsymbol{F} \circ \varphi^{-1} \circ \boldsymbol{S}$
  $pk \leftarrow \boldsymbol{p} = (p_1, \ldots, p_m) \in \mathbb{F}_2[x_1, \ldots, x_{n+v}]^m$ ▷ the first $m = n - a$ polynomials
  $sk \leftarrow (\boldsymbol{F}, \boldsymbol{S}, \boldsymbol{T})$
  return (pk, sk)
end procedure

Signature Generation

The signature generation is described in Algorithm 5.2. Given a message M to sign and a hash function family $\boldsymbol{H} : \{0,1\}^* \to \{0,1\}^K$, the message digest is computed recursively as $\boldsymbol{d} := \{d_1, \ldots, d_m\} \xleftarrow{\approx} \boldsymbol{H}(M)$. The message digest $\boldsymbol{d}$ is embedded in

the signing process by applying the inverse transformations of the maps in the secret key to generate a signature σ.

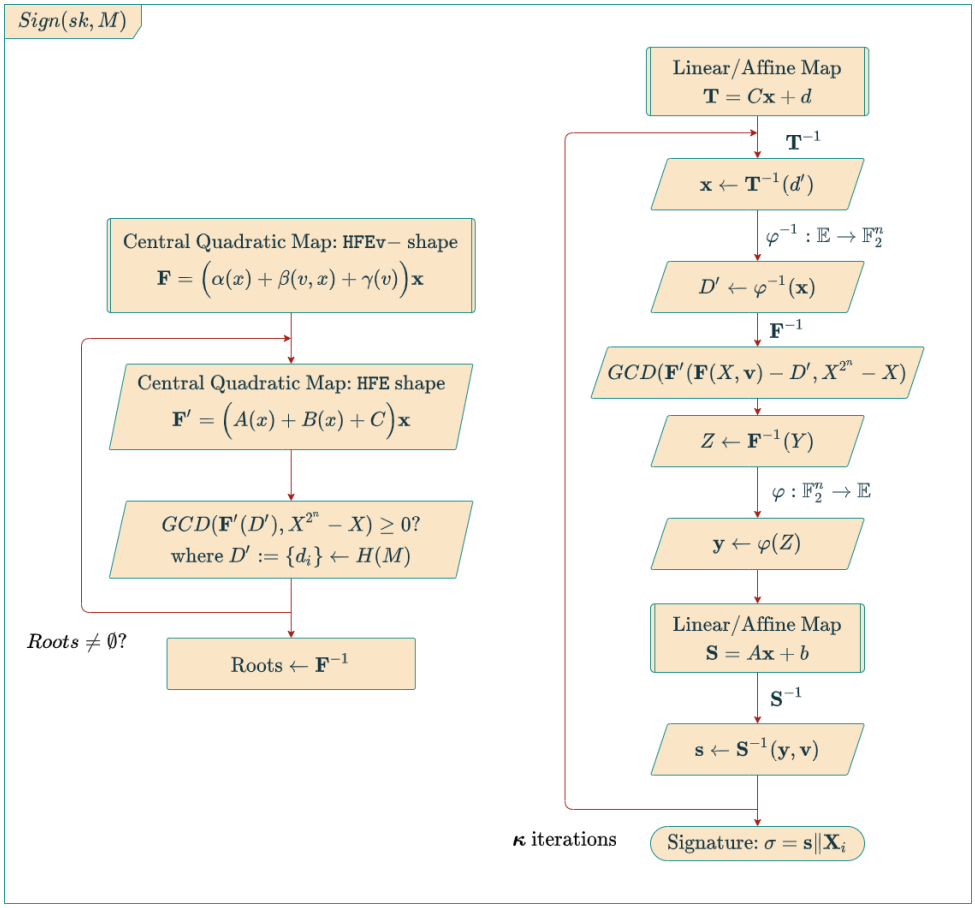


Figure 5.1: Flowchart of the signing process in GeMSS

The 3 main components of $\mathsf{Sign}(\cdot)$ (shown in Fig. (5.1)), as described in Alg. (5.2)) are (i) finding the roots of the polynomials, (ii) to invert the generated affine maps and the central map, and finally (iii) to embed the message (or the message digest) in the signing process as the constant term in the system of equations to find their solution. Inverting the central map to fit the embedded message digest is the trickiest part of the scheme and requires the central map to be surjective, so that every message has a signature. If the system of equations do not have a solution, it means that the roots cannot be found, and thus the message cannot be signed. In this case, the signing process is repeated by sampling new values for the vinegar variables, to

generate a new central map in HFE shape, that can satisfy the surjectivity property.

Algorithm 5.2 Signature Generation $\mathsf{Sign}(sk, m)$ [CFMR$^+$20]

procedure SIGN(sk, m)
  function SIGN(M, sk)
    $\boldsymbol{H} : \{0,1\}^* \to \{0,1\}^K$ ▷ $\boldsymbol{H} \implies$ SHA3
    require: $\Upsilon_0 \leftarrow \mathbf{0} \in \mathbb{F}_2^m$ ▷ where Υ is the system state
    $\boldsymbol{h} \leftarrow \boldsymbol{H}(M)$
    for $1 \leq i \leq \kappa$ **do** ▷ $\kappa = 4$ for GeMSS
      $\Omega_i \leftarrow \boldsymbol{h}\big|_{1,\ldots,m}$ ▷ where Ω stores the first m bits of digest
      $\boldsymbol{s} \leftarrow \text{Inversion}(\Omega_i, sk)$
      $(\Upsilon_i, \boldsymbol{X}_i) \leftarrow \boldsymbol{s}(\Omega_i \oplus \Upsilon_{i-1})$
      $\boldsymbol{h} \leftarrow \boldsymbol{H}(\boldsymbol{h})$
    end for
    $\sigma \leftarrow (\Upsilon_\kappa, \boldsymbol{X}_\kappa, \ldots, \boldsymbol{X}_1)$
  end function
  function INVERSION$(\boldsymbol{d}, sk)$
    require: $\boldsymbol{r} \xleftarrow{\$} \mathbb{F}_2^{n-m}$, $\boldsymbol{d} \in \mathbb{F}_2^m$, $\boldsymbol{d'} \leftarrow (\boldsymbol{d}, \boldsymbol{r})$
    repeat
      $\boldsymbol{v} \xleftarrow{\$} \mathbb{F}_2^v$
      $D' \leftarrow \varphi^{-1}(\boldsymbol{d'} \times \boldsymbol{T}^{-1}) \in \mathbb{F}_{2^n}$
      $F_{D'}(X) \leftarrow F(X, \boldsymbol{v}) - D'$
      $(\cdot, \text{Roots}) \leftarrow \text{FindRoots}(F_{D'})$
    until ROOTS $\neq \emptyset$
    $Z \xleftarrow{\$}$ Roots
    $\boldsymbol{s} \leftarrow (\varphi(Z), \boldsymbol{v}) \times \boldsymbol{S}^{-1} \in \mathbb{F}_2^{n+v}$
    return $\boldsymbol{s}$
  end function
  function FINDROOTS$(F_{D'}(X))$
    $X^{(n)} \leftarrow X^{2^n} - X \mod F_{D'}$
    $G \leftarrow gcd(F_{D'}, X^{(n)})$ ▷ if $G = 1$, then there is a unique solution
    if $G \geq 1$ **then**
      return ROOTS
    else
      return $\emptyset$ ▷ empty or null set
    end if
  end function
  return σ ▷ size: $m + \kappa(n + v)bits$
end procedure

If the central map $\boldsymbol{F}$ has solutions, then it is possible to generate the signature. First, the term $\boldsymbol{D'}$ is computed by applying the inverse transformations φ^{-1} and

$\boldsymbol{T}^{-1}$ on the message digest $\boldsymbol{d}'$, to generate the set of equations $F_{D'}(X)$. The GCD of $(F_{D'}, X^{(n)})$ is computed using the Berlekamp algorithm [VZGG13, BS17], which helps to check if the roots Z exist. If yes, an intermediate value $\boldsymbol{s}$ is computed using Z, by applying the transformations $\varphi(Z)$ and $\boldsymbol{S}^{-1}$. The value $\boldsymbol{s}$ satisfies the equation $\boldsymbol{p}\big|_{[1,m]}(\boldsymbol{s}) = \boldsymbol{d}\big|_{[1,m]}$, but since m is small, $\boldsymbol{s}$ cannot be used as the signature directly, due to the birthday paradox. Therefore, the signing process is iterated κ times, as outlined by the Fiestel-Patarin design [CFMR$^+$20]. Thus, signing is done by computing the hash of the message many times [2] and concatenating those multiple signed versions of the hash outputs along with the final signature output.

Verification

The verification process is described in Algorithm 5.3, which is simpler and faster than the signing process. To verify, one needs to compute the hash of the message and use the signature as the point at which to solve the system of equations embedded in the public key $\mathcal{P}$. If these values are equal, i.e. if $(w \leftarrow H(M)) == (w' \leftarrow \mathcal{P}(\sigma))$, the signature is valid and accepted, else it is rejected.

Algorithm 5.3 Verification $\mathsf{Verify}(pk, m, \sigma)$ [CFMR$^+$20]

procedure VERIFY(pk, m, σ)
  $\boldsymbol{h} \leftarrow \boldsymbol{H}(M)$
  $(\Upsilon_\kappa, \boldsymbol{X}_\kappa, \ldots, \boldsymbol{X}_1) \leftarrow \sigma$
  for $\big(i = 1; i \leq \kappa; i++\big)$ **do**
    $\Omega_i \leftarrow \boldsymbol{h}\Big|_{1,\ldots,m}$ ▷ first m bits of digest
    $\boldsymbol{h} \leftarrow \boldsymbol{H}(\boldsymbol{h})$
  end for
  for $\big(i = (\kappa - 1); i \geq 0; i--\big)$ **do**
    $\Upsilon_i \leftarrow \boldsymbol{p}(\Upsilon_{i+1}, \boldsymbol{X}_{i+1}) \oplus \Omega_{i+1})$ ▷ $\boldsymbol{p}$ is the public key
  end for
  if $\Upsilon_0 == \mathbf{0}$ **then**
    return `ACCEPT` ▷ or `VALID`
  else
    return `REJECT` ▷ or `INVALID`
  end if
end procedure

[2] in this case, κ times, where the hash is computed recursively on the truncated m bits of the message digest

5.2 Hard Problem

Multi-variate Quadratic equations Problem (MQP)

The signature schemes like GeMSS base their security on the difficulty of solving multi-variate quadratic equations. The $\mathcal{MQ}$ problem is stated below:

Problem 5.1. MQP ([Pet17])
Given m quadratic polynomials in n variables, $p_1(\boldsymbol{x}),\ p_2(\boldsymbol{x}),\ \ldots,\ p_m(\boldsymbol{x})$, find a vector $\tilde{\boldsymbol{x}} = \{\tilde{x}_1,\ \ldots,\ \tilde{x}_n\}$ such that, $p_1(\tilde{\boldsymbol{x}}) = \mathbf{0},\ p_2(\tilde{\boldsymbol{x}}) = \mathbf{0},\ \ldots,\ p_m(\tilde{\boldsymbol{x}}) = \mathbf{0}$.

Informally, this problem is about finding the roots of the quadratic polynomials that constitute this system of multivariate equations. This problem, also called the **P*o*lynomial System S*o*lving (PoSSo)**, is believed to be NP-hard for both classical and quantum computers. However, there is no concrete or tight security proof for this assumption that the MQ problem is quantum resistant.

Extended Isomorphism of Polynomials (EIP)

An isomorphism is a mapping that preserves the structure between the two constituent sets. In group theory, a group isomorphism is a function that preserves the structure of the group elements in accordance with the rules of the group operations. Similarly, the isomorphism of polynomials is defined as follows:

Definition 5.2. ([Pet17]) The polynomial systems $\mathcal{G} : \mathbb{F}^n \to \mathbb{F}^m$ and $\mathcal{H} : \mathbb{F}^n \to \mathbb{F}^m$ are called Isomorphic, if and only if, there exist linear or affine maps $\mathcal{L}_1$ and $\mathcal{L}_2$ such that, $\mathcal{H} = \mathcal{L}_1 \circ \mathcal{G} \circ \mathcal{L}_2$.

The central map $\mathcal{F}$ and the public key $\mathcal{P}$ of a multi-variate quadratic cryptosystem are isomorphic since $\mathcal{P} = \mathcal{T} \circ \mathcal{F} \circ \mathcal{S}$.

Problem 5.3. EIP ([Pet17])
Given the public key $\mathcal{P}$ of a multi-variate quadratic system, find linear (affine) maps $\bar{\mathcal{S}}$ and $\bar{\mathcal{T}}$, and invertible quadratic map $\bar{\mathcal{F}}$, such that $\mathcal{P} = \bar{\mathcal{T}} \circ \bar{\mathcal{F}} \circ \bar{\mathcal{S}}$.

The hardness of the EIP problem depends on how the central quadratic map $\mathcal{F}$ is constructed. In general, it is conjectured that the security analysis of MPKCs is harder. This is because of the uncertainty about the complexity of the problem and also much lesser understanding of the multi-variate polynomial systems for larger parameters (i.e. higher number of equations and unknowns).

5.3 Security

Proving the security of MPKCs is an open problem, as there is no security proof or theoretical reduction for the same. This is because, although the MQP and EIP are believed to be NP-complete problems, it does not necessarily translate into post-quantum security. However, practical results have been very close to the theoretical estimates in terms of the security claimed for such schemes. The known results about the provable security of GeMSS are described below.

Theorem 5.4. *The number of iterations $\boldsymbol{\kappa}$ of the Sign(·) and Verify(·) algorithms has to be chosen such that*

$$2^{m \cdot \frac{\boldsymbol{\kappa}}{\boldsymbol{\kappa}+1}} \geq 2^{\lambda}$$

where (i) λ is the security parameter and (ii) m is the number of MQ equations.

This bound on the number of iterations is important to protect the scheme from generic hash collision attacks against the Feistel-Patarin approach of constructing the signature. Another important security property of a signature scheme is the Existential Unforgeability under adaptive Chosen Message Attack (EUF-CMA) defined below for the GeMSS instantiation.

Definition 5.5. ([CFMR+17]) The GeMSS signature scheme is $(\varepsilon, q_h, q_s, t)$-secure if there is no adversary $\mathcal{A}$ running in time t, with q_h queries to the random oracle and q_s queries to the signing oracle, that can produce a valid forgery of (msg, sign) pair with non-negligible probability ε.

The goal from the designer's perspective is to reduce the provable security of GeMSS to the hardness of the inversion problem, i.e. inversion of the public-key. This is the crucial part in the signing process that can assure the unforgeability of the scheme.

Definition 5.6. ([CFMR+17]) The GeMSS KeyGen(·) is (ε, t)-secure if there is no adversary $\mathcal{A}$ running in time t, that given the public-key $\boldsymbol{p}$ and a challenge $\boldsymbol{d}$, can find a pre-image $\boldsymbol{s}$, such that $\boldsymbol{p}(\boldsymbol{s}) = \boldsymbol{d}$ with non-negligible success probability ε.

To achieve this, a simple solution is to add a random salt $\in \{0,1\}^{\ell}$ along with the message and compute the hash of (msg $\|$ salt). This prevents the adversary from computing the inverses parallelly for κ iterations, but makes the signing process costly as this requires iterating the inversion phase as many times as the degree of the secret polynomial. However, this has not been done in GeMSS due to efficiency concerns, because the number of calls to the root-finding function is $\propto D : D \leftarrow deg_{\mathtt{HFEv-}}(\boldsymbol{F})$, where typically, D ranges from $15 \lessapprox D \lessapprox 515$ [3].

[3] for this case, the number of iterations (κ) of Sign(·) is one

5.4 Attacks on the signature schemes

Cryptography has two main branches: *cryptology* and *cryptanalysis*. The former deals with the design and construction of efficient algorithms that provide confidentiality and data integrity. The latter deals with the analysis of constructed cryptosystems to look for loopholes and vulnerabilities that might be exploited by an attacker. Both these branches are very essential to develop strong, robust and secure protocols that safeguard data and communications.

Until now, we have discussed about some signature schemes that are believed to be secure, based on cryptographic hard problems presumably hard to solve both by classical and quantum computers. However, any cryptosystem is secure enough only when it stands the test of time and continuous evaluation by the cryptographic community. In the following sections, two major types of attacks on the three SIG schemes have been presented. It contains a description of the nature of the attacks and their variants as specifically applied to the schemes.

5.4.1 Multi-Target Attacks

Multi-Target Attack (MTA) is the most basic type of attack that is mounted against cryptosystems based on symmetric primitives. This attack has been mounted on Picnic and Sphincs+ as well, as they are symmetric primitive based schemes. Usually, MTA is simple to mount due to efficient *sort-and-match* algorithms, which is used to recover the secret key. However, this has not been the case with Picnic due to its efficient design, requiring much cryptanalysis to be able to identify this possibility [DN19]. For Sphincs+ though, it has been much simpler to mount such attacks due to the frequent use of hash functions, possibly with the same keys [HRS16].

The attack consists in gaining as many targets as possible, say G, for a given cryptosystem. A *target* is essentially the output of the cryptosystem evaluated at different secret inputs (or keys). The *data points* (e.g. signatures) are created by the users and may be associated with the short-term keys (e.g. random numbers) or long-term keys (e.g. secret signing keys). The attacker guesses a key, evaluates the cryptosystem to get an output, G' and compares this with all the targets to see if there is a match. The work required to make the correct guess, is reduced by a factor $\geq G$, due to the birthday paradox [DN19]. The different variants of MTA are summarized at a high level in Table 5.1.

Multi-User Single Target (MUST)	Single-User Multi-Target (SUMT)	Multi-User Multi-Target (MUMT)
$G = U$	$G_i = D_i$ for each user $i \in [1, U]$	$G = D = \sum D_i$
Long term keys are vulnerable	Short term keys of each user U are vulnerable	All short term keys are vulnerable

Table 5.1: G: number of *targets* obtained by attacker; D_i: the set of all *data points* obtained for each user; U: number of users ([DN19])

Picnic

A brief outline of MTA [DN19] on Picnic is given in Algorithm 5.4. The computational complexity of the attack is determined by the number of invocations of the PRG. (e.g. $2^{\lambda-d}$ for the case when $d \leq \lambda/2$). For the security parameter λ, and for the number of allowed MPC protocol runs to the attacker, $D \leq 2^{\lambda/2}$, the attack complexity is $T = 2^{\lambda}/D$. For example, with $\lambda = 128$ and $D = 2^{42}$ ($\approx 2^{35}$ signatures) we would obtain $T = 2^{86}$. However, Picnic has addressed this attack and added countermeasures in later versions by adding a random salt to the seed given as input to the PRG.

Algorithm 5.4 Multi-Target Attack - Picnic [DN19]

Require: $D = 2^d$ and $D' = 2^{\lambda-d}$
1: **for** $i \in [1, D]$ **do**
2: $r \in [1, D]$;
3: $\{b_r\} \leftarrow \texttt{PRG}_u(r)$ ▷ $\{b_r\}|_{2^d}$
4: $R_i \leftarrow (r, \{b_r\})$
5: **for** $k \in [1, D']$ **do** ▷ seed values selected arbitrarily
6: $\{b'_k\} \leftarrow \texttt{PRG}(k)$ ▷ $\{b'_k\}|_{2^{\lambda \cdot d}}$
7: **if** $\{b_r\} == \{b'_k\}$ **then**
8: compute witness x for each matching pair (r, k)
9: **else**
10: repeat from step 5 for different k
11: **end if**
12: **end for**
13: **end for**
14: **return** witness x

Sphincs+

MTA mounted on Sphincs+ is given in Algorithm 5.5. The idea of MTA for hash-based schemes is simpler compared to Picnic. Generally, hash functions are used multiple times in a cryptosystem. For instance, if a hash function with an output of length n-bits is used d times, then the adversary $\mathcal{A}$'s workload is reduced, as he only

needs to find one pre-image out of the d hash outputs. This downgrades the attack complexity to $\mathcal{O}(2^n/d)$ as opposed to the assumed $\mathcal{O}(2^n)$ [HRS16]. Similar to Picnic, Sphincs+ has also addressed MTA by using the concept of "tweakable hash functions" that have a different salt added for every function call based on the context.

Algorithm 5.5 Multi-Target Attack - Sphincs+ [HRS16]

Input: Message $M \xleftarrow{\$} \{0,1\}^*$
Let $\boldsymbol{H} : \{0,1\}^* \rightarrow \{0,1\}^n$
while $1 \leq i \leq d$ **do**
    $\boldsymbol{h}_i \leftarrow \boldsymbol{H}(\mathrm{M})$
end while
for all $\boldsymbol{h}_i : 1 \leq i \leq d$ **do**
    Compute at least one pre-image $m_i \leftarrow \boldsymbol{H}^{-1}(\boldsymbol{h}_i)$
end for
$m' \leftarrow m_k : m_k = \boldsymbol{H}^{-1}(\boldsymbol{h}_k)$ ▷ $m_k \rightarrow$ pre-image from $k^{th} \in [1, d]$ hash output
return m'

5.4.2 Key Recovery Attacks

The attacks on Multi-variate Public Key Cryptosystems (MPKCs) are of two main types: *direct* and *structural* [DP17]. Direct attacks, as the name suggests, aim to solve the public-key polynomial system of equations, $\mathcal{P}(\boldsymbol{z}) = \boldsymbol{w}$ where $\boldsymbol{z} \leftarrow \textsc{Sign}(\boldsymbol{w})$ and $\boldsymbol{w} \leftarrow \boldsymbol{H}(M)$, as an instance of the MQP. Structural attacks exploit the special structure of the central map $\mathcal{F}$ i.e. `HFEv-` or `UOV` shape, to decompose the public key as $\mathcal{P} = \mathcal{T} \circ \mathcal{F} \circ \mathcal{S}$.

