RWTH Aachen University

Bachelor thesis

Implementation of the PQC algorithm BIKE in theorem prover EasyCrypt

Simon Bausch

Reviewers are
Prof. Dr. rer. nat. Dominique Unruh
and
Prof. Dr.-Ing. Ulrike Meyer

Fachgruppe Informatik
Lehrstuhl für Quanteninformationssysteme

November 13, 2024

Contents

Acknowledgement

First and foremost, I would like to thank my supervisor Prof. Dr. Dominique Unruh for his support and guidance throughout this thesis. He was always available for questions and provided valuable feedback on my work. His knowledge and expertise in the field of quantum information systems as well as cryptography has helped me a lot during the thesis. Also, I would like to thank my family and friends for their support and encouragement during this time.

1 Introduction

> *"They call it Q-Day: the day when a quantum computer, one more powerful than any yet built, could shatter the world of privacy and security as we know it."*
>
> — Zach Montague, New York Times

Cryptography is the science of secure communication in the presence of possible adversaries. It has been a critical element of communication for thousands of years, evolving from simple substitution ciphers to complex mathematical algorithms that form the backbone of modern digital communication. In today's connected world, cryptography plays a vital role in protecting sensitive information, ensuring privacy and enabling secure transactions over the internet. But quantum computing will change the field of cryptography forever. It is not known when the first fully functional quantum computer will be built, but we need to be prepared for it. By using the principles of quantum mechanics, quantum computers can perform calculations that are infeasible for classical computers. This gives a significant threat to the security of widely used cryptographic schemes, such as the Rivest-Shamir-Adleman algorithm (RSA).

In order to address this threat, the National Institute of Standards and Technology (NIST) brings a competition to life, to develop new cryptographic algorithms that are secure against quantum attacks. One of the candidates is the Bit Flipping Key Encapsulation (BIKE) algorithm, a post-quantum cryptographic scheme based on the hardness of problems from coding theory. Through the high level of security against not only quantum attacks but also classical adversaries, BIKE is one out of three promising candidates in the last round of NIST's competition. Because BIKE is one of the less analyzed candidates, it was chosen for this thesis.

To further analyze the security of a cryptographic system, such as BIKE, a theorem prover can be used. For proofing the security and completeness of BIKE, the foundation was laid in this thesis, with an implementation of BIKE in the EasyCrypt framework. In the following, the BIKE algorithm gets explained in detail and implemented within the framework EasyCrypt, which is specifically designed for the verification of cryptographic algorithms and protocols, providing a high level of automation and support for reasoning about security properties.

In this thesis, the aim is to contribute to the ongoing research in post-quantum cryptography and formal verification of BIKE. The goal is to gain a better understanding of BIKE and the security guarantees, by implementing and analyzing the BIKE algorithm in EasyCrypt.

After this introduction, in chapter 2 the mathematical background is presented, as well as the important notations and definitions. Chapter 3 gives an overview of the cryptographic background, including classical cryptographic attacks and security properties. Furthermore, the RSA algorithm, a widely used asymmetric encryption scheme, is introduced and its vulnerabilities to quantum attacks is discussed. This is then followed by chapter 4 presenting the limitations of RSA using Shor's algorithm on quantun computers. Additionally, the NIST Post-Quantum Cryptography (PQC) competition is introduced, by presenting the candidates and explaining the standardization process with the main criteria. In chapter 5, the BIKE

algorithm is being presented in a detailed manner, including the security claims and the underlying mathematical problems. The implementation of BIKE in the EasyCrypt theorem prover is going to be discussed in chapter 6. Finally, the thesis concludes with a summary of the findings and an outlook on future research directions. The full implementation of the BIKE algorithm in EasyCrypt can be found on GitLab [1].

1.1 Related Work

The BIKE algorithm has been the subject of several research papers and various implementations. The BIKE team themself published implementations, also linked on their website [5]. One of the implementations is in C, which is available on GitHub [45]. Another is done in VHDL, a hardware description language [51]. There is also an implementation of BIKE for the use in the Transport Layer Security (TLS) protocol [24]. Quite a lot of research was done in terms of the decoder for BIKE. If it is the analyses of different decoders [34], or the correlation between the decoder and weak keys [47], many aspects of the decoder were analyzed. A general security discussion about BIKE's security can be found in [46].

For more information about the Standardization process of the NIST PQC competition, the reader is referred to the NIST PQC website [60]. The currently most up-to-date information about the competition and their candidates can be found there, the latest report is from round 3 [61], describing and providing a brief analyses of the candidates.

Apart from EasyCrypt, there are other theorem provers that can be used for the formal verification. One of them is Isabelle [14], where the security properties as well as the correctness of the CRYSTALS-KYBER algorithm were verified [31]. For CRYSTALS-KYBER, a formal specification was done in EasyCrypt as well as an implementation in the Jasmin framework [3]. Additionally, a certified verification of the reference implementation for KYBER was done in another theorem prover called Coq [68]. CRYSTALS-Dilithium has a CMA security proof in the random oracle model of EasyCrypt [6], providing the CRYSTALS family with a solid foundation of automatic verification. A formal verification of *XMSS*, the primary building block for Classic McEliece, was done in EasyCrypt [7]. But also classical verifications are done, such as by BIKE, where team presented an IND-CCA proof on paper [38]. For FALCON an analysis over the key recovery attacks was presented [22]. Some of these papers and analysis and many more are to be found also in the report paper of the competition [61].

Of course, it is of interest to see how BIKE compares to other candidates in the competition. Comparing BIKE's performance with the performance of HQC and Classic McEliece makes sense, since these 3 are the remaining candidates in the competition and are all based on codes. In [32] a comparison between these 3 candidates was done, showing that Classic McEliece has the biggest key sizes of them all as well as a significantly higher need of computational resources for the key generation, making it less suitable for constrained environments such as IoT devices. HQC and BIKE on the other hand demonstrated smaller key and ciphertext sizes, making it more suitable for constrained environments. Overall BIKE and HQC show in this paper a more balanced performance, than Classic McEliece.