Structural attacks have two subcategories: *rank* and *differential* attacks. The MinRank attack is a structural attack that looks for a linear combination of the quadratic forms[4] of the public-key polynomials of low rank. This linear combination corresponds to the central map, and the private key of the MPKC can be recovered by finding such combinations of low rank. The Differential attack looks for symmetries or invariants of the differential, defined as $G(x, y) = P(x + y) - P(x) - P(y) + P(0)$, of the public key, which can help in analyzing the structure and thus, recovering the private key [DP17].

GeMSS

An improved key recovery attack on `HFEv-` schemes, particularly GeMSS, has been published recently by Ding et. al. [Din20]. The core idea of this attack is to find equivalent keys of the public key composition $\mathcal{P}$, such that $\mathcal{P} = \mathcal{T} \circ \mathcal{F} \circ \mathcal{S} = \mathcal{T}' \circ \mathcal{F}' \circ \mathcal{S}'$ where $\mathcal{F}$ and $\mathcal{F}'$ have the same algebraic structure. This allows the adversary to

[4] polynomials in which all the terms are quadratic

use the equivalent maps $\mathcal{T}', \mathcal{F}'$ and $\mathcal{S}'$, to easily forge a valid signature for any message [Din20]. Based on this idea, Theorem 5.7 gives the number of equivalent keys for a given `HFEv-` based MPKC.

Theorem 5.7. *([Din20]) Let $\mathcal{P}$ be the public key of `HFEv-` scheme over finite field $\mathbb{F}_q$ with q elements, v be the number of vinegar variables, a be the number of minus equations and n be the degree of the field extension. Then there exist*

$$nq^{a+2n+vn}(q^n-1)^2\prod_{i=0}^{v-1}(q^v-q^i)\prod_{i=(n-a-1)}^{n-1}(q^n-q^i) \tag{5.5}$$

equivalent keys for the public key $\mathcal{P}$.

For instance, with the parameters $(q, n, v, D, a) = (7, 7, 2, 14, 2)$, the number of equivalent keys is $\approx c \times 7^{46} \times (7^7-1)^2$, where c is a constant.

This key recovery attack on GeMSS is described in the Algorithm 5.6. The attack essentially exploits the min-rank problem and builds on top of the available algorithms to solve it and recover the secret key. The MinRank problem can be solved using one of the below methods (a) linear algebra search (b) Kipnis-Shamir modeling (c) minors modeling (d) support minors modeling [Din20].

Let $(\theta_1, \theta_2, \ldots, \theta_n)$ be the basis vector of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$. Let $M = (\theta_1^{q^k}, \theta_2^{q^k}, \ldots, \theta_n^{q^k})$ where $0 \leq k < n$, be the matrix with Frobenius powers [5] of the basis elements. The matrix $\widetilde{M}$ is defined as

$$\widetilde{M} = \begin{pmatrix} M & 0 \\ 0 & I_v \end{pmatrix}; \quad I_v \to v \times v \text{ Identity matrix} \tag{5.6}$$

The matrix F^{*0} represents the central map $\mathcal{F}$ as shown below:

$$F^{*0} = \begin{pmatrix} F_0 & 0 & F_1 \\ 0 & 0 & 0 \\ F_1^t & 0 & F_2 \end{pmatrix} = \begin{pmatrix} \alpha_{i,j} & 0 & \gamma_{i,j} \\ 0 & 0 & 0 \\ \gamma_{j,i} & 0 & \delta_{i,j} \end{pmatrix} \tag{5.7}$$

[5] a Frobenius map ($x \mapsto x^p$) is one which maps every element x to its p^{th} power, where p is prime. [CFMR+20]

Algorithm 5.6 Key Recovery Attack - GeMSS [Din20]

function RECOVER LINEAR MAP $\mathcal{S}$
Input: HFEv- parameters (q, n, v, D, a), $d = \lceil log_q D \rceil$, quadratic forms $(P_0, \ldots, P_{n-a-1})$ of public-key polynomials, matrix $\widetilde{M}$
Output: Equivalent Linear Transformation $\mathcal{S}'$

set $\boldsymbol{b}_i = (1, u_1, \ldots, u_{n+v-1})P_i$, $0 \leq i < n - a$
▷ $\boldsymbol{b}_i$ forms the rows of matrix Z
$Z \leftarrow \{\boldsymbol{b}_i\} : Z \in \mathcal{M}_{(n-a)\times(n+v)}(\mathbb{F}_{q^n})$ and $\texttt{rank}(Z) \leq d$

$(u_0, u_1, \ldots, u_{n+v-1}) \leftarrow \texttt{MinRank}(Z)$
▷ using minors modeling or support minors modeling

set $U = \begin{pmatrix} \{u_j^{q^k}\} \\ \{r_{ij}\} \end{pmatrix}$ where $\begin{cases} 0 \leq i < v \\ 0 \leq k < n \\ 0 \leq j < n + v \end{cases}$ ▷ $r_{ij} \xleftarrow{\$} \mathbb{F}_q : U$ is invertible

compute $\mathcal{S}' = (\widetilde{M}U)^{-1}$
return $\mathcal{S}'$
end function

function RECOVER LINEAR AND CENTRAL MAPS $\mathcal{T}$ AND $\mathcal{F}$
Input: HFEv- parameters (q, n, v, D, a), $d = \lceil log_q D \rceil$, quadratic forms $(P_0, \ldots, P_{n-a-1})$ of public-key polynomials, Frobenius matrix M, recovered linear map $\mathcal{S}'$
Output: Equivalent Maps $\mathcal{F}'$ and $\mathcal{T}'$
Let $\{w_k\}$ be unknowns with $w_0 = 1$, solve the set of equations

$$\sum_{k=0}^{n-a-1} w_k U P_k U^t = \sum_{i=1}^{a} l_i F^{*i} + F^{*0} \text{ and find solution } \{w'_k\} \tag{1}$$

Solve bilinear equations and univariate polynomial equations obtained from (1) to recover F^{*0}
Obtain equivalent central map

$$F' = (X^{q^k}, x_i)F^{*0}(X^{q^k}, x_i)^t \text{ and compute } F^{*k}, 1 \leq k < n$$

Solve a set of linear equations to recover $\mathcal{T}'$

$$(P_0, \ldots, P_{n-a}) = (SMF^{*0}M^tS^t, \ldots, SMF^{*n-1}M^tS^t)M^{-1}T \tag{2}$$

return $\mathcal{F}', \mathcal{T}'$
end function

First step: Recover the linear map $\mathcal{S}$

A matrix $U = \widetilde{M}^{-1}\mathcal{S}^{-1}$ with $n+v$ rows is selected such that, each of the n rows, are set as the i^{th} row vectors $\boldsymbol{b}_i$, that make up the rows of the low-rank matrix Z. Then, the matrix Z is solved for the MinRank problem (using *minors modeling* or *support minors modeling* methods) to obtain the solution vector $\boldsymbol{u}$. The last v rows of the matrix U are chosen randomly from $\mathbb{F}_q$ such that U is invertible. Thus, an equivalent map $\mathcal{S}' := U^{-1}\widetilde{M}^{-1}$ is recovered.

Second step: Recover the central map $\mathcal{F}$

For this, a set of $d(n-d-a)$ linear equations in $(n-a-1)$ variables w_i, are solved to obtain the solution vector $\boldsymbol{w}$ given by Eq. (1). From this, a set of $(d+a)(n+v)$ bilinear equations and $\binom{v+1}{2}$ univariate polynomials (each with q^d solutions)

$$\begin{bmatrix}\alpha_{l_i} & \gamma_{l_i}\end{bmatrix} = \mathbf{0}; \qquad \sum \lambda_{ij}\delta_{ij}^{q^d} + \eta_{ij} := 0 \tag{3}$$

are solved to obtain $\alpha_{ij}, \gamma_{ij}, \delta_{ij}$ values. These values are used to compute the matrix F^{*0} which in turn can be used to recover the equivalent central map $\mathcal{F}'$.

Final step: Recover the other linear map $\mathcal{T}$

After computing F^{*0}, F^{*k} is also computed for $k \in [1, n-1]$. Using these values, and given that $U^{-1} = \mathcal{S}M$, the Eq. (2) can be re-written as

$$P_k := \left(U^{-1}F^{*k}(U^{-1})^t\right)M^{-1}\mathcal{T} \tag{2a}$$

which produces $(n-a)$ linear system of equations in n variables, that can be solved to obtain the equivalent map $\mathcal{T}'$. Thus, having recovered valid equivalent secret keys $\mathcal{S}', \mathcal{F}', \mathcal{T}'$, it becomes easy for the adversary to forge signatures on any messages of his choice.

This attack mounted recently (Nov 2020) on GeMSS, exploits the structure of the underlying `HFEv-` construction, where the authors of [Din20] state that it would be difficult to build secure signature schemes from the `HFEv-` design. They claim that the attack complexity bounded exponentially by the parameter D, makes it hard to choose an optimal value for D such that the signature scheme is secure, yet efficient. Due to this serious key recovery attack, the road ahead seems to be closed for GeMSS in the NIST standardization process.

Chapter 6
NIST Finalists

This chapter briefly discusses the other three digital signature schemes in the third round of the NIST Standardization process, namely Crystals-Dilithium, Falcon and Rainbow. Crystals-Dilithium and Falcon are lattice-based signatures, while Rainbow is a multi-variate quadratic polynomial based signature (similar to GeMSS discussed in Chapter 5).

Although any given post-quantum signature scheme can generally be categorized into one of the cryptographic families such as lattice-based, MPKC-based, hash-based or others, they most often employ concepts and techniques from each other, that are proven to enhance performance and/or security. For instance, Dilithium uses the Σ-protocol and the zero-knowledge framework like Picnic, Falcon uses polynomials to instantiate the key-pair and the tree in Falcon is similar to the Merkle-trees in hash-based signatures.

It might appear confusing to the reader that, in this thesis, the alternate candidates have been analyzed in more detail than the finalists. This is due to the main reason that, the aim of this thesis was to study and understand the signature schemes built on simpler mathematical assumptions or cryptographic problems. In this regard, the simple and innovative construction of Picnic seemed very attractive along with the hash-based scheme Sphincs+. Later on, the progression of the NIST rounds also influenced the direction and path of this thesis. Specifically, after the results of the third round were announced, the added curiosity about why these two schemes did not make it to the spot among the finalists became the drive to explore the alternate candidates in more detail. Nevertheless, a basic understanding of the finalist schemes has been obtained, and presented in this chapter, for adequate comparison with the alternate candidates.

6.1 Crystals-Dilithium

CRYSTALS-DILITHIUM is a signature scheme, part of the **Cry**ptographic **S**uit**e** for **A**lgebraic **L**attice**s** [DKL+18]. It uses the *Fiat-Shamir heuristic with Aborts* over *module lattices*, and thus has the framework of a zero-knowledge protocol. The novelty of the scheme lies in the technique used to reduce the sizes of the public key (pk) and signature (σ), where the low-order bits in pk are omitted and compensated with minimal additional information ("*hint*") in σ.

The Fiat-Shamir with Aborts [Lyu09] is a method by which a lattice-based signature scheme can be constructed from an identification (ID) scheme. The Fiat-Shamir heuristic is the same as described in Section 3.1, with a slight difference. The ID scheme which uses a Σ-protocol is converted to a non-interactive signature scheme using the Fiat-Shamir transform. However, it is essential that the response "masks" the witness appropriately to ensure indistinguishability. For this reason, the response computed by the prover is checked to be lying in a much larger range than the ones of witness and challenge, or in other words, if the response is in some group G^m. If not, then the protocol is *aborted* and restarted from the point of selecting and committing to a random value. This is the idea behind the Fiat-Shamir with Aborts technique. This is known to be one of the efficient ways of constructing lattice-based signature schemes with small component sizes.

6.1.1 Scheme

Key Generation

The key generation algorithm (6.1) is given below. The public key $(\mathbf{A}, \boldsymbol{t})$ consists of a matrix $\mathbf{A}_{k\times\ell}$ with polynomial entries defined over the ring $R_q = \mathbb{Z}_q[X]/(X^n + 1)$, where $q = 2^{23} - 2^{13} + 1$ is prime and $n = 256$. It also consists of a vector $\boldsymbol{t}$ defined as a linear combination of secret key vectors (s_1, s_2), whose coefficients are obtained through uniform sampling. The vector $\boldsymbol{t}$ is split into high-order and low-order bits, and only the high-order bits are stored in the public key, which reduces its size by more than 50%. KeyGen also generates additional random seeds $(\mathtt{t_r}, K)$ that are used later during the signing process. The function `CRH` is a collision-resistant hash function instantiated with SHAKE-256.

Algorithm 6.1 Key Generation $\mathsf{Gen}(1^\lambda)$ [DKL+18]

1: **procedure** KeyGen(λ) ▷ λ - security parameter
$\delta \leftarrow \{0,1\}^{256}$ ▷ secret random seed to generate randomness for further use
2: **require:** $(\rho, \varrho, K) := H(\delta) \in \{0,1\}^{256\times 3}$ ▷ random seeds used to generate key-pair
$(s_1, s_2) := H(\varrho) \in S_\eta^\ell \times S_\eta^k$ ▷ $H \rightarrow$ SHAKE-256 $\equiv$ CRH (hash function)
3: $\mathbf{A} := \mathtt{Expand}(\rho) \in R_q^{k\times\ell}$
4: $(\boldsymbol{t}_1, \boldsymbol{t}_0) \leftarrow \boldsymbol{t} := \mathbf{A}s_1 + s_2$ ▷ $\boldsymbol{t}_1 \rightarrow$ high-order bits and $\boldsymbol{t}_0 \rightarrow$ low-order bits
5: $\mathtt{t_r} := \mathtt{CRH}(\rho \parallel \boldsymbol{t}_1) \in \{0,1\}^{384}$
6: $pk \leftarrow (\rho, \boldsymbol{t}_1)$ ▷ public key
7: $sk \leftarrow (\rho, \boldsymbol{t}_0, K, \mathtt{t_r}, s_1, s_2)$ ▷ private/secret key
8: **return** (pk, sk)
9: **end procedure**

Signature Generation

The signing algorithm (6.2) is essentially a *Zero-Knowledge proof* of the secret key. It generates a polynomial vector $\boldsymbol{y}$ using the deterministic or randomized seed ρ', which masks the secret key s_1. The value $\tilde{c}$ is computed as the hash of the message concatenated with the high-order bits $\boldsymbol{w}_1$ of $\boldsymbol{w}$ (given by step 7). The signer then computes the *challenge* c by giving $\tilde{c}$ as the input to an XOF. The initial signature is given by $\boldsymbol{z} = \boldsymbol{y} + cs_1$. In order to ensure that this does not leak information about the secret key, *rejection sampling* is used to determine the coefficients of $\boldsymbol{z}$. If the coefficients of $\boldsymbol{z}$ are larger than $\gamma_1 - \beta$ or the coefficients of $\mathbf{A}\boldsymbol{z} - c\boldsymbol{t}$ are larger than $\gamma_1 - \beta$, then the signing procedure is restarted [1]. This is essentially the "Fiat-Shamir with Aborts" method. These two conditions are necessary for the security and correctness of the zero-knowledge signing algorithm. Since $|pk|$ is reduced by $\sim$50%, the signature includes a *hint* vector $\boldsymbol{h}$ that is used to recompute the entire public key, and thus verify the signature. ($|\sigma| +\approx 100$ bytes)

[1] this may require from 4 to 7 repetitions

Algorithm 6.2 Signature Generation $\mathsf{Sign}(sk, m)$ [DKL+18]

procedure SIGN(sk, M)
  require: $\mu := \mathtt{CRH}(\mathtt{t_r} \parallel M) \in \{0,1\}^{384}$ ▷ randomized message digest
  or: $\rho' := \mathtt{CRH}(K \parallel \mu) \in \{0,1\}^{384}$ ▷ deterministic signing
  or: $\rho' \xleftarrow{\$} \{0,1\}^{384}$ ▷ randomized signing
  $\mathbf{A} := \mathtt{Expand}(\rho) \in R_q^{k \times \ell}$
  $\kappa = 0$, $(\boldsymbol{z}, \boldsymbol{h}) = \perp$
  while $(\boldsymbol{z}, \boldsymbol{h}) = \perp$ **do**
    $\boldsymbol{y} := \mathtt{Expand}(\rho', \kappa) \in S_{\gamma_1}^{\ell}$
    $(\boldsymbol{w}_1, \boldsymbol{w}_0) \leftarrow \boldsymbol{w} := \mathbf{A}\boldsymbol{y}$ ▷ $\boldsymbol{w}_1 \leftarrow High(\boldsymbol{w})$
    $\tilde{c} := H(\mu \parallel \boldsymbol{w}_1) \in \{0,1\}^{256}$
    $c \leftarrow XOF(\tilde{c})$ ▷ eXtendable Output Function
    $\boldsymbol{z} := \boldsymbol{y} + cs_1$
    $(\boldsymbol{r}_1, \boldsymbol{r}_0) \leftarrow \boldsymbol{r} := \boldsymbol{w} - cs_2$ ▷ $\boldsymbol{r}_0 \leftarrow Low(\boldsymbol{w} - cs_2)$
    if $\|\boldsymbol{z}\| \geq (\gamma_1 - \beta)$ **or** $\|\boldsymbol{r}_0\| \geq (\gamma_2 - \beta)$ **then** ▷ Fiat-Shamir with Aborts
      $(\boldsymbol{z}, \boldsymbol{h}) = \perp$
    else
      $\boldsymbol{h} := \mathtt{MakeHint}(-c\boldsymbol{t}_0, \boldsymbol{w} - cs_2 + c\boldsymbol{t}_0, 2\gamma_2)$ ▷ construct hint about pk for sig
      if $\|c\boldsymbol{t}_0\| \geq \gamma_2$ **or** $(\#1's \in \boldsymbol{h}) > \omega$ **then**
        $(\boldsymbol{z}, \boldsymbol{h}) = \perp$
      end if
    end if
    $\kappa := \kappa + \ell$
  end while
  return $\sigma \leftarrow (\boldsymbol{z}, \boldsymbol{h}, \tilde{c})$
end procedure

Verification

Since the signing algorithm is zero-knowledge, the verification algorithm (6.3) verifies the zero-knowledge and correctness properties of the signature. The verifier computes $\boldsymbol{w}_1'$ from the hint $\boldsymbol{h}$ and $\mathbf{A}\boldsymbol{z} - c\boldsymbol{t}$, checks if $\tilde{c}$ is indeed the hash of the message and $\boldsymbol{w}_1'$, same as the challenge. Along with verifying that the coefficients of $\boldsymbol{z}$ are within the limit $\gamma_1 - \beta$, the verification checks if $High(\mathbf{A}\boldsymbol{z} - c\boldsymbol{t}) == High(\mathbf{A}\boldsymbol{y}) \iff High(\mathbf{A}\boldsymbol{y}) == High(\mathbf{A}\boldsymbol{y} - cs_2)$ (this proves the correctness property). This is easily seen as

$$\begin{aligned} \mathbf{A}\boldsymbol{z} - c\boldsymbol{t} &= \mathbf{A}\boldsymbol{y} - cs_2 \\ \mathbf{A}(\boldsymbol{z} - \boldsymbol{y}) &= c\,(\boldsymbol{t} - s_2) \\ \mathbf{A}(cs_1) &= c\,(\mathbf{A}s_1) \end{aligned}$$

And also, $High(\boldsymbol{w} - cs_2, 2\gamma_2) = High(\boldsymbol{w} - cs_2 + cs_2, 2\gamma_2) = High(\boldsymbol{w}, 2\gamma_2) = \boldsymbol{w}_1$, thus $\boldsymbol{w}_1' == \boldsymbol{w}_1$ and the verifier accepts the signature.

Algorithm 6.3 Verification $\mathsf{Verify}(pk, m, \sigma)$ [DKL+18]

procedure VERIFY(pk, M, σ)
  $\mu := \mathtt{CRH}(\mathtt{CRH}(\rho \parallel \boldsymbol{t}_1) \parallel M) \in \{0,1\}^{384}$
  $\mathbf{A} := \mathtt{Expand}(\rho) \in R_q^{k \times \ell}$
  $c \leftarrow XOF(\tilde{c})$
  $\boldsymbol{w}_1' := \mathtt{UseHint}(\boldsymbol{h}, \mathbf{A}\boldsymbol{z} - c\boldsymbol{t}_1(\cdot 2^d), 2\gamma_2)$ ▷ $d \approx 13$
  $\nu := [\|\boldsymbol{z}\| < (\gamma_1 - \beta)] \ \wedge \ [\tilde{c} := H(\mu \parallel \boldsymbol{w}_1')] \ \wedge \ [(\#1's \in \boldsymbol{h}) \leq \omega]$
  if $\nu == \mathtt{TRUE}$ **then**
    return `ACCEPT` ▷ or `VALID`
  else
    return `REJECT` ▷ or `INVALID`
  end if
end procedure

6.1.2 Security

Dilithium is constructed using module lattices, and the security is based on the Module-LWE (MLWE), Module-SIS (MSIS) and its variant Self-Target MSIS problems. A basic description of the SIS problem and the LWE (Ring-LWE) problem is given Section 2.9.

The MLWE problem is whether $(\mathbf{A}, \boldsymbol{t} := \mathbf{A}s_1 + s_2)$ can be distinguished from $(\mathbf{A}, \boldsymbol{u})$ where $\boldsymbol{u}$ is uniformly random. The MSIS problem is to find a vector $\boldsymbol{z}'$ with small coefficients such that $\mathbf{A}\boldsymbol{z}' = 0$. The Self-Target MSIS problem is to find a vector $\begin{bmatrix} \boldsymbol{z} & c & \boldsymbol{v} \end{bmatrix}^T$ with small coefficients and message digest μ such that ([DKL+18])

$$H\left(\mu \parallel [\mathbf{A} \mid \boldsymbol{t} \mid I] \cdot \begin{bmatrix} \boldsymbol{z} \\ c \\ \boldsymbol{v} \end{bmatrix}\right) = c, \text{ where } I \rightarrow \text{Identity matrix}$$

The above problems are believed to be hard as there are no known efficient algorithms to solve them, neither classical nor quantum. Some of the common types of attacks on Dilithium are *algebraic* attacks, *dense sub-lattice* attacks, specialized $\ell_\infty - SIS$ attacks and so on. The scheme is proven secure in the ROM, but does not have a tight reduction in the QROM. However, recent research [LZ19] has shown that the *special soundness* and the *collapsing* properties of the Dilithium Σ-protocol can be used to prove that the scheme is secure in the QROM. The special-soundness property in the quantum setting implies that, given a pair of valid transcripts (a, c, r)

and (a, c', r') such that $c \neq c'$, a quantum *extractor* can extract the witness x with probability 1. The collapsing property is when a quantum adversary cannot detect if the superposition of many valid responses for a given commitment-challenge pair has been measured or not [LZ19].

6.2 Falcon

FALCON stands for **Fa**st-Fourier **L**attice-based **Co**mpact signatures over **N**TRU [FHK+18]. It is built using the *Hash-and-Sign* paradigm, where the design makes use of the *GPV Framework* [GPV08] instantiated on NTRU Lattices. A creative concept is the use of *Falcon Trees* to generate the key-pair, while a *Gaussian Rejection Sampler* is used to generate the components of the signature.

The Gentry-Peikert-Vaikuntanathan (GPV) framework describes a secure method for constructing lattice-based signatures. It can be summarized as below (adapted from [FHK+18, GPV08, Kle00]):

- Public key: full-rank $\mathbf{A}_{n\times m} \rightarrow$ generating q-ary lattice Λ
  (i.e. $x \in \Lambda \iff x \pmod q \in \Lambda$)
- Private key: $\mathbf{B}_{m\times m} \rightarrow$ generating lattice $\Lambda_q^\perp$
  (orthogonal to $\Lambda \pmod q$) such that $\mathbf{B} \cdot \mathbf{A}^t = \mathbf{0}$
- Sign/Verify: msg m, short $\mathbf{s} \in \mathbb{Z}_q^m$ such that $\mathbf{s}\mathbf{A}^t = H(m)$
  where $H \rightarrow$ hash function
- Valid `SIG`:
  - compute pre-image (not necessarily short) $\mathbf{c}_0 \in \mathbb{Z}_q^m : \mathbf{c}_0\mathbf{A}^t = H(m)$
  - use $\mathbf{B}$ to compute $\mathbf{v} \in \Lambda_q^\perp$ close to $\mathbf{c}_0$
  - compute $\mathbf{v}$ using randomized variant of *nearest plane algorithm*
  - $\mathbf{s}$ is a valid signature $\iff \mathbf{s} = \mathbf{c}_0 - \mathbf{v}$ is short
  - such that $\mathbf{s}\mathbf{A}^t = \mathbf{c}_0\mathbf{A}^t - \mathbf{v}\mathbf{A}^t = \mathbf{c} - \mathbf{0} = H(m)$

6.2.1 Scheme

Key Generation

Falcon uses a *cyclotomic* (which is monic and irreducible) polynomial ϕ, a modulus q, and real $\beta > 0$ for key generation, as described in Algorithm 6.4. For a positive (prime) integer n, a cyclotomic polynomial is of the form

$$\Phi_n(x) = 1 + x + x^2 + \cdots + x^{n-1} = \sum_{k=0}^{n-1} x^k = (x - \omega_1)(x - \omega_1)\ldots(x - \omega_s)$$

where $\omega_i \rightarrow$ primitive n^{th} roots of unity [Por15].

Algorithm 6.4 Key Generation $\mathsf{Gen}(1^\lambda)$ [FHK+18]

procedure KEYGEN(λ) ▷ λ - security parameter
  require: monic polynomial $\phi \in \mathbb{Z}[x]$, modulus $q = 2^n$

  $f, g, F, G \leftarrow \mathtt{NTRUGen}(\phi, q)$
  ▷ finding polynomials that satisfy NTRU eq.
  function NTRUGEN(ϕ, q)
    repeat
      $f \xleftarrow{\$} \mathbb{Z}[x]/(\phi)$
      $g \xleftarrow{\$} \mathbb{Z}[x]/(\phi)$
    until ($\mathtt{IsInv}(f, \phi) == \mathtt{TRUE}$) ▷ check that f is invertible mod q
    $F, G \leftarrow \mathtt{NTRUSolve}(f, g) : fG - gF = 1 \mod \phi$
    ▷ computing F, G that satisfy NTRU eq.
    return (f, g, F, G)
  end function
  $\mathbf{B} := \begin{bmatrix} g & -f \\ G & -F \end{bmatrix}$
  $\hat{\mathbf{B}} \leftarrow \mathtt{FFT}(\mathbf{B})$ ▷ FFT → Fast Fourier Transform
  $\mathbf{G} \leftarrow \hat{\mathbf{B}} \times \hat{\mathbf{B}}^*$
  $\mathtt{T_F} \leftarrow \mathtt{ffLDL}^*(\mathbf{G})$ ▷ computing the Falcon (LDL*) Tree
  $\sigma \leftarrow 1.55\sqrt{q}$ ▷ Std. Dev. for binary case $\phi := x^n + 1$
  for all $\mathtt{leaf} \in \mathtt{T_F}$ **do**
    $\mathtt{leaf}.value \leftarrow \frac{\sigma}{\sqrt{\mathtt{leaf}.value}}$ ▷ Normalization of Falcon Tree leaves
  end for
  $h \leftarrow gf^{-1} \mod q$
  $pk \leftarrow h$ ▷ public key
  $sk \leftarrow (\hat{\mathbf{B}}, \mathtt{T_F})$ ▷ private/secret key
  return (pk, sk)
end procedure

The private key consists of four polynomials (f, g, F, G) that satisfy the *NTRU Equation*, given by

$$fG - gF = 1 \mod \phi \tag{6.1}$$

The public key is a polynomial $h = gf^{-1} \mod (\phi, q)$. The public and private keys also form the basis for a "$2n$"-dimensional lattice Λ,

$$\mathbf{A} := \begin{bmatrix} -h & I_n \\ qI_n & O_n \end{bmatrix}, \quad \mathbf{B} := \begin{bmatrix} g & -f \\ G & -F \end{bmatrix}$$

where I_n - identity matrix and O_n - zero matrix, both of dimension n.

KeyGen has two main steps: (1) *Solving the NTRU equation* and (2) *Computing the Falcon Tree*. For the former (1), a "*Tower-of-Fields*" method is used to map the NTRU equation from $\mathbb{Z}[x]/(\phi)$, to smaller rings, and finally down to $\mathbb{Z}$. The NTRU equation is solved in $\mathbb{Z}$ and eventually lifted up the *Tower-of-Fields* upto $\mathbb{Z}[x]/(\phi)$. For the latter (2), a "LDL^**-Decomposition*" is done on the matrix $\mathbf{G} := \hat{\mathbf{B}} \times \hat{\mathbf{B}}^* = \mathbf{LDL}^*$, where $\mathbf{L}$ - lower triangular matrix with unity diagonal, $\mathbf{D}$ - diagonal matrix, and $\mathbf{L}^*$ - complex conjugate transpose of $\mathbf{L}$. The non-trivial terms of $\mathbf{L}$ are stored as the root of a Falcon Tree ($\mathtt{T}_{\boldsymbol{F}}$), followed by splitting of the matrix $\mathbf{D}$ into sub-matrices $\mathbf{G}_i$, that in turn are converted into sub-trees in $\mathtt{T}_{\boldsymbol{F}}$ through recursive LDL^*-decomposition.

Signature Generation

Signature generation (Alg. (6.5)) uses a *Trapdoor Sampler*, i.e. *Fast Fourier Sampling*, that outputs (rejection) samples as per a discrete Gaussian distribution, which form the coefficients of $\boldsymbol{z}$ used to generate a short vector $\boldsymbol{s}$. The message to be signed is concatenated with a random salt and hashed to a polynomial c. A pre-image $\boldsymbol{t}$ of c, given to the FFT Sampler, outputs two polynomials (s_1, s_2) such that $s_1 + s_2 h = c \mod q$. The signature sig is the pair (r, s).

Algorithm 6.5 Signature Generation $\mathsf{Sign}(sk, m)$ [FHK+18]

procedure SIGN(sk, m, β)
  $\mathtt{r} \xleftarrow{\$} \{0,1\}^{320}$ ▷ r → random salt
  $c \leftarrow \mathbf{H}(\mathtt{r} \| m)$
  $\boldsymbol{t} \leftarrow (\mathtt{FFT}(c), \mathtt{FFT}(0)) \cdot \hat{\mathbf{B}}^{-1}$
  do
    $\boldsymbol{z} \leftarrow \mathtt{ffSampling}(\boldsymbol{t}, \mathtt{T}_{\boldsymbol{F}})$ ▷ *Fast Fourier Sampling*
    $\boldsymbol{s} := (\boldsymbol{t} - \boldsymbol{z})\hat{\mathbf{B}}$
  while $\|\boldsymbol{s}\| > \beta$
  $(s_1, s_2) \leftarrow \mathtt{InvFFT}(\boldsymbol{s})$
  $\mathtt{s} \leftarrow \mathtt{Compress}(s_2)$ ▷ encoding format
  return $\mathtt{sig} \leftarrow (\mathtt{r}, \mathtt{s})$
end procedure

Verification

The verification algorithm (6.6) is given below. It computes the hash of the message concatenated with randomness r, and checks if $c \stackrel{?}{=} H(m \| \mathtt{r})$. Next, given $\boldsymbol{s}$, it checks if the relation $s_1 + s_2 h = c \pmod q$ holds, provided that the signature $\boldsymbol{s}$ is short. If these two conditions are satisfied, then the signature is accepted as valid, otherwise rejected.

Algorithm 6.6 Verification $\mathsf{Verify}(pk, m, \sigma)$ [FHK+18]

```
1: procedure VERIFY(pk, m, sig, β)
2:     c ← H(r‖m)                          ▷ along with inputs q and n
3:     s_2 ← Decompress(s)                 ▷ decoding the signature format
4:     s_1 ← (c − s_2 h)  mod q
5:     if ‖(s_1, s_2)‖ ≤ β then
6:         return ACCEPT                   ▷ or VALID
7:     else
8:         return REJECT                   ▷ or INVALID
9:     end if
10: end procedure
```

6.2.2 Security

Falcon is based on the hard problem of *SIS over NTRU Lattices*. The SIS problem in lattices has been described in Section 2.9. For the NTRU equation (Eq. (6.1)), computing the polynomials F and G is believed to be hard in general, and there are currently no known algorithms that can efficiently solve these two problems. Some of the common types of attacks on Falcon are *combinatorial* attacks, *hybrid* attacks, *high rank sub-lattice* attacks and *algebraic* attacks. A recent one, *BEARZ* attack [MHS+19] is a fault attack, that exploits the implementation vulnerabilities of Falcon. The Basis Extraction by Aborting Recursion or Zeroing (BEARZ) attack introduces faults in the signing process and exploits the recursion in the Gaussian Sampler or sets the corresponding coefficients to zero, to effectively recover the secret signing key.

Falcon is built using the GPV framework, which has been proven to be secure both in the ROM [GPV08] and QROM [BDF+11]. However, due to the use of the Gaussian Trapdoor Sampler, the scheme is vulnerable to fault and side-channel attacks, which is the case for all schemes based on Gaussian sampling. Despite this, the mathematical complexity and non-straightforward implementation of the Falcon design has made it very difficult to mount such attacks. In terms of performance and security, Falcon is still a good competitor to other lattice-based signature schemes.

6.3 Rainbow

The Rainbow signature scheme is a Multi-variate Public Key Cryptosystem (MPKC), based on the UOV construction. The UOV design is inspired by the inherent unmixing nature of oil and vinegar, extending it to construct non-linear polynomials with $oil(o)$ and $vinegar(v)$ variables such that the number of vinegar variables is larger than the oil variables.

The public key in Rainbow consists of a composition of maps $\mathcal{S}, \mathcal{F}$ and $\mathcal{T}$, which results in a system of multi-variable quadratic polynomial equations to be solved. The private key is the set of the individual maps $(\mathcal{S}, \mathcal{F}, \mathcal{T})$ that allow to easily invert the public key $\mathcal{P}$ and solve the system of equations to generate a signature.

6.3.1 Scheme

Key Generation

The Rainbow scheme can be defined as a multi-layered variant of the UOV paradigm. It consists of u layers defined over a finite field $\mathbb{F}_q$ with q elements. The sets $V_i = \{1, \ldots, v_i\}$ and $O_i = \{v_i, \ldots, v_{i+1}\}, 1 \leq i \leq u$, are defined where the integers v_i $(0 < v_1 < v_2 < \cdots < v_u < v_{u+1})$, specify the set of multi-variate polynomials for each of the u layers. The key generation algorithm (6.7) is given below.

Algorithm 6.7 Key Generation $\mathsf{Gen}(1^\lambda)$ [DS05]

procedure KEYGEN(λ) ▷ λ - security parameter
  ensure: $m \leftarrow o_1 + o_2$ and $n \leftarrow m + v_1$
  $c_{\mathcal{S}} \xleftarrow{\$} \mathbb{F}^m$
  $c_{\mathcal{T}} \xleftarrow{\$} \mathbb{F}^n$
  repeat
    $M_{\mathcal{S}} \leftarrow \mathtt{Mat}_{m \times m} \in \mathbb{F}_q$ ▷ $\mathtt{Mat}_{() \times ()} \rightarrow$ matrix
    $M_{\mathcal{T}} \leftarrow \mathtt{Mat}_{n \times n} \in \mathbb{F}_q$
  until $(\mathtt{IsInv}(M_{\mathcal{S}}) == \mathtt{TRUE}) \wedge (\mathtt{IsInv}(M_{\mathcal{T}}) == \mathtt{TRUE})$
  ▷ $\mathtt{Aff}() \rightarrow$ affine transformation
  $\mathcal{S} \leftarrow \mathtt{Aff}(M_{\mathcal{S}}, c_{\mathcal{S}}) := M_{\mathcal{S}} \cdot x + c_{\mathcal{S}}$
  $\mathcal{S}^{-1} \leftarrow M_{\mathcal{S}}^{-1}$
  $\mathcal{T} \leftarrow \mathtt{Aff}(M_{\mathcal{T}}, c_{\mathcal{T}}) := M_{\mathcal{T}} \cdot x + c_{\mathcal{T}}$
  $\mathcal{T}^{-1} \leftarrow M_{\mathcal{T}}^{-1}$
  $\mathcal{F} \leftarrow \mathtt{RnbwMap}(v_1, o_1, o_2) \in \mathbb{F}_q$ ▷ refer Eq. (6.2)
  $\mathcal{P} \leftarrow \mathcal{S} \circ \mathcal{F} \circ \mathcal{T}$ ▷ $\mathcal{P} : \mathbb{F}^n \rightarrow \mathbb{F}^m$
  $pk \leftarrow \mathcal{P}$ ▷ public key
  $sk \leftarrow ((\mathcal{S}^{-1}, c_{\mathcal{S}}), \mathcal{F}, (\mathcal{T}^{-1}, c_{\mathcal{T}}))$ ▷ private/secret key
  return (pk, sk)
end procedure

The public key $\mathcal{P}$ is composed of a central, quadratic map $\mathcal{F}$, which consists of m quadratic polynomials in n variables, of the form

$$f^{(k)}(x_1, \ldots, x_n) = \sum_{\substack{i,j \in V_\ell \\ i \leq j}} \alpha_{ij}^{(k)} x_i x_j + \sum_{\substack{i \in V_\ell \\ j \in O_\ell}} \beta_{ij}^{(k)} x_i x_j + \sum_{i \in V_\ell \cup O_\ell} \gamma_i^{(k)} x_i + \delta^{(k)} \qquad (6.2)$$

where $k \in [v_1 + 1, n]$ and there exists one $\ell \in [1, u] : k \in O_\ell$ [2]. The polynomial given by Eq. (6.2) does not contain purely quadratic terms in the Oil variables and this *"unmixed"* polynomial allows one to find solutions easily. Two invertible affine maps $\mathcal{S}$ and $\mathcal{T}$ are used to scramble the UOV structure of the central map, thus, making the system hard to solve.

Signature Generation

To generate the signature (Alg. (6.8)), we require $m \leq n$ (*surjective*), so that, every message has a signature. Algorithm 6.8 illustrates the feature of the Rainbow scheme, where similar to the seven colours in a natural rainbow, each of the u layers of multi-variable polynomials are successively reduced to a linear system by selecting (or fixing) random values for the vinegar variables.

The signature is computed on the message digest concatenated with a random salt to achieve EUF-CMA security (see Section 2.2). The `Gauss` function returns a boolean value `t` indicating whether the reduced system of linear equations, obtained by guessing random values for the vinegar variables, is solvable or not. If yes, it returns a random solution of the reduced linear system. If not, the process is repeated again by guessing new values for the vinegar variables in F^{v_1} (step 11).

[2] V_ℓ and O_ℓ are vinegar and oil sub-spaces respectively

Algorithm 6.8 Signature Generation $\mathsf{Sign}(sk, m)$ [DS05]

procedure SIGN($(\mathcal{S}, \mathcal{F}, \mathcal{T}), d$)
  require: $\mathtt{r} \xleftarrow{\$} \{0,1\}^{\ell}$ ▷ $\mathtt{r} \rightarrow$ random salt
  $\boldsymbol{h} \leftarrow \mathbf{H}(\mathbf{H}(d) \| r)$ ▷ $\mathbf{H} : \{0,1\}^{*} \rightarrow \mathbb{F}^{m}$
  $\boldsymbol{x} \leftarrow \mathcal{S}^{-1} \cdot (\boldsymbol{h} - c_{\mathcal{S}})$ ▷ $\mathcal{S} : \mathbb{F}^{m} \rightarrow \mathbb{F}^{m}$
  $\boldsymbol{y} \leftarrow \mathtt{InvF}(\mathcal{F}, \boldsymbol{x}) := \mathcal{F}^{-1}(\boldsymbol{x})$ ▷ $\mathcal{F} : \mathbb{F}^{n} \rightarrow \mathbb{F}^{n}$
  function INVF($\mathcal{F}, \boldsymbol{x}$)
    input: $\boldsymbol{x} \in \mathbb{F}^{m}$ and `RnbwMap` $\mathcal{F}$
    output: $\boldsymbol{y} \in \mathbb{F}^{n}$, such that $\mathcal{F}(\boldsymbol{y}) = \boldsymbol{x}$
    let: $\hat{y}_1 \implies \{y_1, \ldots, y_{v_1}\}$
    let: $\hat{y}_2 \implies \{y_{v_1+1}, \ldots, y_{v_2}\}$
    let: $\hat{y}_n \implies \{y_{v_2+1}, \ldots, y_n\}$
    repeat
      $\hat{y}_1 \xleftarrow{\$} \mathbb{F}$
      $\{\hat{f}^{(v_1+1)}, \ldots, \hat{f}^{(n)}\} \leftarrow f^{(v_1+1)}(\hat{y}_1), \ldots, f^{(n)}(\hat{y}_1)$ ▷ reducing $|o_1|$ polynomials using $\{\hat{y}_1\}$ variables
      $\mathtt{t}, \hat{y}_2 \leftarrow \mathtt{Gauss}(\hat{f}^{(v_1+1)} = x_{v_1+1}, \ldots, \hat{f}^{(v_2)} = x_{v_2})$
      if $\mathtt{t} == \mathtt{TRUE}$ **then**
        $\{\hat{f}^{(v_2+1)}, \ldots, \hat{f}^{(n)}\} \leftarrow \hat{f}^{(v_2+1)}(\hat{y}_2), \ldots, \hat{f}^{(n)}(\hat{y}_2)$ ▷ reducing $|o_2|$ polynomials using $\{\hat{y}_2\}$ variables
        $\mathtt{t}, \hat{y}_n \leftarrow \mathtt{Gauss}(\hat{f}^{(v_2+1)} = x_{v_2+1}, \ldots, \hat{f}^{(n)} = x_n)$
      end if
    until $\mathtt{t} == \mathtt{TRUE}$
    $\boldsymbol{y} \leftarrow (y_1, \ldots, y_n)$
    return $\boldsymbol{y}$
  end function
  $\boldsymbol{z} \leftarrow \mathcal{T}^{-1} \cdot (\boldsymbol{y} - c_{\mathcal{T}})$ ▷ $\mathcal{T} : \mathbb{F}^{n} \rightarrow \mathbb{F}^{n}$
  return $\sigma \leftarrow (\boldsymbol{z}, r)$
end procedure

Verification

The verification (Alg. (6.9)) is straightforward. The hash of the message is computed as $\boldsymbol{h}$, and the public key $\mathcal{P}$ i.e. the system of multi-variable quadratic equations, is applied to the signature $\boldsymbol{z}$ to get the solution $\boldsymbol{h}'$. If the two values are equal, then the signature is accepted to be valid, otherwise discarded.