2 Preliminaries

2.1 Notation

$$\begin{aligned}
\mathbb{F}_2 &= \text{The finite field over } \{0,1\} \\
\mathbb{Z} &= \text{The set of integers} \\
\mathbb{R} &= \text{The set of real numbers} \\
\mathbb{C} &= \text{The set of complex numbers} \\
|x| \text{ for } x \in \mathbb{C} &= \text{Absolute value of } x \\
mod &= \text{Modulo operator} \\
\equiv &= \text{Equivalence with respect to modulo} \\
\gcd(x,y) &= \text{Greatest common divisor of } x \text{ and } y \\
I &= \text{Identity matrix} \\
A^T &= \text{Transpose of matrix } A \\
A^\dagger &= \text{Complex conjugate transpose of matrix } A \\
|x\rangle &= \text{Quantum state in classical possibility } x \\
\{x\}_q &= \text{Residue class ring with respect to modulo } q \\
||x|| \text{ for } x \in \mathbb{R}^n &= \text{Euclidean norm} \\
\mathcal{O} &= \text{O Notation} \\
\$ &= \text{Uniform distribution} \\
\oplus &= \text{XOR operator} \\
\text{iff} &= \text{if and only if}
\end{aligned}$$

2.2 Mathematical Background

2.2.1 Finite Fields

Fields are algebraic structures consisting of abstractions of familiar number systems, such as the real numbers $\mathbb{R}$, and their essential properties. They are defined by two operations, addition and multiplication, along with the set $\mathbb{F}$, that satisfy the properties:

1. $(\mathbb{F}, +)$ is an abelian group with the additive identity 0
2. $(\mathbb{F}\backslash\{0\}, \cdot)$ is an abelian group with the multiplicative identity 1

3. Distributivity holds: $(a + b) \cdot c = a \cdot c + b \cdot c$ for all $a, b, c \in \mathbb{F}$

Now if the set $\mathbb{F}$ is finite, it is called a finite field, denoted as $\mathbb{F}_q$ with q elements [26].

2.2.2 Groups

A group G is an algebraic structure consisting of a non-empty set of elements together with a binary operation $*$ that satisfies the following four axioms:

1. Closure: If two elements A and B are in G, then there is also $A * B$ in G
2. Associativity: For all $A, B, C \in G$ it holds that $A * (B * C) = (A * B) * C$
3. Identity: There is an $I \in G$ such that for all $A \in G$ it holds that $A * I = I * A = A$
4. Inverse: For each element $A \in G$, there is an inverse element $A^{-1} \in G$, such that $A * A^{-1} = I$

A common group is the multiplicative group, where the binary operator $*$ is the multiplication operator [43].

2.2.3 Cyclic Groups

A cyclic group is a group that is formed by one element A. This element produces the group by calculating $A * A$ over again and adding one $*A$ each time. For the purpose of this thesis the most relevant cyclic group is the group formed by addition under the modulo operator. As an example, we consider the group formed by addition modulo 6, denoted as $(\mathbb{Z}/6\mathbb{Z}, +)$. It therefore consists of the element list $\{0, 1, 2, 3, 4, 5\}$ [43].

2.2.4 Hilbert Space

[43] A Hilbert Space, also called Inner Product Space, is a Vector Space with an Inner Product, which is a function that assigns a scalar to two vectors.

Definition 1 (Vector Space)**.** *A vector space V is a set that is closed under finite vector addition and scalar multiplication, fulfilling the following properties. For each $u, v, w \in V$ and $\alpha, \beta \in \mathbb{R}$ it holds that:*

1. $u + v = v + u$
2. $(u + v) + w = u + (v + w)$
3. $u + 0 = u$
4. $u + (-u) = 0$
5. $1 \cdot u = u$
6. $\alpha(\beta u) = (\alpha\beta)u$
7. $(\alpha + \beta)u = \alpha u + \beta u$

8. $\alpha(u + v) = \alpha u + \alpha v$

Definition 2 (Inner Product)**.** *For a vector space V, a function $\langle \cdot, \cdot \rangle : V \times V \to \mathbb{R}$ is called an inner product if it fulfills the following properties. For each $u, v, w \in V$ and $\alpha \in \mathbb{R}$ it holds that:*

1. $\langle u + v, w \rangle = \langle u, w \rangle + \langle v, w \rangle$
2. $\langle \alpha u, v \rangle = \alpha \langle u, v \rangle$
3. $\langle u, v \rangle = \langle v, u \rangle$
4. $\langle u, u \rangle \geq 0$ *iff* $u = 0$

Every inner product space implies a norm defined as:

$$||u|| = \sqrt{\langle u, u \rangle}$$

A finite-dimensional inner product space over the real numbers is called an Euclidean vector space [10].

2.2.5 Vector Subspace

[43] Given a vector space V over a field F, then is a subset U a subspace of V iff:

1. U is not empty
2. U needs to be closed under vector addition
3. U needs to be closed under vector multiplication

2.2.6 Continuous Fractions

The idea of continues fraction is to represent a rational number in a sequence of whole numbers.

$$[a_0; a_1, ..., a_M]$$

For example $\frac{11}{4}$ will be represented as $[2; 1, 3]$. The algorithm for this conversion works as follows:

1. Compute the integer quotient and append it to the result

$$\frac{11}{4} = 2 + \frac{3}{4}$$

2. Invert the resulting fractional part

$$2 + \frac{1}{\frac{4}{3}}$$

3. Repeat steps 1.-2. with the denominator of the resulting fraction, until the remainder equals to 0
$$2 + \frac{1}{\frac{4}{3}} \Rightarrow 2 + \frac{1}{1 + \frac{1}{3}} \Rightarrow 2 + \frac{1}{1 + \frac{1}{\frac{3}{1}}} \Rightarrow 2 + \frac{1}{1 + \frac{1}{3+0}}$$
4. Read of the solution from the result or from the first entry of each fraction
$$[2; 1, 3]$$

So $[a_0; a_1, ..., a_M]$ would be as fraction $a_0 + \frac{1}{a_1 + \frac{1}{\cdots + \frac{1}{a_M + 0}}}$.

This algorithm always terminates, since each iteration the numbers are decreasing, given that the number is rational. When given a fraction $\psi = \frac{a}{b}$ where a, b are Integers with the length L in bit representation, the overall runtime is $\mathcal{O}(L^3)$ in the classical setting. The steps 1-2. are $\mathcal{O}(L)$ times repeated and each use $\mathcal{O}(L^2)$ elementary arithmetic gates [43].