Algorithm 6.9 Verification $\mathsf{Verify}(pk, m, \sigma)$ [DS05]

```
1: procedure VERIFY(P, d, σ)
2:     h ← H(H(d)‖r)                          ▷ H : {0,1}* → F^m
3:     h' ← P(z)
4:     if h' == h then
5:         return ACCEPT                       ▷ or VALID
6:     else
7:         return REJECT                       ▷ or INVALID
8:     end if
9: end procedure
```

6.3.2 Security

Since Rainbow is an MPKC, the security of the scheme relies on the same hard problems as GeMSS (Ch. 5). Specifically, the security is based on the assumed hardness of the MQ problem and the EI problem (Section 5.2). These two problems are roughly based on finding the roots of the polynomials and finding equivalent maps for the public key respectively. Some of the common attacks on the UOV construction and Rainbow are *OV* attack, *min-rank* attack, *Rainbow Band Separation (RBS)* attack and the most recent *intersection* attack and *rectangular min-rank* attack [Beu20].

The goal of RBS attack is to find linear transformations $\mathcal{S}$ and $\mathcal{T}$, that can reduce the public key to the Rainbow form of polynomials (Eq. (6.2)). The idea is that this system of polynomials is not a random quadratic polynomial system, but has two groups of variables X and Y in which it is *bilinear* (i.e. linear with respect to both variables). This makes it much easier to solve than a completely random system of quadratic polynomials. The Intersection attack is essentially a key-recovery attack that can be seen as a generalization of the RBS attack. The goal of this attack is to find a vector $\boldsymbol{x}$ in the intersection $\bigcap_{i\in[1,k]} L_i O_2$ (i.e. in the oil subspace $\mathcal{O}$ of the structure where $\mathcal{P}(\boldsymbol{o}) = 0, \forall \boldsymbol{o} \in \mathcal{O}$), that allows one to find a unique solution to the system ([Beu20]). However, the best-known attack complexity is mostly exponential in the input, and thus requires a marginal tweak of the Rainbow parameters in order to meet the desired security level.

Chapter 7
Methodology

The most important contribution of this thesis is the study and comparison, both qualitatively and quantitatively, of the various claimed quantum-safe digital signatures explored in the previous chapters. In this direction, this chapter addresses the tools and environment used, and the methods chosen to perform a quantitative analysis and comparison of the signatures.

The structure of this chapter is as follows: (a) Section 7.1 and Section 7.2 describe the tools and platforms used for the quantitative analysis (b) Section 7.3 and Section 7.4 discusses the cryptographic protocol TLS v1.3 and evaluations in Liboqs (c) Section 7.5 describes the additional integration of GeMSS into Liboqs.

7.1 Open Quantum Safe Project

Open Quantum Safe (OQS) is an open source project that aims at developing and prototyping implementations of the post-quantum cryptographic algorithms, to achieve security against quantum computers [SM16]. The project supports the claimed quantum-resistant KEMs and SIGs, which have been submitted to the NIST standardization process.

Since large-scale quantum computers are very likely to be built in the near future, it is important to set up an appropriate environment to evaluate these algorithms in the Public Key Infrastructure. Therefore, OQS is a significant step in this direction, giving a real time evaluation of the variant implementations of individual algorithms, flavoured with different parameters. There are two primary paths of work in the OQS project: Liboqs and OQS OpenSSL. OQS OpenSSL has branched out from the original OpenSSL toolkit with added quantum-safe capabilities.

7.1.1 Liboqs

Liboqs [SM21] is an open source C library for cryptographic algorithms that uses the current standard C11 [ISO11], and requires all its member algorithms to support it. Liboqs serves multiple purposes: (a) contains a collection of clean and open-source implementations of quantum-resistant KEMs and SIGs (b) provides an API for integrating such algorithm prototypes into networking protocols and other applications (c) serves as a test suite and benchmarking toolkit for these cryptographic algorithms.

This library also has different wrappers for specific implementations of the NIST candidate algorithms, that can be easily used in other programming languages. Liboqs makes use of another upstream project PQClean, and its clean C implementations to automate the inclusion of such algorithms into Liboqs for its purposes.

7.1.2 PQClean

PQClean [KRS+21] is another (independent) open-source project that aims to provide clean and portable implementations of post-quantum cryptographic algorithms to enable integration into protocols and applications. PQClean follows the C99 [ISO99] standard and requires that the standalone implementations of algorithms adhere to it. The C99 or ISO/IEC 9899:1999 standard, contains features taken from C++ and enhancements of some existing constructs. The use of C99 seems to be due to compatibility reasons in legacy platforms and portability reasons in varied platforms.

7.1.3 OQS OpenSSL

OpenSSL [Ope] is a commercial-grade toolkit and cryptographic software library for secure communication over the internet. OpenSSL provides instantiations of the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols. It is a command-line tool that allows users to perform cryptographic operations such as: (a) generate key-pairs for secure encryption and signing (b) generate trusted digital certificates (X509 certificates) (c) generate Certificate Signing Requests (CSRs) for certifying the generated public keys (d) verify the peer entity's identity through the certificate chain and so on.

Liboqs library of the OQS project enables the integration of post-quantum cryptographic algorithms into TLS (v1.3) through their own version of OpenSSL, called *OQS OpenSSL*, combined with quantum-resistant capabilities. OQS OpenSSL has the additional feature of choosing between using only the quantum-safe algorithms or taking a hybrid approach with classical algorithms to achieve authentication and secure communication.

7.2 Docker

Docker [Doc] is an open source product that provides platform as a service, making use of virtualization at the OS-level, to develop and deliver software applications in packages or *containers*. Containers are essentially isolated from each other and include their own software, libraries and configuration files. It allows to create and run applications separately from the underlying infrastructure enabling easy and quick shipment of software.

In this evaluation, pre-built Docker images from OQS that support different instances of a basic server-client setup, enabled with quantum-safe cryptographic algorithms, have been used for testing and evaluation purposes. It has also been used for the evaluation (Section 7.5) done as part of this thesis. This has made evaluation much easier due to the inherent isolation offered by Docker, and therefore, has been used as the primary platform to carry out all testing operations.

7.3 TLS v1.3

TLS [RFC8446] is a cryptographic protocol that establishes an encrypted channel for secure communication between endpoints over the internet, for instance, between web browsers and servers. Secure Sockets Layer (SSL) is its predecessor which has now been deprecated. The core principles of TLS can be summarized as given below:

1. Connection is *private* or *confidential* due to the use of symmetric cryptography.
   – A shared secret is negotiated for every session between the communicating parties and the symmetric keys are generated from the secret uniquely for each connection. The negotiation is ensured to be secure and reliable.
2. Identity of endpoints can be *authenticated* using public-key cryptography.
   – A public key certificate signed by a trusted Certificate Authority (CA) is provided as the proof of identity of the claimed entity.
3. Connection is *reliable* as it provides data integrity against tampering.
   – A Message Authentication Code (MAC) is included with every message transmitted to avoid tampering or loss of data.

TLS 1.3 consists of three high level protocols in its protocol stack: (i) TLS Handshake protocol (ii) TLS Alert protocol (iii) TLS Record protocol. The handshake protocol, as the name suggests, negotiates a secure session by selecting the cryptographic suites that will be used between the communicating entities. The record protocol defines the actual payload structure while the alert protocol defines the structure of control signaling and warning messages.

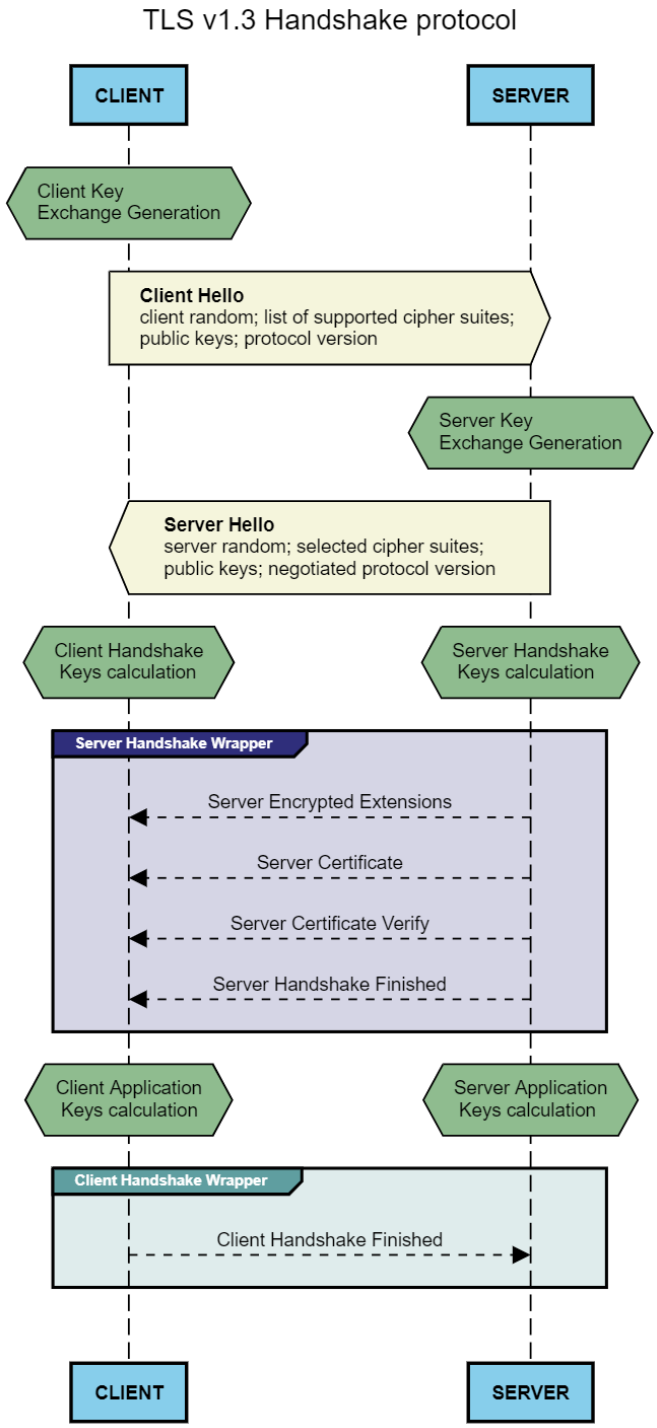


Figure 7.1: TLS 1.3 Illustrated - Handshake Protocol [DT19]

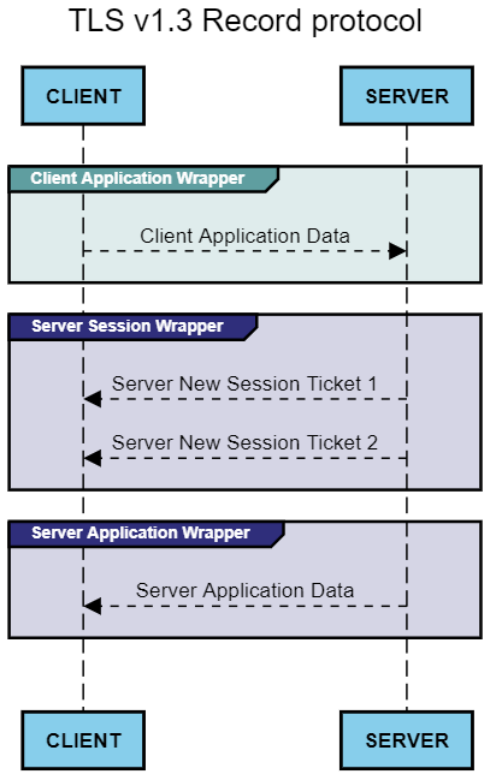


Figure 7.2: TLS 1.3 Illustrated - Record Protocol [DT19]

The handshake procedure initiated to set up a TLS connection between a client and a server is described below, while the figures Fig. (7.1) and Fig. (7.2) give a graphical understanding of the TLS v1.3 handshake and record protocols respectively.

- `Client Hello:` Client sends a connection request to the server along with a list of supported cipher suites, a nonce/random value and public keys.
- `Server Hello:` Server responds with a message that indicates the selected cipher suites, a nonce and its public keys
- `Encrypted Handshake:` Client and Server calculate a symmetric handshake key to exchange the handshake signaling messages

  - `Server Handshake Finished:` Server sends identity information through a digital certificate signed by a trusted CA and server's public encryption key
  - `Client Handshake Finished:` Client verifies the validity of the server certificate and optionally provides its own identification information through a self-signed certificate

- `Encrypted Session:` Client and Server calculate a symmetric application session key to exchange the application data
  - `Session key generation:` a suitable key exchange mechanism is used to generate the random number and unique session key with the additional property of forward secrecy
  - `Session Tickets:` Server sends two unique session tickets or Pre-Shared Keys (PSK) that can be used by the client later to start a new session without re-negotiating the connection.
  - `Session Finished:` Once the shared session keys are generated, an encrypted channel is established for the parties to communicate and terminated after the data exchange.

7.4 Evaluation

Liboqs and OpenSSL

So far, in the preceding chapters, the discussion about the six signature schemes in NIST has been qualitative and theoretical, focusing on the mathematical constructions, while OQS provides a testbed to evaluate them quantitatively. Specifically, the Liboqs platform has automated test frameworks to test the performances of the schemes. Particularly, (i) `test_sig.c` gives the performance metrics of the SIG schemes, that is, the memory usage of the scheme in bytes. (ii) `speed_sig.c` gives the speed metrics, that is, the number of operations per second of each algorithm. In the process of evaluation in OQS, we discovered that GeMSS had not yet been integrated into Liboqs or its collaborative project PQClean. Therefore, integrating GeMSS into PQClean/Liboqs became one of the objectives in this thesis. The details of this integration is presented in Section 7.5. Subsequently, the results obtained by running these test suites for the rest of the schemes have been presented and discussed in Section 8.1.

CMS and TLS

OQS OpenSSL (OQSL) allows to perform signing operations using the post-quantum (PQ) signature schemes in applications like TLS and CMS. The PQ signatures have

been used to generate certificates and validate the handshake in TLS, and used to sign a sample message and verify it successfully in CMS using the OQSL commands.

For CMS, the main steps to generate a signature and verify it are: (i) `genpkey` - generates the signing (private) and verification (public) keys. (ii) `req (csr)` - generates a certificate signing request for the public key. (iii) `x509 (cert)` - generates a signed X509 certificate that certifies the public key. (iv) `cms (sign)` - generates the CMS signature on a sample message. (v) `cms (verify)` - verifies the generated CMS signature. A sample CMS operation is shown in Listing 7.1 using the Picnic signature scheme.

```
/home/oqs $
/home/oqs $ openssl genpkey -algorithm picnic311 -out picnic311_srv.key
/home/oqs $
/home/oqs $ openssl req -new -newkey picnic311 -keyout picnic311_srv.key -out
↪  picnic311_srv.csr -nodes -subj "/CN=oqstest server" -config /opt/oqssa/ssl/openssl.cnf
Generating a picnic311 private key
writing new private key to 'picnic311_srv .key'
-----
/home/oqs $
/home/oqs $ openssl xS09 -req -in picnic311_srv.csr -out picnic311_srv.crt -CA ecdsa_CA.crt
↪  -CAkey ecdsa_CA.key -CAcreateseria1 -days 365
Signature Ok
subject-CN = oqstest server
Getting CA Private Key
/home/oqs $
/home/oqs $ openssl cms -in inputfile -sign -signer picnic311_srv.crt -inkey picnic311_srv.key
↪  -nodetach -outform pem -binary -out signedfile.cms
/home/oqs $
/home/oqs $ openssl cms -verify -CAfi1e ecdsa_CA.crt -inform pem -in signedfile.cms -crlfeol
↪  -out signeddatafile
Verification successful
/home/oqs $
/home/oqs $
```

Listing 7.1: An illustration of signing operations in CMS

```
OpenSSL>
OpenSSL> s_client -groups kyber512 -CAfile /opt/oqssa/bin/CA.crt -verify_return_error -connect
↪  localhost:4433
CONNECTED(00000003)
Can't use SSL_get_servername
depth=0 CN = localhost
verify return:1
---
Certificate chain
 0 s:CN = localhost
   i:CN = oqstest CA
---
Server certificate
-----BEGIN CERTIFICATE-----
MIINPjCCBSoCFGVM3ZaV9c/PZTKIOWgNdn6V+OqEMAOGCysGAQQBAoILBgQDMBUx
EzARBgNVBAMMCm9xc3Rlc3QgQOEwHhcNMjAxMjA1MjM1MTA5WhcNMjExMjA1MjM1
```

```
.
.
.
v8HHzeLu8wALDRIWHUVqjZKam6Sw7wAAAAAAAAAAAAAAAAAAAAAJHjFAsZIMAAAK
UYAEUqQAKCAgA2CMDSjAIMwAoBhAEaMkEASxdCm/XS+fDg==
-----END CERTIFICATE-----
subject=CN = localhost

issuer=CN = oqstest CA

---
No client certificate CA names sent
Peer signature type: Dilithium-2
Server Temp Key: kyber512
---
SSL handshake has read 6474 bytes and written 1193 bytes
Verification: OK
---
New, TLSv1.3, Cipher is TLS_AES_256_GCM_SHA384
Server public key is 9472 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
Early data was not sent
Verify return code: 0 (ok)
---
```

Listing 7.2: An illustration of TLS handshake and verification (client-side)

For TLS, after generating the key pair and the server certificates, similar to CMS using the `genpkey`, `req (csr)` and `x509 (cert)`, the additional steps at the server-side to initiate a handshake are: (i) `s_server` - instantiates a TLS v1.3 server using the server certificate and the server key. (ii) `s_client` - instantiates a TLS client that connects to the server along with a certificate chain to verify the authenticity of the server. It receives the response from the server which contains the handshake messages and two new session tickets sent by the server that can be used for resuming connections later. An illustration of a typical TLS v1.3 handshake is given in Listing 7.2, while the complete handshake for both server and client is given in Appendix B.1 and Appendix B.2 respectively.

7.5 Integration of GeMSS into PQClean and Liboqs

As previously mentioned, the details of the integration of GeMSS into Liboqs/PQClean is presented in this section. While halfway through, we realized that there has been a serious key-recovery attack [Din20] on GeMSS and its underlying HFEv- construction. This showed that developing secure while still efficient cryptosystems based on HFEv- paradigm, would probably be very difficult due to their inherent properties. Due to this reason, integrating GeMSS into liboqs to test its functionality

in TLS/CMS or other applications was not considered relevant for the time being. Therefore, we decided against proceeding with completing the integration and instead focus on the quantitative evaluation of the SIG schemes on different platforms using external benchmarking results (described and discussed in detail in Chapter 8).

Nevertheless, the partial work[1] of integrating GeMSS into Liboqs/PQClean has been described in detail here, including some of the necessary changes that were made as part of this idea. The file structure of the GeMSS-128 standalone reference implementation has been shown in Figure 7.3 that gives an idea of some of the main files that needed modification. The files (i) `api.h` - the header file that specifies the details necessary to instantiate an API, (ii) `parameters_HFE.h` - the header file that contains the different parameter sets of GeMSS for different security levels, and other such header files were modified as required to be able to integrate them into PQClean. From the clean version in PQClean, Liboqs has scripts that automates easy integration of code into Liboqs [con20a].

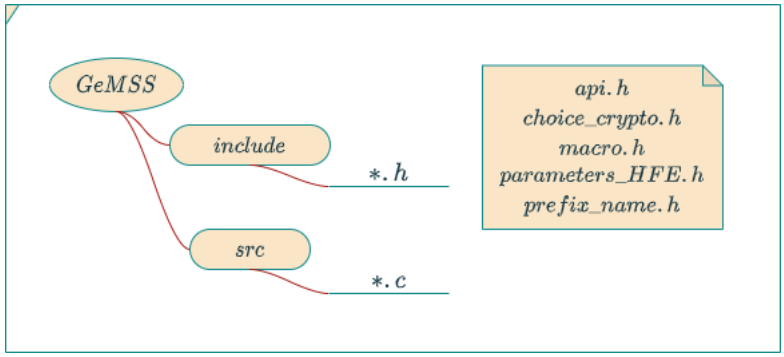


Figure 7.3: GeMSS file structure of standalone implementation

```
name: GeMSS-128
type: signature
claimed-nist-level: 1
length-public-key: 352200
length-secret-key: 16
length-signature: 66
nistkat-sha256: 0fe2b0076528923e102465b2c1af09e2436a22c1fcee82d3bf6c980625a12c72
testvectors-sha256: aa7decd4ae85ede8270030d2417e1ef0a024187391fcd00b9ee395b8a4bbd8f5
principal-submitters:
  - The GeMSS Team
auxiliary-submitters:
  - Alice
  - Bob
implementations:
    - name: clean
```

[1] upto the point where we got to know about the attack in Nov 2020

```
    version: 20201103
```

Listing 7.3: Meta.yml file after integrating into PQClean

```
#ifndef PQCLEAN_GEMSS128_CLEAN_API_H
#define PQCLEAN_GEMSS128_CLEAN_API_H

#include <stddef.h>
#include <stdint.h>

#define PQCLEAN_GEMSS128_CLEAN_CRYPTO_SECRETKEYBYTES    16
#define PQCLEAN_GEMSS128_CLEAN_CRYPTO_PUBLICKEYBYTES    352190
#define PQCLEAN_GEMSS128_CLEAN_CRYPTO_BYTES             365
#define PQCLEAN_GEMSS128_CLEAN_CRYPTO_SEEDBYTES         48

#define PQCLEAN_GEMSS128_CLEAN_CRYPTO_ALGNAME           "GeMSS-128"

int PQCLEAN_GEMSS128_CLEAN_crypto_sign_keypair(uint8_t *pk, uint8_t *sk);

int PQCLEAN_GEMSS128_CLEAN_crypto_sign_signature(
    uint8_t *sig, size_t *siglen,
    const uint8_t *m, size_t mlen, const uint8_t *sk);

int PQCLEAN_GEMSS128_CLEAN_crypto_sign_verify(
    const uint8_t *sig, size_t siglen,
    const uint8_t *m, size_t mlen, const uint8_t *pk);

int PQCLEAN_GEMSS128_CLEAN_crypto_sign(uint8_t *sm, size_t *smlen,
        const uint8_t *m, size_t mlen,
        const uint8_t *sk);

int PQCLEAN_GEMSS128_CLEAN_crypto_sign_open(uint8_t *m, size_t *mlen,
        const uint8_t *sm, size_t smlen,
        const uint8_t *pk);

#endif
```

Listing 7.4: Modified api.h file after integrating into PQClean

For this automated inclusion, a meta.yml (Listing 7.3) file was added in the reference implementation of the source code. This contains information about the entry points for the source code, the set of files that need to be imported, code version and so on. In addition, the modified `api.h` has also been shown in Listing 7.4, that contains necessary changes as part of the integration process into PQClean. The changes from the GeMSS standalone version of `api.h` (included in Appendix B.3) to the PQClean version of `api.h` (Listing 7.4), have been to retain the parameter sets for security level I and remove the other parameter sets, and to remove the macro definitions and other included header files.