2.2.7 Linear Codes

A $(n; k)$-linear code C is a k-dimensional vector subspace, with a length of n, dimension k and co-dimension $r = (n - k)$. It is a binary $(n; k)$-linear code, if the elements of the vector subspace are of $\mathbb{F}_2^n$ A vector $c \in C$ is called a codeword [38].

Therefore, it needs to fulfill the conditions for vector subspaces. For example, the subset $C_1 = \{000, 111\}$ of $\mathbb{F}_2^3$ is a $(3; 1)$-linear code. The subset $C = \{000, 110, 011\}$ of $\mathbb{F}_2^3$ is not a linear code, because $110 + 011 = 101$, which is not in C, and therefore not close under vector addition.

2.2.8 Generator and Parity-Check Matrices

A Matrix $G \in \mathbb{F}_2^{k \times n}$ is called a generator matrix of a $(n; k)$-linear code C if for all vectors $m \in \mathbb{F}_2^k$ it holds that $C = \{mG\}$.

A Matrix $H \in \mathbb{F}^{(n-k) \times n}$ is called a parity-check matrix of a $(n; k)$-linear code C if for all vectors $c \in C$ it holds that $cH^T = 0$.

When the vector c is in $\mathbb{F}_2^n$ but not in C, then $cH^T = s$, with $s \in \mathbb{F}_2^r$ and called a syndrome [38].

2.2.9 Lattices

Let $b_1, b_2, ..., b_n$ be linear independent vectors in the m-dimensional Euclidean vector space $\mathbb{R}^m$. A lattice is the set
$$\Gamma(b_1, b_2, ..., b_n) = \left\{ \sum_{i=1}^{n} x_i \cdot b_i : x_i \in \mathbb{Z} \right\}$$
where n is called the rank and m is the dimension. The vectors $b_1, b_2, ..., b_n$ are the lattice basis and can be represented as a Matrix
$$B = [b_1, b_2, ..., b_n] \in \mathbb{R}^{m \times n}$$

"Graphically, a lattice can be described as the set of intersection points of an infinite, regular n-dimensional grid [37]." Let this be shown by the vectors $x = \begin{pmatrix}1\\2\end{pmatrix}$ and $y = \begin{pmatrix}1\\-1\end{pmatrix}$.

2.2.10 Euler's Totient Function

The Euler's totient function is defined as

$$\phi(n) := |\{a \in \mathbb{N} : 1 \leq a \leq n \wedge gcd(a, n) = 1\}|$$

In essence, the function enumerates the positive integers up to a specified value of n, which are relatively prime to n [43].

2.2.11 Euler's Theorem

For all coprime $a,\ n \in \mathbb{N}$ applies, that

$$a^{\phi(n)} \equiv 1\ (mod\, n)$$

where ϕ is the Euler's totient function [33].

2.2.12 Chinese Remainder Theorem

Given pairwise coprime positiv integers $m_1, \cdots, m_n$, then for any given integers $a_1, \cdots, a_n$ there is an integer x such that $x \equiv a_i\ (mod\, m_i)$ for all $i \in \{1, \cdots, n\}$. Any two solutions x_1 and x_2 are congruent modulo $M = \prod_{i=0}^{n} m_i$ [43].

Therefore it can be followed that if m_1, m_2 are these coprime integers, then it holds for integers x, a that $x \equiv a\ (mod\, m_1)$ and $x \equiv a\ (mod\, m_2)$, if $x \equiv a\ (mod\, m_1 \cdot m_2)$ holds [28].

2.2.13 Hash Functions

A hash function is a function $h : I \to O$ with I usually in $\{0,1\}^n$ and $O \in \{0,1\}^m$, where $m > 0$ and usually fixed and $0 \leq n$. If it meets some additional requirements (as detailed below), it can be used for cryptographic applications and is then known as a cryptographic hash function.

One Way Hash Functions (OWHF)

In order for a hash function to fulfill the One Way property, also called pre-image resistance, these three additional properties need to hold. Given a hash function H and

1. x (any given input), it is easy to compute message $H(x)$
2. $H(x)$, it is computationally infeasible to find x
3. $H(x)$, it is computationally infeasible to find x and x' such that $H(x) = H(x')$

Collision Resistant Hash Functions (CRHF)

A collision free hash function satisfies all OWHF properties and the additional property that for a given H, it is computationally infeasible to find a pair (x, y) such that $H(x) = H(y)$.

Universal One Way Hash Functions (UOWHF)

Given are a polynomial time adversary A and a finite collection U of Hash functions. A gets an input x, a Hash function H and the corresponding output k. Now A has to find a y such that $H(x) = H(y)$, in order to find a collision. U is then called UOWHF, if the probability for A to find y is negligible [56].

2.2.14 Quantum Fourier Transform

The Quantum Fourier Transform is a suitable tool for the extraction of periodic features of wave functions and defined as

$$QFT_q|x\rangle = \frac{1}{\sqrt{q}} \sum_{x'=0}^{q-1} e^{2\pi i x x'/q}|x'\rangle$$

where q is the length of x, and x is element of the Hilbert space. Written as a unitary matrix the QFT is defined as

$$QFT_N = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & w^1 & w^2 & \cdots & w^{N-1} \\ 1 & w^2 & w^4 & \cdots & w^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w^{N-1} & w^{2(N-1)} & \cdots & w^{(N-1)(N-1)} \end{bmatrix}$$

where N is the dimension of the matrix $(N \times N)$ and $w = e^{\frac{2\pi i}{N}}$ [70].

3 Cryptographic Background

At its core, cryptography is the process of encrypting information so that only authorized parties can access it. In the following, these parties are represented by Alice and Bob, wanting nothing more than to send each other secure messages. On the other hand there is Eve, a malicious adversary. She wants to see and read what Alice and Bob send to each other. Alice and Bob need to adapt their communication in such a way that Eve can not read the messages Alice and Bob send each other. This needs to hold even when Eve has access to the data Alice and Bob send each other, a so-called ciphertext. How can Alice and Bob still communicate, especially since the ciphertext does not have to be the same as the original message, the so-called plaintext? This question is central to understanding the principles of modern cryptography. The key lies in the shared knowledge between Alice and Bob. This shared information allows them to encrypt and decrypt messages in a way that preserves the original plaintext, even though the ciphertext may look completely different. The process ensures that their communication remains secure, even if Eve intercepts the ciphertext.

Modern cryptography can be divided into two main areas. Symmetric key cryptography uses the same secret key for both encryption and decryption, while asymmetric or public key cryptography uses separate but mathematically related keys for these processes. Hash functions, which produce randomized outputs from inputs, are a critical part of many of these schemes. In addition, key exchange protocols enable secure communication over insecure channels and form the basis of many internet security protocols. Their main purpose is to allow Alice and Bob to securely establish a secret key, for further communication [52]. An example for this would be the Diffie-Hellman key exchange protocol [35]. A group of key exchange protocols are key encapsulation mechanism (KEM). Such a KEM has typically 3 parts, a Key Generation, where the secret key is being generated, the Encapsulation where the secret key is wrapped, and the Decapsulation, where the key is unwrapped.

Additionally, there is also the concept of verification. Alice maybe do not want to send certain information to Bob, but rather want to verify Bob that he is indeed talking with Alice. Eve could potentially infiltrate the communication between Alice and Bob, and pretend to be Alice in front of Bob. For this signature schemes were invented, which sole purpose is to verify the communication partner. Similar to a KEM, a signature scheme typically consists of a Key Generation algorithm, a Signing algorithm which does not produce a message but a signature, and a Verify algorithm, which verifies the given signature and only if it is a correct one [52].

3.1 Classical Cryptographic Attacks

As mentioned before, one possibility for Eve is to compromise the communication channel between Alice and Bob. This is a technique known as a man-in-the-middle attack. The following attack scenarios are available for Eve:

1. Interception: Eve can read every message send from or to Alice or Bob

2. Modification: Eve can modify the messages and then forward the changed message

3. Impersonation: Eve can pretend to be Alice when communicating with Bob, and vice versa

4. Key Exchange Vulnerability: During key exchange protocols, Eve can intercept and replace public keys, establishing separate secure connections with both Alice and Bob, while they believe they're communicating directly with each other

Man-in-the-middle is not the only way to attack, but it is certainly the most important. In response to the diversity of potential attacks, a range of security properties have been invented to verify and ensure the robustness of cryptographic algorithms [17].

Semantic Secure

When an adversary only gets neglectable information for the decoding of the plaintext out of the ciphertext, the corresponding encryption scheme is called semantic secure [9].

Chosen Plaintext Attack (CPA)

Chosen Plaintext Attack means that a malicious adversary is capable of obtaining ciphertexts of any chosen plaintext. On other words, Eve has access to the Encapsulation algorithm, which is in the case of public key encryption (PKE) always given, since the public key which is used for decryption is publicly known. And under this assumption, it is still not possible for the adversary to gain important information in order to decrypt any given ciphertext [9].

Chosen Ciphertext Attack (CCA)

Chosen Ciphertext Attack is similar to CPA under one critical change. Instead of having access to the Encapsulation algorithm, the adversary has access to the Decapsulation algorithm. Important to note is that having access, does not mean that Eve sees the whole algorithm, but rather has a black box, where she can input a desired text and get the output of the algorithm without seeing any details of the computation. Some resources may call this the Decryption Oracle, since it is not known how it works, but gives the desired output. So a cryptographic scheme is secure under CCA, if an adversary can not gain any important information by having access to the Decapsulation algorithm [9].

Indistinguishability (IND)

The prefix IND can be added to the CPA as well as the CCA security definition, and stands for indistinguishability. For this Eve and Alice play a little game, which is typical for security analysis. The game is for IND-CPA, but it can be modified to IND-CCA by modifying the CPA security properties to the CCA security properties:

1. Eve sends two arbitrary plaintexts to the Encryption algorithm.

2. A random bit is generated, choosing which plaintext gets encrypted by the oracle.

3. The chosen plaintext get encrypted and send back to Eve.

4. Eve does not know which plaintext got chosen to encrypt. Her task is now to determine to which plaintext the given ciphertext belongs.

Random guessing would give a 50% chance of being correct. So the given scheme is IND-CPA secure, if Eve can not guess significantly better than random guessing [9].

Existentially Unforgeable

This is a security property just for signature schemes. It is not possible for an adversary to produce a signature for any message, such that the verification algorithm would accept it. The message to sign can be chosen by the adversary. The property can be extended by the suffix CMA, standing for Chosen Message Attack, by which the adversary also gets access to arbitrary many pairs of messages and their signing [9].

There are many more variants of the shown properties, but these are the most important ones for today's use. Note that IND-CPA is the bare minimum a cryptographic scheme needs, since it is already the case for a public key encryption [2]. Cryptography is not just a technical discipline; it has profound implications for privacy, security and freedom in the digital age. It enables individuals to protect their personal information, businesses to secure their transactions and intellectual property, and governments to protect national secrets. As such, cryptography is often at the center of debates about the balance between security and data privacy, with policymakers grappling with questions of encryption regulation and back-doors. Our digital communication is mostly secured and encrypted, to prevent information being leaked or even manipulated. Through the "s" in https, which stands for "secure", Web browser communication is made save. The https protocol uses the Transport Layer Security Protocol (TLS) to encrypt the data send from the browser to the server TLS often uses Rivest-Shamir-Adleman [12] or other asymmetric encryption schemes. But also symmetric key cryptography like Advanced Encryption Standard (AES)[19] are mentioned and can be used [50].

The focus of this thesis is on signature schemes and key encapsulation mechanisms. Their main purpose is not so send encrypted text or long messages. Digital signatures are used to verify a user. KEM's purpose is the encrypted transfer of a secret key to each party, which then is afterwards used for the encryption of data.

3.2 RSA

[36] RSA is a asymmetric encryption scheme for key encapsulation. Starting on Alice's side, where she needs to compute the public key and her secret (private) key.

Key Generation:

1. Generate two distinct prime number p and q
2. Compute $N = p \cdot q$ as well as $\phi(N) = (p-1) \cdot (q-1)$
3. Select an $e \in \mathbb{N}$ with $1 < e < \phi(N)$ and $gcd(e, \phi(N)) = 1$

4. Compute $d \in \mathbb{N}$ with $1 < d < \phi(N)$ and $e \cdot d \equiv 1 \,(mod\, N)$. A possible algorithm to use for the calculation of d is the extended euclidean algorithm.