Along with these changes, some of the other header files that contained generic macros to define function names, to set the parameters for each security level and so on, were merged into a common header file to create a clean version of the original implementation as per the requirements of the OQS APIs. As part of this, the modified file structure in PQClean/Liboqs looks as shown in Figure 7.4.

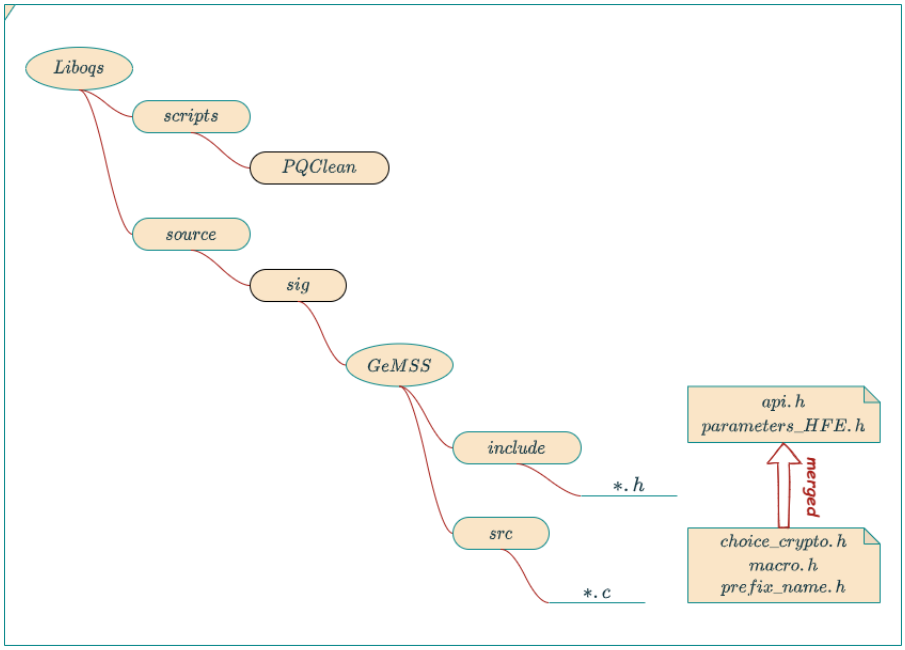


Figure 7.4: Modified file structure in PQClean/Liboqs with GeMSS

Having made these changes, the integration also included creating new IDs as per the Liboqs code structure for the reference implementation of the GeMSS algorithm, so that OQS SIG IDs could be used in the Liboqs and OQS OpenSSL APIs. This would also be needed for subsequent building and debugging of the modified GeMSS algorithm to enable smooth integration and recognition of the algorithm within the Liboqs/PQClean structure.

Chapter 8
Results and Discussion

Having understood the various types of cryptographic constructions for digital signature schemes, this chapter now addresses the final research question of weighing the pros and cons of these schemes, in terms of design, performance in software/hardware, and the security offered for different use cases.

There are three sets of measurements that form the basis for the evaluation of the signature schemes. (1) Performance measurements on an Intel Core i7-8650U at 1.90 GHz (native desktop environment). (2) Performance measurements on an Intel Xeon E3-1220 (306c3 - from SUPERCOP). (3) Hardware performance on FPGAs - Artix-7, Kintex-7 and Virtex-7 (from ATHENA). The reason to include measurements from external benchmarking platforms like SUPERCOP and ATHENA is to get a complete picture of the performances of the schemes, since a few of the schemes were not available for evaluation on specific platforms. An added motivation to include these measurements is for a comprehensive analysis that is not limited to the native desktop environment.

This chapter is organized as follows: (a) Sections 8.1, 8.2 and 8.3 present and describe the results obtained from all the three platforms used for the quantitative analysis (b) Section 8.4 presents a summary and discussion of all the six signature schemes

8.1 OQS Comparisons

As described in Section 7.4, we made the performance measurements and analysis on the native desktop environment using Liboqs library of the Open Quantum Safe (OQS) project. The measurements were made using the OQS *test nginx server* [1] and the pre-built Docker *curl* client image [2]. To start with, Table 8.1 shows the sizes (in kB) of the public key (pk), corresponding secret key (sk) and the generated

[1] [oqs20]
[2] [doc20]

signature (sig), for each of the six signature schemes considered. It can be seen that the lattice-based schemes, Dilithium and Falcon, have balanced sizes for all components, i.e. $|pk|, |sk|, |sig|$, unlike the multivariate counterparts, Rainbow and GeMSS, which have very large public keys and very small signature sizes. Similarly, the signatures based on symmetric primitives, Picnic and Sphincs+, have least key sizes but largest signature sizes compared to the others.

Signature scheme	Public key $\|pk\|$ (kB)	Secret key $\|sk\|$ (kB)	Signature $\|sig\|$ (kB)
Dilithium	1.184	2.8	2.044
Falcon	0.897	1.281	0.690
Rainbow	148.992	92.96	0.064
Picnic3	0.035	0.052	12.491
Sphincs+ (128f)	0.032	0.064	17.088
Sphincs+ (128s)	0.032	0.064	7.856
GeMSS	352	0.016	≈ 0.033

Table 8.1: The public/private key sizes for signing and length of the generated signature for different schemes.

NIST has defined different security levels for all KEM/SIG schemes as given below, based on the number of bits of security required for different applications [Moo19].

Security Level	#bits of security classical/quantum	Description
I	128-bit (AES)	exhaustive key search
II	256-bit (SHA)	collision search
III	192-bit (AES)	exhaustive key search
IV	384-bit (SHA)	collision search
V	256-bit (AES)	exhaustive key search

Based on this, Figure 8.1 shows the time taken (in ms) for signature generation and verification in OQS-OpenSSL for different security levels of all six schemes. We observe that the two lattice-based schemes take the least time to sign and verify. Picnic L1 and Rainbow Ia take minimal times for signing and verifying, while Picnic3 L1 and Sphincs+ Haraka take comparatively longer times. Rainbow Vc takes the longest time for signing compared to all other schemes.

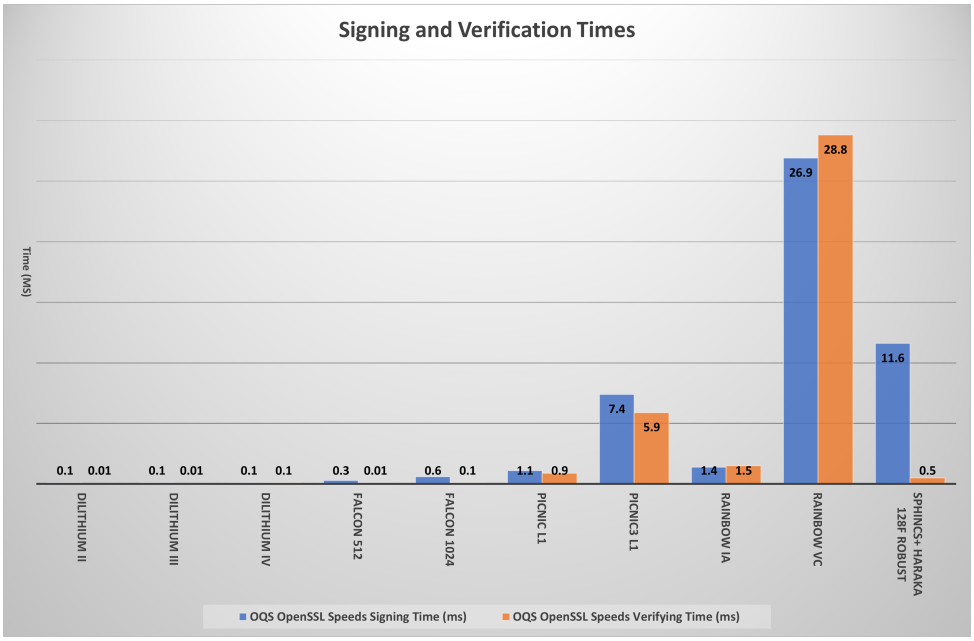


Figure 8.1: An overview of the time taken by different signature schemes to sign a message digest of 20 bytes.

A natural consequence of measuring the time taken for signature generation and verification, is the number of possible signatures and verifications per second (see Figure 8.2). It can be observed that the number of verifications per second is higher compared to the number of signatures per second. This is due to the fact that verification is usually much simpler and faster than signature generation for almost all schemes, to which Rainbow is an exception. As mentioned before, though GeMSS is not included in this measurement, it can be inferred that GeMSS might have the same behaviour as Rainbow as they belong to the same category of MPKCs.

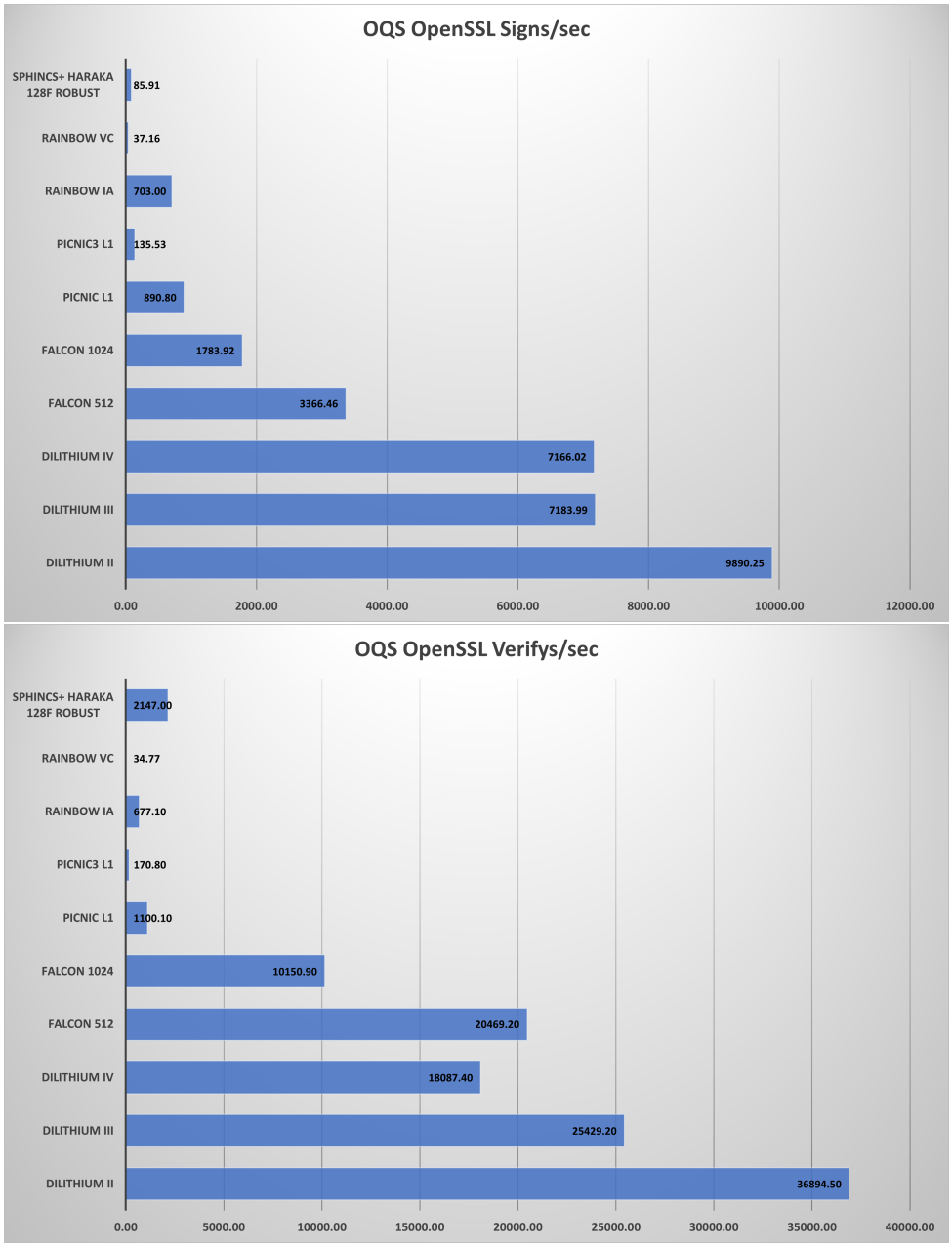


Figure 8.2: Number of signatures and verifications per second for different security levels of the SIG schemes.

8.1.1 Evaluation in TLS

Along with the measurements in Liboqs, we also wanted to measure the performance of the signature schemes in the Transport Layer Security (TLS) v1.3 cryptographic protocol. Since TLS makes use of digital signatures to create and use public-key certificates to ensure authenticity and integrity between the communicating parties, it becomes relevant to evaluate how the different post-quantum SIG schemes perform in this protocol.

Figure 8.3 shows the number of TLS connections possible per second, per user. Here, a typical connection can be the request of a default page from the server and the transfer of payload data within a specified time frame. OpenSSL allows to calculate the time taken to establish one such basic connection in a given time period. As observed, the number of TLS connections per second is proportional to the number of possible verifications per second (from Figure 8.2), for each of the SIG schemes. This is expected since TLS connections [3] typically require the client and the server to mutually be able to verify their public key certificates, in order to establish secure communication.

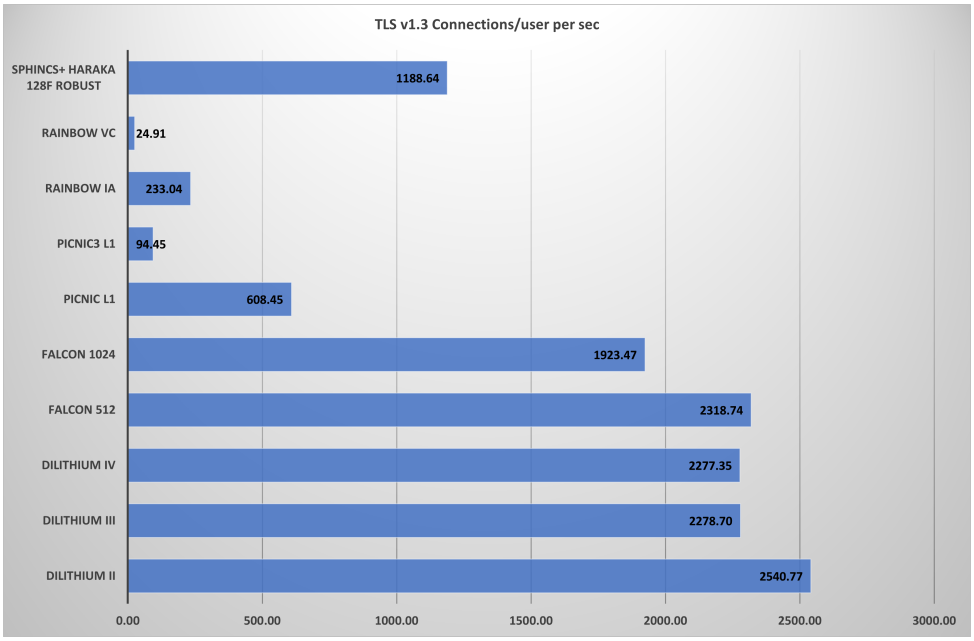


Figure 8.3: Number of TLS (v1.3) connections for each signature scheme

[3] see Section 7.3 for a detailed description of the TLS v1.3 protocol

Listing 8.1 shows a typical http response of connecting an OQS enabled curl client to an OQS enabled test nginx server for an example signature scheme. OQS allows two approaches for testing the post-quantum signature schemes (a) full-fledged post-quantum cryptography (b) hybrid (classical and post-quantum) cryptography. That is, it provides an option to use the post-quantum cryptosystems together with traditionally secure classical cryptosystems to ensure that the testing does not break the application in which it is being used for evaluation. As can be seen below, Picnic SIG scheme is used with the classical X25519 KEM scheme.

```
sahsridh@NTNU16393:~/DockerHub/test$
sahsridh@NTNU16393:~/DockerHub/test$ docker run -v `pwd`:/ca -it openquantumsafe/curl:0.4.0
↪  curl --cacert /ca/CA.crt https://test.openquantumsafe.org:6694 --curves X25519
<!DOCTYPE html>
<html>
<head>
<title>Open Quantum Safe interop test server for quantum-safe cryptography</title>
</head>
<body>
<h1 align=center>
Successfully connected using
picnic3l1-X25519
!
</h1>

Client-side KEM algorithm(s) indicated:
X25519
</body>
sahsridh@NTNU16393:~/DockerHub/test$
```

Listing 8.1: A typical response from the OQS enabled test server for one signature scheme (i.e. picnic3-L1)

8.2 SUPERCOP Comparisons

The measurements considered so far are taken on an Intel Core i7-8650U processor at 1.90 GHz (native desktop environment). However, since GeMSS could not be fully integrated (due to reasons explained in Chapter 7) into the Liboqs library of the OQS project, such measurements were not possible for GeMSS. Thus, an alternative approach was to consider measurements from standard benchmarking toolkits like SUPERCOP [4], that measures the performance of cryptographic algorithms (software implementations) on different platforms. Besides, the NIST standardization process has also considered the measurements of SUPERCOP in their preliminary evaluation of the submitted candidate algorithms.

[4]System for Unified Performance Evaluation Related to Cryptographic Operations and Primitives (SUPERCOP)

SUPERCOP [BL18] measures the performance of the cryptographic primitives, specifically with respect to SIG schemes, based on the below criteria:

- Time to hash and length of the digest
- Length of secret key and/or nonce used for the digest and signing
- Time to generate public and private keys and their lengths
- Time to sign and verify messages using private and public keys respectively and the length of the signature

Figure 8.4 [5] shows the performance of all the signature schemes on an Intel Xeon E3-1220 (*Hiphop* - 306c3 (HW + AES)), taken from the standard benchmarking tool SUPERCOP. The Intel 306c3 Core i5/i7 processor has been chosen as the target platform to obtain the measurements, as it closely matches with the native desktop environment. However, these measurements are given in terms of the number of machine cycles taken to complete each of the operations (i.e. key generation, signing and verification), as opposed to the time in milliseconds in previous measurements (on the native desktop environment).

As seen from Figure 8.4, similar to the OQS measurements, Dilithium and Falcon take lesser number of cycles for signing and verifying compared to other schemes. However surprisingly, Rainbow Ia has the least signing and verifying times compared to even the lattice-based schemes. In addition, GeMSS has average and consistent verification times for different variants but takes very large number of cycles for signing. All other schemes (Picnic, Sphincs+ and Rainbow Vc) have average performance with respect to signing and verifying.

For the key generation, we observe that Picnic is the best as it takes the least number of cycles. This is a natural consequence of using a symmetric block cipher for key generation in Picnic's design. Sphincs+ and Dilithium take average number of cycles for key generation, while the remaining schemes (GeMSS, Rainbow and Falcon) have worst performance in terms of key generation times.

[5] values on the x-axis are in the logarithmic scale (of #machine cycles) for visual convenience

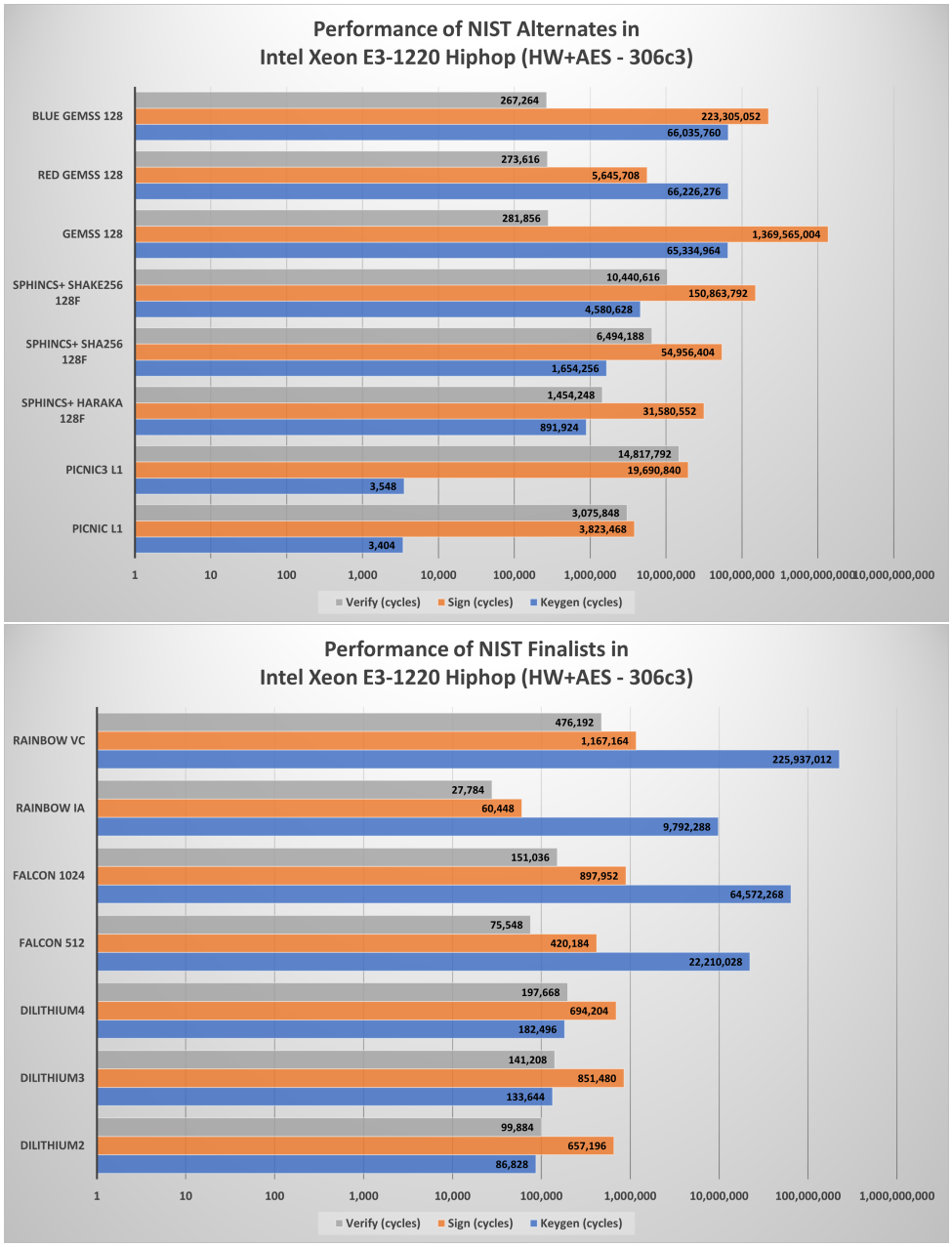


Figure 8.4: Performance of all signature schemes in Intel Xeon E3-1220 processor for different security levels (logarithmic scale on x-axis)

8.3 ATHENa Comparisons

Automated Tools for Hardware EvaluatioN (ATHENa)[6] [GKA+10] is a toolkit to automatically perform evaluations of post-quantum cryptographic algorithms on hardware platforms. Specifically, the evaluation targets are Field Programmable Gate Arrays (FPGAs), Application Specific Integrated Circuits (ASICs), and All Programmable System on Chips (APSoCs).