Now Alice has a public key consisting of (N, e) and a secret key (d). She distributes the public key, to anyone who wants to send data to her, in our case this is Bob.

Encryption:

Now if Bob wants to send data to Alice, he can do it with the public key as follows:
After obtaining the public key, Bob can encrypt the message m by computing

$$c \equiv m^e \,(mod\, N)$$

The message m needs to be an integer in the range of 0 to $N - 1$, else the decryption will not work. Then all Bob has to do is to send Alice the cipher text c.

Decryption:

After Alice reviews the cipher text c from Bob she can decrypt the original message m by computing

$$m \equiv c^d \,(mod\, N)$$

Proof of correctness:

[12] In the following, it is shown that from the cipher text c, Alice can compute the original message m with the secret key d and publicly known N. The cipher text is computed by

$$c \equiv m^e \,(mod\, N)$$

The message is then computed in the decryption phase using the ciphertext.

$$m \equiv c^d \equiv m^{e^d} \equiv m^{ed} \,(mod\, N)$$

Since $e \cdot d \equiv 1 \,(mod\ \phi(N))$ there exists a natural numbers k, such that $e \cdot d = 1 + k \cdot \phi(N)$. Inserting this into the above formula results in

$$m^{ed} \equiv m^{1+k*\phi(N)} \equiv m \cdot m^{k^{\phi(N)}} \,(mod\, N)$$

So it holds that $m \equiv m \cdot m^{k^{\phi(N)}} \,(mod\, N)$ with $m < N = p \cdot q$ and p and q being prime numbers. Therefore the $gcd(N, m)$ can only be 1 or p or q. When $gcd(N, m) = 1$ Euler's theorem holds, resulting in

$$m \equiv m \cdot 1^k \,(mod\, N)$$

Else $gcd(N, m) = p$ or q. The proof of q is the same as for p, by just changing the letters therefore only the proof for p is shown. By using the Chinese remainder theorem, it remains to show that $m = m \cdot m^{k^{\phi(N)}} \,(mod\, p)$ and $m = m \cdot m^{k^{\phi(N)}} \,(mod\, q)$. The first case holds, since $m = hp$ for some $h > 0$, which is equal to 0 modulo p. For the second case it is so far:

$$m = m \cdot m^{k^{\phi(N)}} \,(mod\, q)$$

$$\Rightarrow m = m \cdot m^{k \cdot (p-1) \cdot (q-1)} \, (mod \, q)$$

$$\Rightarrow m = m \cdot m^{(q-1)^{k \cdot (p-1)}} \, (mod \, q)$$

$$\Rightarrow m = m \cdot 1^{k \cdot (p-1)} \, (mod \, q)$$

$$\Rightarrow m = m \, (mod \, q)$$

So it is shown that by encrypting and decrypting a message m, it does not get changed and outputs the original message m.

RSA could also be used as a digital signature. To sign a message m Alice needs to hash the message and then exponentiate with d.

$$sign \equiv h(m)^d \, (mod \, N)$$

To verify the signature $sign$, Bob has to compute

$$sign^e \, (mod \, N)$$

and compare it to $h(m)$, which he also gets from Alice. If they are the same, then Alice's signature is verified.

4 Introduction to Post-Quantum Cryptography (PQC)

4.1 Limitations of RSA

The fundamental mathematical issue underlying RSA is the computation of integer factorization, which is a difficult computational problem. With other problems like discrete logarithm problem, which is used for ElGamal, and elliptic curve problem, they form the basis for classical cryptographic systems. To this day no polynomial time algorithm was presented, which can solve integer factorization in polynomial time [67].

Theorem 4.1.1 (Integer Factorisation Problem)**.** *[43] Given a positive natural number $N > 1$, what are the prime numbers $p_1, ..., p_k$ such that*

$$\prod_{i=1}^{k} p_i = N$$

Definition 3 (Discrete Logarithm Problem)**.** *[43] Given a cyclic group G, a generator g for G, and some element x of G, find a number n so that*

$$x = g^n$$

It is possible that some numbers may be solved in a relatively short time, however, this does not apply in general. As an example RSA-768 was factored 2009 [30] as well as RSA-640 a few years earlier, where the 768 and 640 stand for a 768 or 640 bit long number in binary notation. So it is recommended to use an N between 1024 and 2048 bit length, where a too high N makes RSA inefficient and a too low N makes it insecure [12]. Nevertheless, there is a prevalent assumption that factoring an integer is a challenging problem. However, there is currently no established and published proof of its hardness, and it remains to be proven whether integer factorization is indeed NP-hard [67]. The fastest algorithm for integer factorization to this day for large numbers is the "General Number Field Sieve". It has a runtime of $((c + o(1))n^{1/3}log^{2/3}n)$ for $c < 2$. Therefore, it is not practical or even possible to solve integer factorization on large numbers [13].

Classical computers do not have enough computational power, to solve these mathematical problems, like integer factorization or discrete logarithm, in polynomial time. Given that the implementation and scheme is secure, the crypto systems, which use these problems as a basis, can be also secure.

When implementing such a cryptographic system, it is necessary to pay attention to additional specification of the parameters in order to prevent attacks other than the solution of integer factoring. As with RSA, there are several ways to break RSA that must be considered. For example when using multiple instances of RSA, each with a separate N, it shouldn't use the same prime number p for both. When doing so one could easily compute p by calculating $gcd(N_1, N_2)$ and therefore break the scheme. Another consideration would be, not to pick a tiny e, or otherwise someone could compute m by the Chinese remainder theorem. There are many more ways, and these were just some examples [12].

When it comes to the security of cryptographic schemes, the key size is crucial. In the case of RSA, the key size scales bad with the security compared to schemes based on elliptic curves. "This means that a desired security level can be attained with significantly smaller keys in elliptic curve systems than is possible with their RSA counterparts. For example, it is generally accepted that a 160-bit elliptic curve key provides the same level of security as a 1024-bit RSA key" [26]. The advantages gained from smaller key sizes, include speed and efficiency of the algorithm, but also lower power consumption, bandwidth and storage. A good introduction to elliptic curve cryptography can be found in the book "Guide to Elliptic Curve Cryptography" [26].