At the time of this writing, 2 of the 6 signature schemes considered in this work, Falcon and GeMSS, do not have hardware implementations. Out of the remaining 4 schemes, Dilithium and Sphincs+ have a High-Level Synthesis (HLS) based implementation while Picnic and Rainbow have a Register Transfer Level (RTL) based implementation [DFA+20].

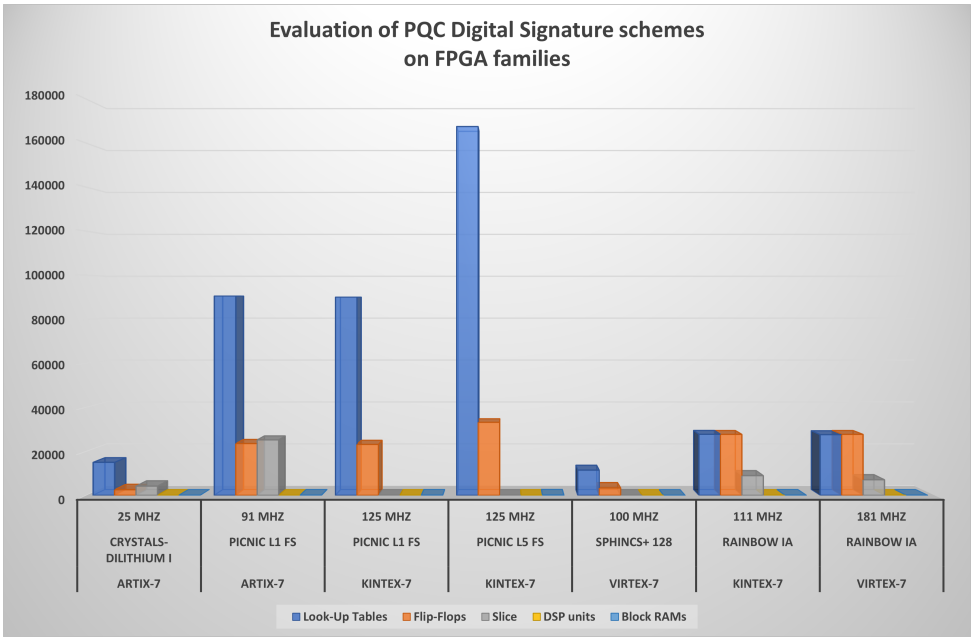


Figure 8.5: Resource usage of PQC signature schemes implemented on select FPGA families [DFA+20, BSNK19, KRR+19]

The difference between these two approaches is that, in the HLS method, the hardware implementation is synthesized from the high-level C implementation of the cryptographic algorithms. On the contrary, in the RTL method the algorithm is developed independently such that it is suitable for hardware implementation. Such

[6] named after the Greek Goddess "*Athena*" of Wisdom and Craftsmanship

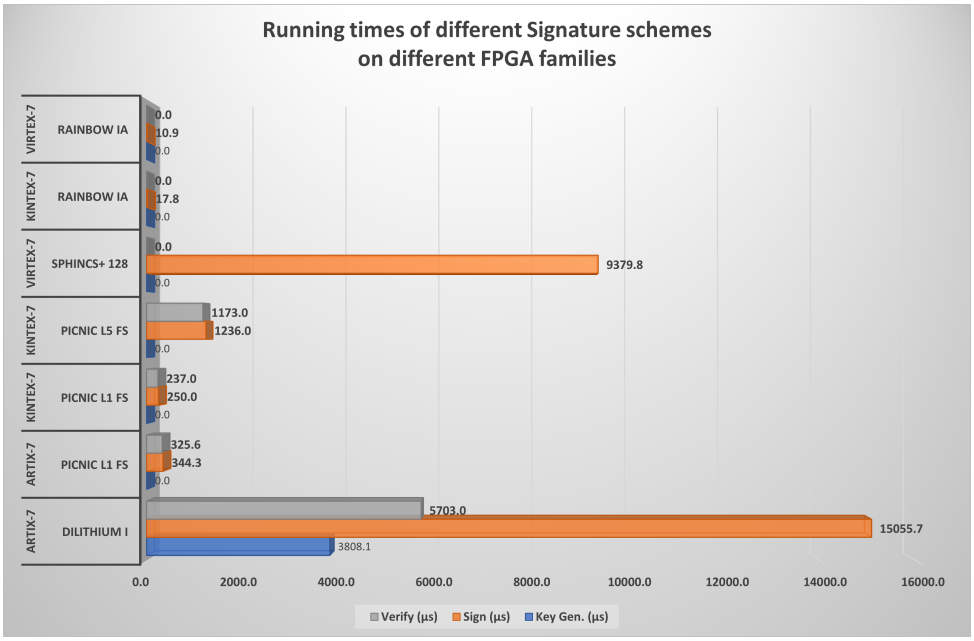


Figure 8.6: Time taken (in μs) for signature generation by different schemes on selected FPGA families [DFA+20, BSNK19, KRR+19]

hardware implementations are usually written using Hardware Descriptive Languages (HDLs) such as VHDL and Verilog.

Figure 8.5 shows the 4 signature schemes, Dilithium, Picnic, Sphincs+ and Rainbow, implemented on one of 3 target FPGAs i.e. Artix-7, Kintex-7 and Virtex-7 [DFA+20]. Particularly, it shows the usage of resources such as Look-Up Tables (LUTs), Flip-Flops (FFs), Slices, Digital Signal Processing units (DSPs) and Block RAMs (BRAMs) by the four signature schemes. Picnic has a higher usage of LUTs compared to other schemes, while Sphincs+ has the least resource usage.

Figure 8.6 shows the running times of the 4 signature schemes on selected hardware platforms. We see that Rainbow [7] has reasonable usage of resources while still having least timings for sign and verify operations. Meanwhile, Dilithium [8] and Sphincs+ have minimal resource-usage but very high signing and verification times. Although Picnic uses a large number of LUTs, it has average signing and verification times comparable across security levels in different FPGA families.

[7] the NIST round 1 version of Rainbow has been implemented in hardware [FG18]

[8] hardware implementation of Dilithium is also the most well-studied lattice-based signature scheme across the literature [BUC19].

8.4 Summary and Discussion

The observations made so far for all the SIG schemes on three different platforms are summarized in this section, by comparing them based on the post-quantum cryptographic family they belong to. This is followed by a detailed discussion on the behaviour of these schemes that is influenced by the design, security and implementation of the schemes.

Lattice-based schemes

Falcon:

1. Least sizes $\implies |pk| + |sk| + |sig|$.
2. Large number of cycles for key generation.
3. Hard to implement $\implies$ no high-speed hardware implementations so far.

Dilithium:

1. Larger sizes $\implies |pk| + |sk| + |sig|$ compared to Falcon.
2. Average number of cycles for key generation.
3. Many hardware implementations for all security levels, but performance is very slow

Having seen how Falcon and Dilithium perform in terms of running times and sizes (in software implementations), they are undoubtedly the best among the lot from both perspectives. Falcon was designed with the sole intent to generate compact signatures, which has been clearly achieved. To this extent, Falcon designers have used complex mathematics like solving the NTRU equation in lattices and innovative techniques like computation of the Falcon Tree for key generation. However, this affected the cost of key generation in Falcon compared to Dilithium, which has a simpler key generation process. Also, we conjecture that since Falcon is quite difficult to implement owing to the complex mathematics, there are no known hardware implementations at the time of this writing. Dilithium does have a hardware implementation, based on HLS design, but the running time is very large compared to other non-lattice based schemes.

In terms of design, Falcon requires high precision Floating Point Units (FPUs) for the signing process, and it makes use of a Gaussian Rejection Sampler to generate the signature components. This makes it vulnerable to fault attacks and side-channel attacks that can effectively recover the secret key [MHS+19]. Therefore, widespread adoption of Falcon may be an issue, since every implementation must be done carefully,

requiring FPUs for both security and efficiency. On the other hand, Dilithium uses a Uniform Sampler for the signature components, and is less vulnerable and easier to implement securely. Nonetheless, it still needs to be protected from fault and side-channel attacks.

In addition, Falcon is based on the GPV framework [GPV08], proven to be secure in both the ROM and QROM, while Dilithium is based on a Fiat-Shamir heuristic, proven secure in the ROM, and also recently in the QROM [DKL$^+$18, LZ19] as well. Since both schemes use a common ring and modulus (i.e. cyclotomic rings), for all security levels for efficiency reasons, it might be possible to exploit the structure of the ring to mount key-recovery attacks.

Multivariate based schemes

Rainbow:

1. Very large key size, very small signature size.
2. Least signing and verification times in software for one of the variants (i.e. Rainbow Ia).
3. Hardware implementation of NIST Round 1 version $\implies$ average usage of resources with least running times.

GeMSS:

1. Largest public key size ($\gtrapprox$ twice the size of Rainbow) but smallest signature size
2. Longest time for signing and average time for verification.
3. No available high-speed hardware implementations so far.

For the multivariate schemes Rainbow and GeMSS, the sizes of the components show an extreme behaviour due to the very nature of their constructions. Particularly, the public-key sizes are unreasonably large, while the signature sizes are extremely small. In terms of design, Rainbow and GeMSS are based on the UOV and HFEv− paradigms respectively, where HFEv− is a "Big Field" scheme, and UOV operates on "small fields". In both cases, since the public key is a set of multivariate quadratic polynomial equations, its large size is a natural consequence.

In terms of performance, Rainbow Ia has the optimum signing time than Rainbow Vc and in general, is much better compared to GeMSS. Infact, Rainbow Ia has the least signing and verifying times in comparison to all other schemes. We believe the optimum times for Rainbow Ia is due to the fact that the set of parameters used for

security level I, tend to have lesser number of operations performed over the defined field. This can be substantiated by the observation that Rainbow Ia has consistently performed better in all three measurement platforms. On the contrary, GeMSS at security level I has the worst performance, which can be conjectured to be due to the iterative operations in the extension field as part of the signing process.

In applications without space constraints, where the public key need not be computed and transmitted frequently, Rainbow can be a huge advantage, since signing is faster and the signature size is also very small. However, a major drawback of MPKC schemes is that they have no tight security reductions. Besides, recent key recovery attacks against both the variants, i.e. UOV and HFEv−, have exploited the structure of the underlying mathematical constructions. These attacks have shown that the computational cost of recovering the secret keys is much lower than the claimed security of the schemes by the designers. The attacks have also raised an important question of whether secure cryptographic constructions based on, particularly HFEv−, are possible at all or not. This is an interesting new direction in research for cryptographers and cryptanalysts alike.

Symmetric-primitives based schemes

Picnic:

1. Smallest key size with large signature size
2. Least number of cycles for key generation
3. RTL-based hardware implementation $\implies$ higher resource usage (LUTs) but average running times

Sphincs+ :

1. Smallest key size but largest signature size.
   (more or less of the same order as Picnic)
2. Average number of cycles for key generation
3. HLS-based hardware implementation $\implies$ least resource usage, but very slow in signing

Coming to Picnic and Sphincs+, both have very small key sizes but larger signature sizes. This is attributed to the fact that the designs of both schemes use symmetric primitives which offer the advantage of shorter key lengths. This is again substantiated by the number of key generation cycles in Picnic, which is a result of using a block cipher (LowMC) for key generation. However, the performance of both schemes in software implementations is average, owing to the fact that signature

generation and verification processes are iterative (in the Decomposition function of Picnic) and/or successively build on the signature (in the hypertree in Sphincs+).

Picnic is the only scheme that has a complete hardware implementation for all security levels across different FPGA families among all the signature schemes [DFA+20, KRR+19]. Even so, Picnic has higher resource usage compared to other schemes but has good performance in hardware. Picnic uses a lot of look-up operations as part of its LowMC cipher implementation that can roughly be considered to be a set of substitutions and permutations, naturally leading to a higher usage of LUTs. Sphincs+ on the other hand has minimum resource usage, as it involves only simpler hashing operations.

Both schemes are proven secure in the ROM, and recently proven to be secure also in the QROM [DFMS19, BHK+19, ABD+20]. Since their design is based on simpler symmetric primitives, straightforward implementations are vulnerable to fault and side-channel attacks, necessitating careful and tweaked implementations that are secure. Despite having innovative designs, which are not based on hard mathematical problems, the reasons for not selecting Picnic or Sphincs+ as the finalists would be (1) LowMC block cipher used by Picnic is not standardized and needs more thorough security analysis. (2) The design approach of Picnic is new and requires more understanding and cryptanalysis. (3) Sphincs+ has very large signature sizes and the time taken for signing and verifying are longer. (4) Deploying Sphincs+ as a replacement to current classic public-key algorithms will introduce significant latencies in communication.

Parameters	Good	Average	Poor
Key sizes	Picnic Sphincs+	Falcon Dilithium	Rainbow GeMSS
Signature size	GeMSS Rainbow	Falcon Dilithium	Picnic Sphincs+
Resource usage (hardware)	Sphincs+ Dilithium	Rainbow	Picnic

Table 8.2: Summary of the component sizes and hardware resource usage in different signature schemes

	Software		Hardware	
Parameters	Good	Poor	Good	Poor
Key Generation	Picnic	Rainbow Vc GeMSS Falcon	Rainbow	Dilithium Sphincs+
Signing	Rainbow Ia Falcon Dilithium	GeMSS		
Verifying	Rainbow Ia Falcon Dilithium	Picnic Sphincs+		

Table 8.3: Summary of running times in software and hardware platforms of all signature schemes

To sum up, each scheme can be categorized based on the different performance indicators like sizes, running times and the resource usage, to evaluate them in a comprehensive way. The above tables 8.2 and 8.3 summarize this comprehensive evaluation giving the reader the key points to take away from each scheme. As far as the NIST PQC standardization process is concerned, lattice-based schemes are the best for general-purpose standard cryptosystems, since they have lower signing and verifying times. Among them, though Falcon has lower sizes compared to Dilithium, the latter's easier implementation gives it an advantage over the former. The multivariate schemes have poor sizes and performance, along with the added serious attacks and uncertainty about their security, making them the worst among the lot. However, Rainbow (Ia) is promising in terms of its best performance across platforms despite its larger public key size. This seems to be the reason for considering Rainbow as one of the finalists in NIST. Finally, Picnic and Sphincs+ seem to appear in between the extremes in sizes, performance and resource usage, making them average among the other schemes. Even so, the innovative designs of both these symmetric-primitive based schemes have been the reason to include them as alternatives in NIST, and this may also lead to one of them being declared as an alternate standard.

Chapter 9
Conclusion and Future work

9.1 Conclusion

In this thesis, the quantum threat to classical public-key cryptography was recognized, leading to a comprehensive theoretical study and analysis of newer mathematical hard problems that are allegedly quantum-safe. Specifically, digital signature schemes based on such quantum-safe cryptographic constructions have been studied in detail, based on the NIST PQC standardization process.

A survey of six signature schemes (three alternates and three finalists), from the third round of the NIST process has been presented, with details of the mathematical hard problems, design, security and related attacks for each of the schemes. Complementary to this theoretical study, a quantitative and qualitative evaluation of the six schemes has been done to get a holistic overview for different practical use-cases. In this regard, we have done the quantitative performance evaluation using the Open Quantum Safe project on the native desktop environment, as it has all the schemes integrated except GeMSS. As part of this, we proceeded to integrate GeMSS into OQS in order to be able to compare it on par with other schemes on the same grounds. However, a recent serious attack put a hold on this, showing that GeMSS and its construction is highly insecure. Nevertheless, for a comprehensive evaluation of all schemes, external benchmarking measurements, both software and hardware, have been considered and presented in this work.

To conclude, lattice-based signature schemes are indeed the best signature schemes for general purpose use cases (only for software implementations). This is due to the fact that lattices have simple linear operations and strong hardness assumptions, and lattice-based cryptography has gained a lot of attention by the cryptographic community over the last decade, producing some of the most attractive cryptographic primitives and notions. For hardware implementations, Rainbow is very good in terms of performance, while the utilization costs are acceptable. For the symmetric-based schemes, their novel designs seem to be promising in terms of security and

are future-proof, which is the most attractive feature, despite average performance across both software and hardware platforms. Clearly, each of these six signature schemes have their own pros and cons, that determine the use cases where these schemes can be used.

9.2 Future Work

The study and evaluation of quantum-resistant digital signature schemes gives room to explore new applications and possibilities where efficient and secure constructions can be used for different use cases in the internet. In this direction, two of the possible use cases for quantum-safe digital signatures are discussed briefly below.

Blockchain

A *Blockchain* (BC) is a type of database that can store different kinds of information securely among a group of users, without the need for a trusted third party. It consists of *blocks* of data that are *chained* together in chronological order. BCs are mostly used as a ledger for transactions of cryptocurrency, and are transparent and immutable (i.e. unchangeable). They can be implemented as centralized or decentralized systems. The decentralized system ensures that transactions recorded are irreversible and facilitates easy traceability of the cryptocurrency.

Digital signatures are used in BCs to authorize and verify transactions and assure the authenticity of the transactions to others in the network. It might also become necessary to mandate authentication of the users to prevent misuse and exploitation of the BC technology for illicit activities. Since current signature schemes in the internet are widely based on classical, pre-quantum hard problems, they would be eventually broken and thus allow fraudulent transactions in blockchain, resulting in heavy losses to the users. Therefore, it is evident that post-quantum digital signatures will have to adapt to suit blockchain and similar applications in the future [Con20b].

Aggregate Signatures

Aggregate signatures have gained significant attention in the last few years, since they have many useful applications in the public key infrastructure (PKI). An aggregate signature is a way to combine multiple signatures on messages by multiple users. It is similar to a digital signature scheme, with an extra algorithm Combine to combine the individual signatures, and a modified Verify algorithm to verify the aggregated signature. The Combine algorithm takes as input a set containing n tuples of messages m_i, corresponding public keys pk_i, and the individual signatures σ_i computed on those messages. The Verify algorithm takes the n pairs (pk_i, m_i) and checks if the aggregate signature σ is actually generated from a combination of n individual signatures σ_i.

There are two ways of aggregating signatures - *parallel* and *sequential*. *Parallel* aggregation requires that all users generate their individual signatures simultaneously, and then these signatures can be aggregated by anyone without the need for any secret key. In *Sequential* aggregation, a signer takes the aggregate signature from the previous signer, computes his signature on the message to add to the aggregate and passes this aggregate to the next signer who signs the message and so on. The aggregate signature in this case, is sequentially and incrementally computed from one signer to the next. However, in either case, the aggregate signature must have the same length as a single signature on a message [Bon11].

Aggregate signatures have many application areas, a significant one being the PKI Certificate chains, where the shorter lengths of aggregate signatures implies reduced storage and possibly faster verification times. Since post-quantum digital signatures need to be widely adopted soon, they can be studied further to find the best ways to achieve aggregation, while maintaining a balanced trade-off between security and efficiency. Also, another important research direction would be to see which post-quantum signature constructions lend themselves to either form of aggregation - sequential and/or parallel - for different applications.

References

[AAB+19] Gadi Aleksandrowicz, Thomas Alexander, Panagiotis Barkoutsos, Luciano Bello, Yael Ben-Haim, D Bucher, FJ Cabrera-Hernández, J Carballo-Franquis, A Chen, CF Chen, et al. Qiskit: An Open-source Framework for Quantum Computing, 2019. Accessed: January 29, 2021.

[Aar05] Scott Aaronson. Quantum computing and hidden variables. *Physical Review A*, 71(3):032325, 2005.

[AASA+19] Gorjan Alagic, Jacob M Alperin-Sheriff, Daniel C Apon, David A Cooper, Quynh H Dang, Carl A Miller, Dustin Moody, Rene C Peralta, Ray A Perlner, Angela Y Robinson, et al. Status Report on the First Round of the NIST Post-Quantum Cryptography Standardization Process. *US Department of Commerce, NIST*, 2019.

[AASA+20] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, John Kelsey, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, et al. Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process. *US Department of Commerce, NIST*, 2020.

[ABD+20] Jean-Philippe Aumasson, Daniel J Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, et al. SPHINCS+ – Submission to the 3rd round of the NIST post-quantum project, 2020.

[Ant17] Einar Løvhøiden Antonsen. Lattice-based cryptography-A comparative description and analysis of proposed schemes. Master's thesis, 2017.

[ARS+16] Martin Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. Cryptology ePrint Archive, Report 2016/687, 2016. https://eprint.iacr.org/2016/687.

[BDF+11] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In *International conference on the theory and application of cryptology and information security*, pages 41–69. Springer, 2011.

[BDH+08] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. The sponge and duplex constructions. https://keccak.team/sponge_duplex.html, 2008. Accessed: January 18, 2021.

[BDPVA07] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge functions. In *ECRYPT hash workshop*, volume 2007. Citeseer, 2007.

[BDS09] Johannes Buchmann, Erik Dahmen, and Michael Szydlo. Hash-based Digital Signature Schemes. In *Post-Quantum Cryptography*, pages 35–93. Springer, 2009.

[Beu20] Ward Beullens. Improved Cryptanalysis of UOV and Rainbow. Cryptology ePrint Archive, Report 2020/1343, 2020. https://eprint.iacr.org/2020/1343.

[BHK+19] Daniel J Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The SPHINCS+ signature framework. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2129–2146, 2019.

[BHT97] Gilles Brassard, Peter Hoyer, and Alain Tapp. Quantum algorithm for the collision problem. *arXiv preprint quant-ph/9705002*, 1997.

[BL18] Daniel J Bernstein and Tanja Lange. Supercop: System for unified performance evaluation related to cryptographic operations and primitives, 2018.

[Bon11] Dan Boneh. *Aggregate Signatures*, pages 27–27. Springer US, Boston, MA, 2011.

[BS17] Dan Boneh and Victor Shoup. A graduate course in applied cryptography. 2017.

[BSNK19] Kanad Basu, Deepraj Soni, Mohammed Nabeel, and Ramesh Karri. NIST Post-Quantum Cryptography - A Hardware Evaluation Study. *IACR Cryptol. ePrint Arch.*, 2019:47, 2019.

[BUC19] Utsav Banerjee, Tenzin S Ukyab, and Anantha P Chandrakasan. Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols. *arXiv preprint arXiv:1910.07557*, 2019.

[CDG+17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-Quantum Zero-Knowledge and Signatures from Symmetric-Key Primitives. Cryptology ePrint Archive, Report 2017/279, 2017. https://eprint.iacr.org/2017/279.

[CDG+20] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. The Picnic Signature scheme. https://github.com/microsoft/Picnic/blob/master/spec/design-v2.2.pdf, 2020. Accessed: January 2, 2021.

[CFMR+17] Antoine Casanova, Jean-Charles Faugere, Gilles Macario-Rat, Jacques Patarin, Ludovic Perret, and Jocelyn Ryckeghem. GeMSS: A great multivariate short signature. *Submission to NIST*, 2017.

[CFMR+20] Antoine Casanova, Jean-Charles Faugere, Gilles Macario-Rat, Jacques Patarin, Ludovic Perret, and Jocelyn Ryckeghem. GeMSS: A Great Multivariate Short Signature. https://www-polsys.lip6.fr/Links/NIST/GeMSS_specification_round2_V2.pdf, 2020. Accessed: January 2, 2021.

[con20a] Liboqs – Contributing Guide. https://github.com/open-quantum-safe/liboqs/wiki/Contributing-Guide, 2017-2020. Accessed: January 2, 2021.

[Con20b] Luke Conway. Blockchain Explained. https://www.investopedia.com/terms/b/blockchain.asp, 2020. Accessed: January 2, 2021.

[Dam02] Ivan Damgård. On Σ-protocols. *Lecture Notes, University of Aarhus, Department for Computer Science*, 2002.

[DF17] Luca De Feo. Mathematics of isogeny based cryptography. *arXiv preprint arXiv:1711.04062*, 12, 2017.

[DFA+20] Viet Ba Dang, Farnoud Farahmand, Michal Andrzejczak, Kamyar Mohajerani, Duc Tri Nguyen, and Kris Gaj. Implementation and benchmarking of round 2 candidates in the nist post-quantum cryptography standardization process using hardware and software/hardware co-design approaches. *Cryptology ePrint Archive: Report 2020/795*, 2020.

[DFG13] Özgür Dagdelen, Marc Fischlin, and Tommaso Gagliardoni. The Fiat–Shamir transformation in a quantum world. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 62–81. Springer, 2013.

[DFMS19] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Security of the Fiat-Shamir transformation in the quantum random-oracle model. In *Annual International Cryptology Conference*, pages 356–383. Springer, 2019.

[DH76] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[Din20] Chengdong Tao Albrecht Petzoldt Jintai Ding. Improved Key Recovery of the HFEv- Signature Scheme. Cryptology ePrint Archive, Report 2020/1424, 2020. https://eprint.iacr.org/2020/1424.

[DKL+18] Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-Dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 238–268, 2018.

[DN19] Itai Dinur and Niv Nadler. Multi-target attacks on the picnic signature scheme and related protocols. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 699–727. Springer, 2019.

[Doc] Docker Inc. Docker. https://www.docker.com/.

[doc20] Dockerhub OQS Curl image. https://hub.docker.com/r/openquantumsafe/curl, 2017-2020. Accessed: January 2, 2021.

[DP17] Jintai Ding and Albrecht Petzoldt. Current state of multivariate cryptography. *IEEE Security & Privacy*, 15(4):28–36, 2017.

[DS05] Jintai Ding and Dieter Schmidt. Rainbow, a new multivariable polynomial signature scheme. In *International Conference on Applied Cryptography and Network Security*, pages 164–175. Springer, 2005.

[DT19] Michael Driscoll and Ronnie P. Thomas. The New Illustrated TLS Connection. https://tls13.ulfheim.net/, 2019. Accessed: January 2, 2021.

[Exc14] Crypto Stack Exchange. Importance of capacity and bit-rate in sponge construction, 2014.

[FG18] Ahmed Ferozpuri and Kris Gaj. High-speed FPGA implementation of the NIST round 1 Rainbow signature scheme. In *2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–8. IEEE, 2018.

[FHK+18] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-Fourier lattice-based compact signatures over NTRU. *Submission to the NIST's post-quantum cryptography standardization process*, 36, 2018.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986.

[GCZ16] Steven Goldfeder, Melissa Chase, and Greg Zaverucha. Efficient post-quantum zero-knowledge and signatures. Technical report, Cryptology ePrint Archive, Report 2016/1110, 2016.

[GKA+10] Kris Gaj, Jens-Peter Kaps, Venkata Amirineni, Marcin Rogawski, Ekawat Homsirikamol, and Benjamin Y Brewster. ATHENa-automated tool for hardware evaluation: Toward fair and comprehensive benchmarking of cryptographic hardware using FPGAs. In *2010 International Conference on Field Programmable Logic and Applications*, pages 414–421. IEEE, 2010.

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. ZKBoo: Faster zero-knowledge for boolean circuits. In *25th {usenix} security symposium ({usenix} security 16)*, pages 1069–1083, 2016.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 197–206, 2008.

[Gro96] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[HRS16] Andreas Hülsing, Joost Rijneveld, and Fang Song. Mitigating multi-target attacks in hash-based signatures. In *Public-Key Cryptography–PKC 2016*, pages 387–416. Springer, 2016.

[Hül19] Andreas Hülsing. Hash-based Signatures. https://huelsing.net/wordpress/wp-content/uploads/2019/07/20190702_PQCschool_hash.pdf, 2019. Accessed: February 02, 2021.

[HV09] Sean Hallgren and Ulrich Vollmer. Quantum computing. In *Post-quantum cryptography*, pages 15–34. Springer, 2009.

[Hø13] MAT4250 Course Høst. Cyclotomic Fields. *Lecture Notes, University of Oslo, Department for Mathematics*, 2013.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 21–30, 2007.

[ISO99] Programming languages – C. Standard, International Organization for Standardization, December 1999.

[ISO11] Information Technology – Programming languages – C. Standard, International Organization for Standardization, December 2011.

[Kat10] Jonathan Katz. *Digital signatures*. Springer Science & Business Media, 2010.

[KL20] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2020.

[Kle00] Philip Klein. Finding the closest lattice vector when it's unusually close. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 937–941, 2000.

[KRR+19] Daniel Kales, Sebastian Ramacher, Christian Rechberger, Roman Walch, and Mario Werner. Efficient FPGA Implementations of LowMC and Picnic. Cryptology ePrint Archive, Report 2019/1368, 2019. https://eprint.iacr.org/2019/1368.

[KRS+21] MJ Kannwischer, J Rijneveld, P Schwabe, D Stebila, and T Wiggers. The PQClean Project. https://github.com/PQClean/PQClean, 2019-2021. Accessed: January 1, 2021.

[Lan16] Tanja Lange. Code-Based Cryptography. *Post-Quantum Cryptography Winter School*, 2016.

[Lyu09] Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 598–616. Springer, 2009.

[LZ19] Qipeng Liu and Mark Zhandry. Revisiting post-quantum Fiat-Shamir. In *Annual International Cryptology Conference*, pages 326–355. Springer, 2019.

[Mar17] Chloe Martindale. An introduction to isogeny based crypto. http://www.martindale.info/talks/Isogeny_based_crypto.pdf, 2017. Accessed: January 12, 2021.

[McA16] Alasdair McAndrew. *Introduction to Cryptography with open-source software*. CRC Press, 2016.

[MHS+19] Sarah McCarthy, James Howe, Neil Smyth, Séamus Brannigan, and Máire O'Neill. BEARZ Attack FALCON: Implementation Attacks with Countermeasures on the FALCON Signature Scheme. *IACR Cryptol. ePrint Arch.*, 2019:478, 2019.

[MM+74] Bernard R McDonald, Bernard R McDonald, et al. *Finite rings with identity*, volume 28. Marcel Dekker Incorporated, 1974.

[Moo19] Dustin Moody. Round 2 of NIST PQC competition. *Invited talk at PQCrypto*, 2019.

[MP] Chloe Martindale and Lorenz Panny. Isogeny-based cryptography.

[MWK18] Eduardo Morais, Ceesvan Wijk, and Tommy Koens. Zero Knowledge Set Membership. In *ING Bank*. ING Bank, 10 2018.

[NIS21] National Institute of Standards and Technology (NIST). https://www.nist.gov/about-nist, 1901-2021. Accessed: January 28, 2021.

[Ope] OpenSSL Project. OpenSSL. https://www.openssl.org/.

[oqs20] OQS Interoperability Test Server. https://test.openquantumsafe.org/, 2017-2020. Accessed: January 2, 2021.

[Pei16] Chris Peikert. A decade of lattice cryptography. *Foundations and Trends® in Theoretical Computer Science*, 10(4):283–424, 2016.

[Pet17] Albrecht Petzoldt. Multivariate Cryptography Part 1: Basics. https://2017.pqcrypto.org/school/slides/1-Basics.pdf, 2017. Accessed: January 12, 2021.

[Por15] Brett Porter. Cyclotomic polynomials, 2015.

[Qis20a] Grover's Algorithm. https://qiskit.org/textbook/ch-algorithms/grover.html, 2020. Accessed: January 28, 2021.

[Qis20b] Qiskit: Quantum computing in a nutshell. https://qiskit.org/documentation/qc_intro.html, 2020. Accessed: February 20, 2021.

[Qis20c] Shor's Algorithm. https://qiskit.org/textbook/ch-algorithms/shor.html, 2020. Accessed: January 28, 2021.

[Res18] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, RFC Editor, August 2018.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[Sho94] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

[Sho99] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[Sho09] Victor Shoup. *A computational introduction to number theory and algebra*. Cambridge university press, 2009.

[SKD20] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. Post-Quantum Authentication in TLS 1.3: A Performance Study. *IACR Cryptol. ePrint Arch.*, 2020:71, 2020.

[SM16] Douglas Stebila and Michele Mosca. Post-quantum key exchange for the internet and the open quantum safe project. In *International Conference on Selected Areas in Cryptography*, pages 14–37. Springer, 2016.

[SM21] D Stebila and M Mosca. Open Quantum Safe Project. https://openquantumsafe.org/liboqs/, 2017-2021. Accessed: January 1, 2021.

[Sma16] Nigel P Smart. *Cryptography made simple*. Springer, 2016.

[Sri19] Sahana Sridhar. Towards Post-Quantum Cryptosystems - Digital Signature Schemes built on NIZK Proofs. Project report in TTM4502, Department of Information Security and Communication Technology, NTNU – Norwegian University of Science and Technology, December 2019.

[Unr15] Dominique Unruh. Non-interactive zero-knowledge proofs in the quantum random oracle model. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 755–784. Springer, 2015.

[VZGG13] Joachim Von Zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge university press, 2013.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167. IEEE, 1986.

Appendix A
Sphincs+ Security Notions

Some important security notions, required to establish the security of Sphincs+ and hash-based schemes in general, are given in the two tables below (see Table A.1 and Table A.2). These security notions determine the bounds on the complexity of mounting classical or quantum attacks on such schemes.

Pre-image Resistance (PR) or One-wayness (OW)	$K \xleftarrow{\$} \{0,1\}^k; M \xleftarrow{\$} \{0,1\}^m$ $Y \leftarrow H_K(M)$ $Succ^{\mathsf{OW}}_{\mathcal{H}_n}(\mathcal{A}) = Pr[M' \xleftarrow{\$} \mathcal{A}(K,Y) : Y = H_K(M')]$
Single-function, Multi-target Pre-image Resistance (SM-OW)	$K \xleftarrow{\$} \{0,1\}^k; M_i \xleftarrow{\$} \{0,1\}^m$ $Y_i \leftarrow H_K(M_i), 0 < i \leq p$ $Succ^{\mathsf{SM\text{-}OW}}_{\mathcal{H}_n,p}(\mathcal{A}) = Pr[M' \xleftarrow{\$} \mathcal{A}(K,(Y_1,\ldots,Y_p))$ $: \exists\ 0 < i \leq p : Y_i = H_K(M')]$
Multi-function, Multi-target Pre-image Resistance (MM-OW)	$K_i \xleftarrow{\$} \{0,1\}^k; M_i \xleftarrow{\$} \{0,1\}^m$ $Y_i \leftarrow H_{K_i}(M_i), 0 < i \leq p$ $Succ^{\mathsf{MM\text{-}OW}}_{\mathcal{H}_n,p}(\mathcal{A}) = Pr[(j,M') \xleftarrow{\$} \mathcal{A}((K_1,Y_1),\ldots,(K_p,Y_p))$ $: Y_j = H_{K_j}(M')]$

Table A.1: Definitions of the security notion of *pre-image resistance* or *one-wayness* for hash-based schemes; p - number of *targets*. [HRS16]

The security notions of OW and SPR have been defined in Section 2.3.3 but given here again for ease of understanding MTA in hash-based schemes. The bounds on the attack complexity are determined by the success probability of an adversary $\mathcal{A}$, against the scheme, to efficiently find a pre-image of a given hash function and produce a valid forgery. (a) SM-OW defines the success probability of $\mathcal{A}$ to find at least 1 pre-image, given p targets or images Y_i, of the keyed hash function $H_K(\cdot)$. (b) MM-OW defines the success probability of $\mathcal{A}$ to find an instance j among many keyed hash functions $H_{K_j}(\cdot)$ and a pre-image M', given p targets, such that it

produces the j^{th} image Y_j. (c) SM-SPR defines the success probability of $\mathcal{A}$ to find another pre-image M', given p targets or pre-images M_i of the keyed hash function $H_K(\cdot)$, which is not in the set of received pre-images but collides with one of M_i. (d) MM-SPR defines the success probability of $\mathcal{A}$ to find an instance j among many keyed hash functions $H_{K_j}(\cdot)$ and another pre-image M', given p targets, which is not in the set of received pre-images but collides with one of M_j given as input to the instantiated j^{th} keyed hash function $H_{K_j}(\cdot)$. All the defined probabilities are made to be negligible so that the attack complexity is very high, thus ensuring the security of the scheme.

Second Pre-image Resistance (SPR)	$K \xleftarrow{\$} \{0,1\}^k; M \xleftarrow{\$} \{0,1\}^m$ $Succ^{\mathsf{SPR}}_{\mathcal{H}_n}(\mathcal{A}) = Pr[M' \xleftarrow{\$} \mathcal{A}(K, M)$ $: M' \neq M \wedge H_K(M) = H_K(M')]$
Single-function, Multi-target Second Pre-image Resistance (SM-SPR)	$K \xleftarrow{\$} \{0,1\}^k; M_i \xleftarrow{\$} \{0,1\}^m, 0 < i \leq p$ $Succ^{\mathsf{SM\text{-}SPR}}_{\mathcal{H}_n,p}(\mathcal{A}) = Pr[M' \xleftarrow{\$} \mathcal{A}(K, (M_1, \ldots, M_p))$ $: \exists\ 0 < i \leq p : M' \neq M_i \wedge H_K(M_i) = H_K(M')]$
Multi-function, Multi-target Second Pre-image Resistance (MM-SPR)	$K_i \xleftarrow{\$} \{0,1\}^k; M_i \xleftarrow{\$} \{0,1\}^m, 0 < i \leq p$ $Succ^{\mathsf{MM\text{-}SPR}}_{\mathcal{H}_n,p}(\mathcal{A}) = Pr[(j, M') \xleftarrow{\$} \mathcal{A}((K_1, M_1), \ldots, (K_p, M_p))$ $: M' \neq M_j \wedge H_{K_j}(M_j) = H_{K_j}(M')]$

Table A.2: Definitions of the security notion of *second pre-image resistance* for hash-based schemes; p - number of *targets*. [HRS16]

Appendix B
Code Listings

TLS Handshake - Server side

```
~ $ openssl req -x509 -new -newkey dilithium3 -keyout dilithium3_CA.key -out dilithium3_CA.crt -nodes -subj "/CN=oqstest_CA" -days 365 -config /opt/oqssa/ssl/openssl.cnf
Generating a dilithium3 private key
writing new private key to 'dilithium3_CA.key'
-----
~ $
~ $ openssl genpkey -algorithm dilithium3 -out dilithium3_srv.key
~ $
~ $ openssl req -new -newkey dilithium3 -keyout dilithium3_srv.key -out dilithium3_srv.csr -nodes -subj "/CN=oqstest_server" -config /opt/oqssa/ssl/openssl.cnf
Generating a dilithium3 private key
writing new private key to 'dilithium3_srv.key'
-----
~ $ openssl x509 -req -in dilithium3_srv.csr -out dilithium3_srv.crt -CA dilithium3_CA.crt -CAkey dilithium3_CA.key -CAcreateserial -days 365
Signature ok
subject=CN = oqstest_server
Getting CA Private Key
~ $
~ $ openssl s_server -cert dilithium3_srv.crt -key dilithium3_srv.key -www -tls1_3
Using default temp DH parameters
140380408016200:error:02006062:system library:bind:Address in use:crypto/bio/b_sock2.c:161:
140380408016200:error:20093075:BIO routines:BIO_bind:unable to bind socket:crypto/bio/b_sock2.c:162:
   0 items in the session cache
   0 client connects (SSL_connect())
   0 client renegotiates (SSL_connect())
   0 client connects that finished
   0 server accepts (SSL_accept())
   0 server renegotiates (SSL_accept())
   0 server accepts that finished
   0 session cache hits
   0 session cache misses
   0 session cache timeouts
   0 callback cache hits
   0 cache full overflows (128 allowed)
~ $
```