Exploiting the computational complexity of factorisation and discrete logarithms proved to be useful in the past [67]. With the scientific progress in the field of quantum computers, it is crucial to develop security systems strong enough to withhold technical feasable capabilities emerging from possible future breakthroughs. Even if it is not clear when fully functional quantum computer will emerge, any threats that could come quantum computing should be considered [58]. "It is clear that a quantum computer leveraging the quantum mechanics properties of superposition and entanglement could carry out quite complex computations in a matter of hours, which is not possible for classical computers in years [67]."

This is due to the fact, that quantum computers do not utilize classical bits, with two possible states, but instead make use of so called qubits (for "quantum bit", see below). Consequently, each classical bit is swapped out with a qubit, which is capable of existing in an infinite number of states, in contrast to the two possible states of a classical bit. Combined with the concept of superposition and entanglement of quantum physics, quantum computers gain much more computational power [28].

4.2 Quantum Computing

Quantum computers, unlike classical computers, which are using bits (0 or 1), rely on quantum bits or qubits, which can exist in a superposition of 0 and 1. This allows qubits to represent multiple states simultaneously, expressed in Dirac or ket-notation:

$$\alpha \cdot |0\rangle + \beta \cdot |1\rangle$$

with the condition that $|\alpha|^2 + |\beta|^2 = 1$. An example is a qubit in superposition:

$$\frac{1}{\sqrt{2}} \cdot |0\rangle + \frac{1}{\sqrt{2}} \cdot |1\rangle$$

where the probabilities of measuring 0 or 1 are each equal to $\frac{1}{2} = \left|\frac{1}{\sqrt{2}}\right|^2$.
In quantum computing, operations are performed using unitary matrices The Hadamard gate (H), for example, places the qubit in superposition:

$$H \cdot |0\rangle = \frac{1}{\sqrt{2}} \cdot (|0\rangle + |1\rangle)$$

Unitary matrices preserve the conditions required for valid quantum states, and applying them to qubits allows for quantum computations.

Shor's Algorithm

Shor's algorithm, a quantum mechanical algorithm factorises integers 4.1, the backbone of the RSA key encryption, in polynomial time relative to the input length. The algorithm was introduced by Peter Shor in 1994 and utilizes a reduction of order finding.

Given a random $x \in \{1, \cdots, N-1\}$ and a N, the goal is to find the order r of x, such that r is the smallest number for which holds $x^r \equiv 1 \, (mod \, N)$. The algorithm starts by picking such a random number x, where the order r is to be found. Then the algorithm creates a superposition of all possible states on the input wires, which is done by the Hadamard gate, where the number of input wires $|a|$ is sufficiently larger than $2N^2$. Now x^a will be computed, where a are the states on the input wire, therefore the superposition of all possible states. After this the algorithm applies the quantum Fourier transform, which is a quantum version of the classical Fourier transform, extracting the period of the function $f(x) = x^a (mod N)$. Now by measuring the output of the quantum Fourier transform, the period r can be extracted, with a certain probability, by computation of the continues fraction. When the period is found, the algorithm can compute the factors of N by computing $gcd(x^{r/2} - 1, N)$ and $gcd(x^{r/2} + 1, N)$. Therefore, this algorithm can factor integers in an efficient time, breaking the RSA scheme [55].

A more detailed explanation of Shor's algorithm and the surrounding concepts is written down in Appendix A. For any more information about quantum computing, the book "Quantum Computation and Quantum Information"[43] is recommended.

4.3 NIST

4.3.1 PQC Standardization Process

The National Institute of Standards and Technology (NIST) is a governmental institution in the United States of America, which is responsible for the development of standards to enhance economic security and improve quality of life. This also includes cryptographic schemes. One example for cryptography is the standardization of the Secure Hash Algorithms Family (SHA), a series of Hash functions, where SHA-256 and SHA-384 are part of. These are also used by Bit Flipping Key Encapsulation (BIKE) [16]. The Post Quantum Cryptography (PQC) Standardization Process was first called out in 2016 by NIST [58]. As a first step, NIST invited the public to comment on the draft requirements and criteria for the submission and evaluation of candidate algorithms. These requirements were posted for everyone online, and participants had time until November 2017 to submit their algorithm [57]. "NIST received 82 submissions from 6 continents and 25 countries. Among them, 69 submissions were announced as the first round candidates. After about one year analysis and evaluation, 26 candidates made it to the second round, which was announced in January 2019 [16]." The goal was not to find the one best algorithm, but rather to find a selection of algorithm suitable for different fields of use [59]. The selection was based on three criteria.

1. Security

The goal of this competition is to find suitable algorithm for use in a wide variety of protocols, such as TLS mentioned in section 3.2 or Secure Shell (SSH). These algorithms need to be secure with regard to quantum computers, but also in classical setting. Therefore, this property is the most important. For the classical setting there are already defined security properties as mentioned in chapter 3, which are taken into account when evaluating the schemes [53]. KEM's need to be at least IND-CCA secure, when they are a general-use encryption and key-establishment scheme. For short-lived use cases they even need to be IND-CPA secure. "Digital signature schemes were required to provide existentially unforgeable signatures with respect to an adaptive chosen message attack (EUF-CMA security) [42]." The quantum setting on the other hand is more difficult. Two uncertainties make security evaluating for the quantum setting difficult. First the discovery of new quantum algorithm, which can brake currently assumed secure algorithm. And second, the unknown advancements quantum computers will make in the future. What about the speed or memory size these machines will actually have. Do they get insanely difficult and therefore expensive to make and operate? Since the security property in the quantum setting is difficult to categorize, NIST proposed 5 security levels depending on two known algorithms. The AES algorithm and the SHA Hash function. Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for:

Security Level	Requirement
1	key search on a block cipher with a 128-bit key (e.g. AES-128)
2	collision search on a 256-bit hash function (e.g. SHA-256/SHA3-256)
3	key search on a block cipher with a 192-bit key (e.g. AES-192)
4	collision search on a 384-bit hash function (e.g. SHA-384/SHA3-384)
5	key search on a block cipher with a 256-bit key (e.g. AES-256)

Table 1: Security Levels and Corresponding Requirements for the Quantum Setting

This means for Level 3 that the algorithm needs at least as much computational power as to brute force AES with a key size of 192-bits. For a classical computer this is completely infeasible since there are 2^{192} different keys. Grover's algorithm would speed up the process for a quantum computer quadratic, therefore making the search faster by now $2^{\sqrt{192}} = 2^{96}$ searches needed. This would still take enormous time and therefore be considered save for now. Unlike RSA, AES can be made save again, by raising the key size high enough. This of course only holds for a quantum computer with unlimited circuit depth, which is not a realistic assumption. NIST calls this parameter MAXDEPTH, which is the number of logical quantum gates used by the computer. "Plausible values for MAXDEPTH range from 240 logical gates (the approximate number of gates that presently envisioned quantum computing architectures are expected to serially perform in a year) through 264 logical gates (the approximate number of gates that current classical computing architectures can perform serially in a decade), to no more than 296 logical gates (the approximate number of gates that atomic scale qubits with speed of light propagation times could perform in a millennium)." [42] In a real application one has to run Grover's algorithm multiple times, making the speedup less drastic, but also the analysis more difficult [63][42]. It can be argued that

security schemes with good scaling properties, such as elliptic curve cryptographic systems, can be beneficial in that they make users safer by raising the key size. This is on the condition that the scheme in question does not present any other security problems.

2. Cost and Performance

The second-most important criterion is the performance and cost estimation, where cost is not meant in money but the cost for the computer to operate the algorithm. Size of the public keys, ciphertext and signatures is as well as the needed size for RAM and the overall code size, are taken into account. Computational efficiency in form of computational time of the algorithm, is also important. And last but not least is the performance in terms of failure rate for decryption. NIST themselves tested the algorithm on "an Intel x64 running Windows or Linux and supporting the GNU Compiler Collection (GCC) compiler", and therefore also required this from the contestants.

3. Algorithm and Implementation Characteristics

In the event of a choice between two algorithms, the latter may be selected on the grounds of its greater flexibility or simplicity. The algorithms that are capable of operating effectively on diverse platforms, or even employ parallelism, will maybe grant priority. Furthermore, simple and elegant designs are preferred, as further analysis will be easier and other people are more likely to work with it. Any further reasons for or against an implementation of the algorithm, will be taken into account, such as intellectual property problems [53][42].

Controversy regarding NIST

But there are also certain concerns of NIST's independence, as it is not self-hosting but part of an official U.S. government department. While NIST claims to be independent, especially in the case of the PQC standardization process, the Dual_EC_DRBG incident shows that such an independence is not always guaranteed, where the NSA's involvement led to the inclusion of a possible backdoor. Although NIST did not create this standard, it was published in their 2006 guidance and only revoked in 2014 after reports revealed NSA interference [62][66]. Given the history of intelligence agencies attempting to compromise digital infrastructure, caution is important and double-checking is never wrong. NIST's efforts in leading global collaboration on post-quantum cryptography are valuable, and their transparency and decision-making throughout the process is essential for trust and accountability [58]. In addition, because of NIST's authority, the competition is taken seriously, and many people are working on it, which is good for the field of post-quantum cryptography.

4.3.2 The Competitors

On 22 July 2020 NIST announced the end of the second round and the seven finalists candidates, with additional 8 candidates for possible advancing into a fourth round. In this round the candidates have time for tweaks in their design and NIST and the cryptography community time for analyzing the algorithm. Here NIST already said that from 3 out of 4 finalists

KEM's, only one will be standardized, since they are all based on the same fundamental concept [61]. This shows that NIST does not want to put all the eggs in one basket and wants to have several algorithms for standardization, making the selection more diverse [2]. This holds also for 2 of 3 Signature schemes namely "CRYSTALS-Dilithium" and "FALCON". Out of these 15 finalists, 4 were then planned to be standardized [2]. And on the 13th of August 2024, 3 of them where approved and the standardization paper have been published, namely "CRYSTALS-KYBER" now called "FIPS 203" [40], "CRYSTALS-Dilithium" now under "FIPS 204" [39] and "SPHINCS+" with "FIPS 205" [41]. "FALCON" as the third signature scheme will also be standardized, but this needs some more time. At the moment round 4 is still active, with four different KEM's and no digital signature scheme. This is because since now only one KEM is going to be standardized, namely "CRYSTALS-KYBER". In this fourth round the participants are BIKE, Classic McEliece, HQC and SIKE, where SIKE was broken by Wouter Castryck and Thomas Decru in 2022 and is therefore out of competition [15]. This happened to many participants, and it remains to be seen if the others are in fact secure [2]. The submitted algorithms can be classified into three categories based on their underlying mathematical principles, namely lattice-based, code-based and others.

Lattice-based cryptography

Based on lattices, many proposed cryptographic systems are based on some kind of lattice problem. The underlying NP-hard problem is "Shortest Vector Problem" (SVP), but for today's cryptographic system some form of "Learning with Errors" (LWE) or "Small Integer Solution" (SIS) is used. The computational complexity of LWE and SIS is given through reduction to this SVP problem [2].

Definition 4 (Shortest Vector Problem)**.** *Given a basis B of a lattice $L \subseteq \mathbb{Z}^n$, the goal is to find a nonzero vector $n \in L$ such that $||u|| = min||v||$ for all $v \in L\backslash\{0\}$.*

Definition 5 (Small Integer Solution)**.** *Given a modulo q, a Matrix $A \in \mathbb{Z}^{n\times m}$, where $n \leq m$ and a real constant v, find a nonzero vector $u \in \mathbb{Z}^m$ such that $Au \equiv 0 \,(mod\, q)$ and $||u|| \leq v$.*

Definition 6 (Learning with Error)**.** *Let $a \in \mathbb{Z}_q^n$ uniformly at random, $e \in \mathbb{Z}_q$ distributed over a given distribution χ, then $A_{s,\chi}$ is the tupel $(a, \langle a, s\rangle + e) \,(mod\, q)$. Now given the $A_{s,\chi}$ and the distribution χ as well as the modulo q, the goal is to find the corresponding s, given an arbitrary number of samples of $A_{s,\chi}$.*

[49]

Code-based cryptography

This type of cryptographic system is based on binary linear-codes, where there are two decision problems, forming the basis of this type of scheme.