Listing B.1: Complete TLS handshake - Server side

TLS Handshake - Client side

```
TLS_DEFAULT_BASIC
/ # openssl
OpenSSL>
OpenSSL> s_client -groups kyber512 -CAfile /opt/oqssa/bin/CA.crt -verify_return_error -connect
↪  localhost:4433
CONNECTED(00000003)
Can't use SSL_get_servername
depth=0 CN = localhost
verify return:1
---
Certificate chain
 0 s:CN = localhost
   i:CN = oqstest CA
---
Server certificate
-----BEGIN CERTIFICATE-----
MIINPjCCBSoCFGVM3ZaV9c/PZTKIOWgNdn6V+0qEMAOGCysGAQQBAoILBgQDMBUx
EzARBgNVBAMMCm9xc3Rlc3QgQOEwHhcNMjAxMjA1MjM1MTA5WhcNMjExMjA1MjM1
MTA5WjAUMRIwEAYDVQQDDAlsb2NhbGhvc3QwggSOMAOGCysGAQQBAoILBgQDA4IE
oQD9SIs0uKB2OpHhuttr8bOLqWf/n/Bh+I65LiaOcU3qxxbpyFu/hguocQrsysNV
tgMPNSSyE5VZlU+YOaXIrQowgOPwzogKKHbWdka2PBfrEePu/FOT5aUtR3RaScDN
F37uwCjZ1SJ1mowRYJsWgUPRoyzpsA26HfFFTAO8VfsyeEb/rn1jSGzG2mVoaDSV
GbLxS7mfUnPcXUrBqn0hpVyw+8uHeAR1fKOu844alolXppBRyUVXiLR1O8pPe87H
c6i/o28+LQAIjzjDTiIUrLGfcKvv5gKF1VIKuk4mVSxy8PI5hqRY5T+13bJkoU5+
OlscK4c1bthAOOSPvlzxgpmHeJp+kT7RfOYtowhJsVsdyBlkpHKZR7mwvqGEzpyp
YvLlqgqtAynwD6AaGFUALo1um41Uy0DEhZPcI3BB276L7gUNrRvSy3RJTcAcwNsa
Z/rw/JB2OSZVyA59nE6OwjFSLNyt+SppBdCTUS95LVbDRJGuJrqJEjTmKhgH+TPh
UOfeCJkD2ZKaflBOZYEu+Lj5sZh4wdzUW4LAum0kl2WmaVDbtJcgmOfMm5knD8k4
5vUvu5cOQmyNgeKCSC7OoCrURMUnWwdQrFGmwaNE7R74JpBf5FRamNPq6dmG8Yug
K4l6XajmzOnEG7iTRo6VP+rn7LFo7s+qjmF5R44un5VDWpxaoHmi4V/OPkJbPBEP
XPdZFf7q2imr7O0kcbZ84kpD9JpstQIH+/cWaTzsXVLqGvNWGava64CAFmBGpXTo
NHgMuSlXFOOrRTp1w14NNjSenJ2ckYrkTumS2z/z20GN2ynZZ5Nt48nxplkEN9e7
NHfvafj9iYYMGLdMkCJ7tr1woJHQp41hT+jMk/XXdzoDnjNsFQlsUn5LyXNIB2tS
HwgtUBk+tmH5eB86vK2wwt5xllk6SGSahAHnwXn0QzBQUUbbRXilCBoBCcOYtSAO
059d2TpfUZ62qgmW1rrQOZry16AdfLNGD1TpwyuzBWy2CaWL1ui2gFZTdQsMK5Oq
2x4hNbbknnjjqRgxClEi5MuQ6JzJAewYOwkyskLYx8XlJEtryN5npPTSNANmIuOf
M48RdfBqyrslYTNaZ1lkr6xCcenHFP17iAZQ6k+Kik9weqFpWL93qtCcGvdbX3I/
1c6O1WOMPchXeyHne8KqFIuxqgDR7Co3t46RBgjSCmSoYm4nGrkUnftdu4+bcf7/
kxN964uSW9NpTOf20nUral+WXQC6dt/hoNmmf+Qu/9pHqSvbUIZIOQqIVI35NdOO
kTiUOODrGDkgvsWRnROZMB1Tn28pUpf5F2yWn2dpXBzDxCNxb9Wx8Q63iyBw71SQ
oPo8evg9Ui3mVwkYzqcsO4bhqYVWcZ8nTckevmfSJurgohkaXwaO1zcBg+dF6CIW
JCAbyo0FqZ4b6AnE8OVRPhFQ+ogUmZ9Pdepa+x6y0ny0hWjVO9WhiGnw7WmJCpTw
YpNOnB7fzZOsXauTTM3sdy7p2E2O1trhjvHWqLtbienEKgXKVCJBhrtpk91cypD9
SyOfh1YsYBjF+OHIrawwZ4e+BzdjZRQt8sPi7g2qaAN3UjANBgsrBgEEAQKCCwYE
AwOCB/OA3/xOQNs2gLAQ8UsVOAGS+p7CEumkSrakuC1DCLJ6ot3sSBfOF9MWRYtq
ygCZOUPxoHKJp3aZXKBDCcvI4OX2KnskKzDHXlq4adTlLthLh7cFRZGbUjpqPDts
czMBNZ4LeVLmkqLziX6H57owv2FDk8RIOzzomvx5GhVTJuqqJ5IU7qO/xPuwbon1
uDFB88tO6Rk9IVkTOZ5HfQJBfoO+6SOCeUHNYDXFDHVKGxjY+Z2d3T5EAZi1F7fw
1yF5xs9RDh7wdYBgD+Uj7bt49tKZo1zb4fNBcON8YlcsoPGptiSD/QxPhqHnOnV/
OIDOsHkyom9pWsZf51sNm8PdWXXnK4srHESpvR94wjnpEgoPQpk5nJTnLRDhn7vO
U4c2du1IU1SVdLXupOEIFNGlmk1o/oRzaqzEwXatiMOhRo/OmKT+rYWyLFgkEHDM
3a5gYimSSn+CLjX2XiZoc4FXy7rBubBogGsjTiePsyRPFsKs7QPyzZ/hpBe5DWuY
Dsza58PCMWGfvbTkZ9VCrcmqmSYVOgWgxcbfqU7cHaYPYogApwRnl9R/J+HObLKH
aJ5ZEmIO49GpYOTO9I9Pc2rZkt3GyIu2+iJhk+tF3XY6mnDb98KTOxSK9TRFo65h
mA+pC8k9PDJ4OR3iWmohYsP8QERdkSUHOoREDzO1DcY7yUo32nEeR2/JcTlOTCzh
DMiM621oaFxQQ5l/e18f+PDgd5Ywf6nfL+wiOCNjOr2nUOrZiyMekBsd8j3T9ePh
SttKvYgVSDT319kZTXUw9vlrdeWfxmjTzwlWbdB/efq6Bl8MYBdAA/WDhwszPRGV
ZFdcjWa2ugr6Awh+lJtIsrOSatsg//+OKmDOa5alEdFY+QMnWPYNchoWvYNbf89y
JN2E6IY4uHu2/eCQDc+C3txSAF3+p4iv8Dhx1l3q8/5emkzA99EclKl+R+z/SXpW
```

```
Nl3/Srfb7RlugWYv5HVtDh9LxBIFd4pJL5i4fr625hdk1cUpuR48xG4q+BL/JeQu
Z7Co6sNJv53Om7CYbBEDHTCzO5w9oPiV4NMJofoXvvANTDK8js0Hl+CKMYBSFRU+
1j6xniILFmqp2BZla3yrZNL/vnq+6IEVtcxsIAHOGxxJOHWgxiuoKuTe5H5s8Ec2
shoSulszrNuP7cjaW8Bw3GIwyvh3yHx9Mte3KOa+1fwuL7Y7ASFVtNCazVFcyDtq
4QdA1fEsCnepLxjxpaHu4HH7y3vvNuh5saOA+zEz8S/UeXzCtjIf7lLwZBN+shEH
OSn+hLcFBG3C3CQuxkdQDO2+bqEJ62CT1eyPGtqJhT6RFnY/aiPxjD42ezIaBaGX
r4U19ZUSpzWGqilBl1p5a/K5ggNoYpIfOFmaDuAmQgVUtLTS3aTYblf/nEvqIgha
gzglWClJ+2SsAuXD5q21rCJs2Un7WzXPjxrH8nhL/vwFykr8epJaekbXPCQuMGmt
62OhmQ2OnGBiAFdFtX985GIXj9pphheWvKolIwnGOTsx99MkfqVWAIrMIonDzUZo
aq/xkvDvBZ3jyeOmgmynldz6lOQehZHLVczO649JH3s/wxQBs0t5+/XXrO5cGIWT
bAtJOO8Yn3BpgQotuJcCE+/p/cPEx/D95ZaNh86ZylrX1oNouO39zMicyzUCgPcK
Dn7GYjSaXZhzaV4YLY+Pv0XHoZca8/VS68leOKsZpBjVHLcefyo0egZpQ+FO2+TT
sk9/UBS1YOSYYsGd4t3EcfFC2XoHYH9/kCXrcMZXIiLOVI3v2KuJgFn5C+++U6Ob
+elGoCl9ePVYpXqRSbXRtZMJa7qhcvj78kcZRZVge5U/FLSBz5bUjLaWIIpVGwFP
RHShc+r1JMVnSUXsFZreXtXPJNOrpWrG2EVU/aT68tXXHxicQAh1QK8r6Vm57GgY
svhswj2Yex88RpqXuUD+L9r9N/SKLz1+nFz3G64P5OOfHZQjoO/hKo9RoVoXjHaj
Zl6a7xyqj9Aws56NzhmGCY2bLVRkvIUxXU9iBDCsHMcFOC6ecK8OkRHAVUUNj+Za
45RfZ4YOsP5RtPMAgm/TeoBJO7XxJuHhB/Bjo67WZOQF3BPXcxsiT2xLMwKPxz56
sEHzGdcBOIY81g6mnh2k+rZtdHX6cDotmF48u0+wPbZlje5+BP8eJOzMTI7vkTw8
3RMV3GwMZ1WXHKZVMropIWhtxhbHscmV+McIOhRvtSOy6PTZ3iNDOn8JngdyrGWR
C9RCpYlGpaPvtf5cqBiHKj1iIBiFG19gMBX/422Q1kMQwMzibXUrFB9tkNCs2Wjx
zKlxmNDwac7VK3PbC+bLT7kwi2OpyytcWNqqpDDUWTKOeQkvyZIM+v5ZyvTJdOD6
lkM4NA/OnRQK/T+VyOTBY7EBYSZRlN9Opx2biKcMoD9K/gGqi9VNsZ7xR44zpBDn
omfQay+K/IetFiHA8dOvxG7ZDdahQYkifeUHylB/MEiYFXwgGwuwT6gOLy6ZBdi2
7riaXLS4O+hZNFzYWjfkNvs8fLUhaJiJo5ItR/NFKa82nKUSriShp2wYvblI1sXf
zjbQ2tf+BQgXTaaz6O35FycoNzlIVWBrgIeanqi5ON3j5vX+AAOVIStOXGlzgYip
v8HHzeLu8wALDRIWHUVqjZKam6Sw7wAAAAAAAAAAAAAAAAAAAAAJHjFAsZIMAAAK
UYAEUqQAKCAgA2CMDSjAIMwAoBhAEaMkEASxdCm/XS+fDg==
-----END CERTIFICATE-----
subject=CN = localhost

issuer=CN = oqstest CA

---
No client certificate CA names sent
Peer signature type: Dilithium-2
Server Temp Key: kyber512
---
SSL handshake has read 6474 bytes and written 1193 bytes
Verification: OK
---
New, TLSv1.3, Cipher is TLS_AES_256_GCM_SHA384
Server public key is 9472 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
Early data was not sent
Verify return code: 0 (ok)
---
---
Post-Handshake New Session Ticket arrived:
SSL-Session:
    Protocol  : TLSv1.3
    Cipher    : TLS_AES_256_GCM_SHA384
    Session-ID: C14C2B060A705FA053D0BAAEC1CA29A6F6888C87596E722BA2648673E37C40ED
    Session-ID-ctx:
    Resumption PSK:
    ↪  E0073940105052225E1D18C2619AF42B69C070E283ACE9513EAE089ECC8C2DD34F509E5F55D28DA358A734332D7B50E4
    PSK identity: None
    PSK identity hint: None
```

```
    SRP username: None
    TLS session ticket lifetime hint: 7200 (seconds)
    TLS session ticket:
    0000 - dc b0 d3 ff a0 ed ca c9-20 69 73 82 17 f5 56 55   ........ is...VU
    0010 - 84 58 f3 82 63 7a 35 00-29 97 cc ea f1 f5 03 cb   .X..cz5.).......
    0020 - 03 d8 98 95 2a 92 3c 9f-a2 ba 68 6b f6 ac b0 24   ....*.<...hk...$
    0030 - ed 89 62 a5 53 b6 23 9a-bc d3 8b 9f e5 57 4b 41   ..b.S.#......WKA
    0040 - 04 98 81 bd e9 25 83 11-63 c8 13 66 76 21 73 8d   .....%..c..fv!s.
    0050 - 5c 3d f5 01 b3 70 58 a0-32 76 2f 01 a1 4e a0 be   \=...pX.2v/..N..
    0060 - 12 29 c9 7b d7 d0 fd 06-0c a0 c0 cf bd 1e 06 a7   .).{............
    0070 - 83 ab 9e 30 af 82 ef 7e-cf 9c 71 b2 c1 76 77 c8   ...0...~..q..vw.
    0080 - 68 22 6f 8c 1a e2 1f ac-a9 01 48 8c 5b d9 76 d4   h"o.......H.[.v.
    0090 - 55 24 2a 67 2a 64 c2 69-32 c0 e9 32 8f 4b 8b 7f   U$*g*d.i2..2.K..
    00a0 - 42 ab d6 f6 49 25 23 cb-ed 16 9d 07 7c 30 cc f0   B...I%#.....|0..
    00b0 - 1f 5a 4c b6 7f 31 1e 48-6b b0 b2 c9 f7 dc a1 17   .ZL..1.Hk.......

    Start Time: 1607278278
    Timeout   : 7200 (sec)
    Verify return code: 0 (ok)
    Extended master secret: no
    Max Early Data: 0
---
read R BLOCK
---
Post-Handshake New Session Ticket arrived:
SSL-Session:
    Protocol  : TLSv1.3
    Cipher    : TLS_AES_256_GCM_SHA384
    Session-ID: 0926D7EF572CA824DA4803FE7BD1B1D681327D7BCD7E69F9FFAB1F9CA5B38C56
    Session-ID-ctx:
    Resumption PSK:
↪   3048689CAA813C0C03B38007FA85FF7E4C47462C60B0DF67EAEA9A90711BC809721DFDD43EFF4C11458CA7E1B72CB276
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 7200 (seconds)
    TLS session ticket:
    0000 - dc b0 d3 ff a0 ed ca c9-20 69 73 82 17 f5 56 55   ........ is...VU
    0010 - 8f b0 7e 8f 7c 0b 2d 94-08 e8 ca f7 e4 8a c0 0f   ..~.|.-.........
    0020 - 6b d7 5c ac f7 87 bb 01-95 9b 0b cd fc a6 f4 8c   k.\.............
    0030 - 0d 9e 6a 97 fe da 3f 14-22 67 36 37 67 cf 45 ab   ..j...?."g67g.E.
    0040 - f3 9e 61 bd 0a 80 8d c9-d7 3a 29 25 8b 53 71 b3   ..a.......:)%.Sq.
    0050 - a6 73 53 78 b0 71 81 67-9a 34 39 32 6c e1 cf c8   .sSx.q.g.492l...
    0060 - ef 2b 3c 83 86 28 09 58-8d 08 9d 37 cb 9c eb 9d   .+<..(.X...7....
    0070 - ee 09 9a 61 c5 f7 f1 8b-55 c8 40 31 ff 20 f5 a5   ...a....U.@1. ..
    0080 - 6d e3 41 13 98 53 60 e9-62 6b 37 79 41 1a aa 7c   m.A..S`.bk7yA..|
    0090 - 94 f3 3e 1d 05 3a c4 62-cc a4 24 3d 29 e7 70 42   ..>..:.b..$=).pB
    00a0 - 2e 8c f2 d1 7b c4 64 00-8e 70 4b a1 3c 8f 44 ff   ....{.d..pK.<.D.
    00b0 - c1 49 47 16 e8 68 0a 36-9b 3d 92 2b 5c c5 91 13   .IG..h.6.=.+\...

    Start Time: 1607278278
    Timeout   : 7200 (sec)
    Verify return code: 0 (ok)
    Extended master secret: no
    Max Early Data: 0
---
read R BLOCK
```

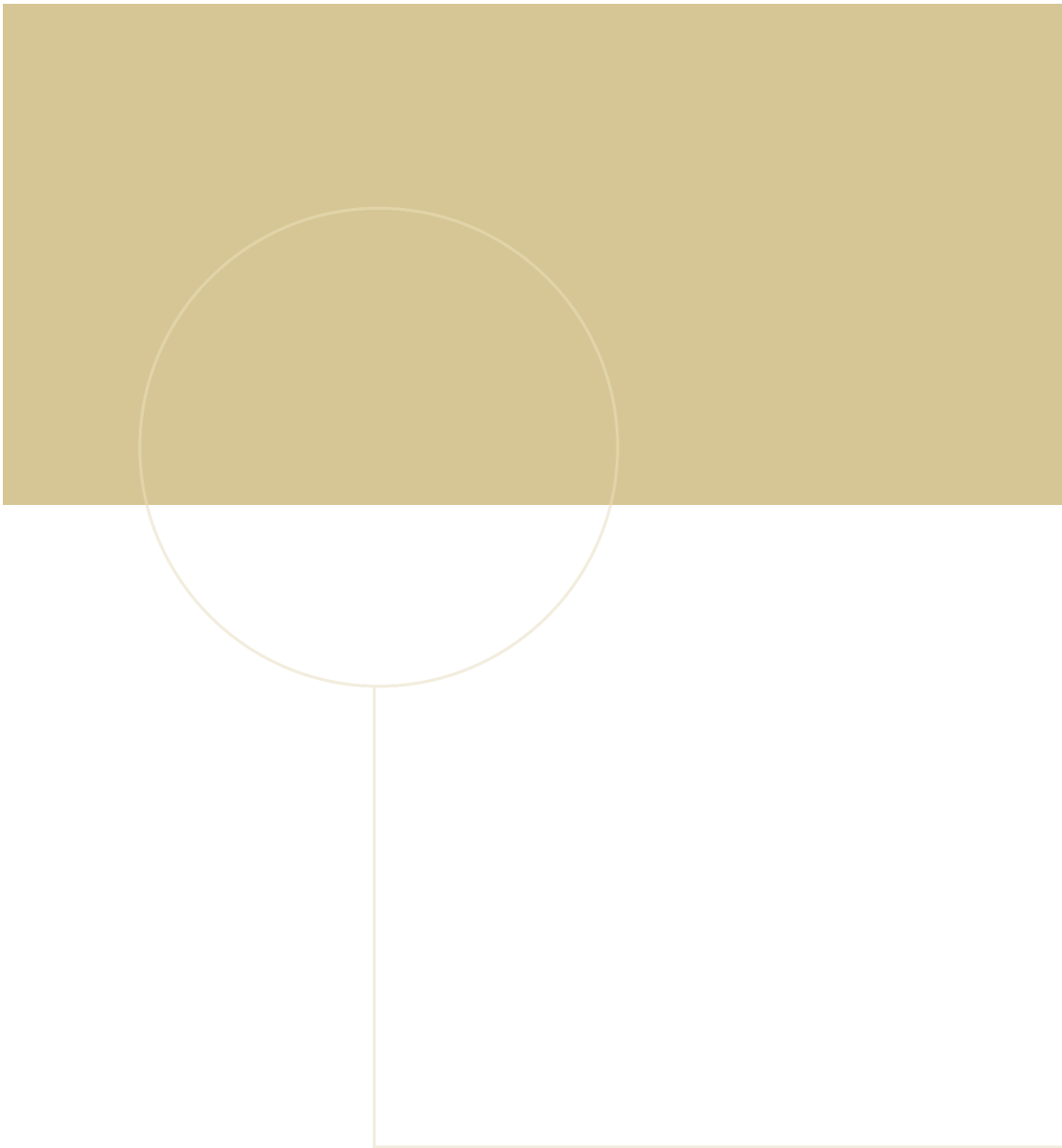
Listing B.2: Complete TLS handshake - Client side

GeMSS `api.h` file

```
#ifndef _API_H
#define _API_H

/** Set to 1 if you want to use MQsoft in SUPERCOP. */
#define SUPERCOP 0

#include "prefix_name.h"
#include "parameters_HFE.h"
#include "sizes_HFE.h"
#include "choice_crypto.h"
#include "sizes_crypto.h"


/** Size of the secret-key in bytes. */
#define CRYPTO_SECRETKEYBYTES SIZE_SK
/** Size of the public-key in bytes. */
#define CRYPTO_PUBLICKEYBYTES SIZE_PK
/** Size of the signature (without the document) in bytes. */
#define CRYPTO_BYTES SIZE_SIGN


/** Name of the current used cryptosystem. */
#ifdef GeMSS
    #define CRYPTO_ALGNAME "GeMSS"
#elif defined(BlueGeMSS)
    #define CRYPTO_ALGNAME "BlueGeMSS"
#elif defined(RedGeMSS)
    #define CRYPTO_ALGNAME "RedGeMSS"
#elif defined(WhiteGeMSS)
    #define CRYPTO_ALGNAME "WhiteGeMSS"
#elif defined(CyanGeMSS)
    #define CRYPTO_ALGNAME "CyanGeMSS"
#elif defined(MagentaGeMSS)
    #define CRYPTO_ALGNAME "MagentaGeMSS"
#elif defined(FGeMSS)
    #define CRYPTO_ALGNAME "FGeMSS"
#elif defined(DualModeMS)
    #if INNER_DualModeMS
        #define CRYPTO_ALGNAME "Inner_DualModeMS"
    #else
        #define CRYPTO_ALGNAME "DualModeMS"
    #endif
#elif defined(QUARTZ)
    #define CRYPTO_ALGNAME QUARTZ_
#elif defined(QUARTZ_V1)
    #define CRYPTO_ALGNAME QUARTZ_V1_
#else
    #define CRYPTO_ALGNAME "MQsoft"
#endif
```

```
#if SUPERCOP
int
crypto_sign_keypair(unsigned char *pk, unsigned char *sk);

int
crypto_sign(unsigned char *sm, unsigned long long *smlen,
            const unsigned char *m, unsigned long long mlen,
            const unsigned char *sk);

int
crypto_sign_open(unsigned char *m, unsigned long long *mlen,
                 const unsigned char *sm, unsigned long long smlen,
                 const unsigned char *pk);
#else
int
PREFIX_NAME(crypto_sign_keypair)(unsigned char *pk, unsigned char *sk);

int
PREFIX_NAME(crypto_sign)(unsigned char *sm, unsigned long long *smlen,
                         const unsigned char *m, unsigned long long mlen,
                         const unsigned char *sk);

int
PREFIX_NAME(crypto_sign_open)(unsigned char *m, unsigned long long *mlen,
                              const unsigned char *sm, unsigned long long smlen,
                              const unsigned char *pk);

#define crypto_sign_keypair PREFIX_NAME(crypto_sign_keypair)
#define crypto_sign PREFIX_NAME(crypto_sign)
#define crypto_sign_open PREFIX_NAME(crypto_sign_open)

#endif


#endif
```

Listing B.3: Existing `api.h` file in GeMSS standalone reference implementation



NTNU

Norwegian University of
Science and Technology