Definition 7 ((Decisional) Syndrome Decoding problem)**.** *Given a $(n-k)\times n$ parity-check matrix H for a binary $(n;k)$-linear code C, a vector $y \in \mathbb{F}_2^{n-k}$, and a target hamming weight $t \in \mathbb{N}$, determine whether there exists $x \in \mathbb{F}_2^n$ that satisfies $xH^T = y$ and $|x| \leq t$ [2].*

Definition 8 ((Decisional) Codeword Finding problem)**.** *Given a* $(n-k) \times n$ *parity-check matrix* H *for a binary* $(n;k)$*-linear code* C *and a target hamming weight* $w \in \mathbb{N}$*, determine whether there exists* $x \in \mathbb{F}_2^n$ *that satisfies* $xH^T = 0$ *and* $|x| = w$ *[2].*

The two problems were demonstrated to be NP-complete for a general binary linear code C by Berlekamp, McEliece, and van Tilborg [11]. Also, a search to decision reduction is known, making the corresponding search problem also NP-complete [4]. This doesn't mean that a cryptographic system using these problems are indeed hard to break, especially in the quantum setting. But in the case of Classic McEliece, HQC and BIKE they are secure, each providing IND-CPA secure PKE schemes, with "proofs that depend on (a variant of) one of these two problems [2]."

Other

In addition, numerous other algorithms cannot be assigned to either of the previously described categories. I will present some more algorithms and the basic underlying problem with it, without going into too much detail. SIKE, which was shown to be not secure, is based on SIDH, a Diffie-Hellman key exchange protocol variant, whose security is based on the hardness of finding isogenies between supersingular elliptic curves. Attacking or solving this, is possible by using quantum algorithms for claw-finding and collision-finding [29]. Even thought SIKE is broken, the underlying problem of elliptic curves is still interesting [26].
SPHINCS$^+$ is a stateless hash based cryptographic scheme. It is based on Hash functions, which form a tree based structure so sign a message with the leafs of the three. Many problems were based on multivariate, but none got the final rounds [2]. In the following we will look a bit more at the 4 finalists to be standardized and the 3 remaining candidates in round 4.

	Public-Key Encryption/KEM's	Digital Signatures
Algorithms to be Standardized	CRYSTALS-KYBER	CRYSTALS-Dilithium
		FALCON
		SPHINCS$^+$
Candidates in the Fourth Round	BIKE	
	Classic McEliece	
	HQC	
	SIKE [1]	

Table 2: Overview of current NIST PQC Standardization Process

The following is a brief presentation of the 4 finalists to be standardised, as well as the three unbroken candidates in the fourth round. The strengths and weaknesses of the algorithms are presented in order to give an overview of the current state of the standardisation process and to compare the algorithms.

[1] SIKE is broken and therefore not going to be standardized

CRYSTALS-KYBER

CRYSTALS-KYBER is a lattice-based KEM that uses the Learning With Errors (LWE) problem. It operates by encoding messages as vectors and encrypting them using public matrices and error vectors. It has efficient key generation, encryption and decryption algorithms, but for the cost of relative high size of the public key and ciphertext. Not as bad as in Classic McEliece, but worse than the other competitors.

CRYSTALS-Dilithium

CRYSTALS-Dilithium is a lattice-based digital signature scheme that builds on the Fiat-Shamir protocol with aborts paradigm. It uses the Module-LWE problem, where signatures are created by transforming errors in a lattice to hide the secret key. Just like CRYSTALS-KYBER, the algorithm is very efficient and good for high-performance environments. The signatures are also short compared to other algorithm making it an overall good scheme. The key sizes are not the shortest, but competitive enough.

SPHINCS$^+$

SPHINCS$^+$ is a stateless hash-based signature scheme. Instead of relying on complex mathematical problems like factoring, it uses one-time and few-time signature schemes, which are fast, along with hash functions to guarantee security. Additionally, it is stateless, meaning less storage needed for the algorithm, reducing the security risk. The hash functions used by SPHINCS$^+$ are also well-known and assumed to be safe against quantum attacks. On the other hand it produces large signatures and has a slow signing process, compared to others.

FALCON

FALCON is another lattice-based signature scheme, but it uses the NTRU problem and Fourier sampling to achieve compact signatures. FALCON trades ease of implementation for smaller signature sizes and faster signature generation, making it ideal for applications with bandwidth constraints.

Classic McEliece

Classic McEliece is a code-based key encryption scheme using binary Goppa codes. It has a long history of resisting cryptanalysis, making it highly secure in the classical setting. However, its public key sizes are notoriously large, and the key generation is slow, making it not suitable for every use case. On the other hand is the size of the ciphertext minimal compared to most PQC candidates.

HQC

HQC is a KEM based on quasi-cyclic moderate density parity-check (QC-MDPC) codes. It encrypts using an LWE-like (learning with error) protocol, where ciphertexts are formed with a mix of random polynomials. A positive aspect is the fast and efficient decryption, to a cost of the ciphertext length compared to similar working schemes like BIKE.

BIKE (Bit Flipping Key Encapsulation)

BIKE is a KEM based on QC-MDPC codes 5.1 and uses a bit-flipping decoder to recover the secret message. BIKE is designed for efficient key encapsulation and decryption using moderate-density parity-check codes. Public key and ciphertext are competitive to other good schemes, and also the decryption is fast. A downside is the complex key generation, but more importantly the possibility to fail the decryption. This chance is very low, but it is to note.

For more information about the decision process, the candidates or detailed security analysis, look at the official paper from NIST to the end of round 3. Many other papers and analyses by other people in the cryptographic world are also mentioned there [2].

5 BIKE

The Bit Flipping Key Encapsulation Algorithm is one of the 3 remaining finalists in round 4 of the NIST PQC Standardization Process and was published by a large group of international scientist. It is categorized as a key encapsulation mechanism based on the Niederreiter cryptographic system, which is based on codes. BIKE itself makes use of Quasi-Cyclic Moderate Density Parity Check codes (QC-MDPC codes). The scientific team around BIKE also proposed a IND-CCA proof, under four prerequisites, see section 5.8, and uses the Fujisaki-Okamoto transformation [23] to make an IND-CCA KEM out of an IND-CPA PKE [44]. Besides the typical Encapsulation, Decapsulation and Key Generation, BIKE also has dedicated decoder, which can be swapped out with careful consideration. The system is designed for use with synchronous communication protocols like TLS, with ephemeral keys, i.e. with a fresh public/private key pair for every key exchange session. In particular, decapsulation with a given private key should be allowed only once. Such usage model provides forward secrecy. The following will mainly fokus on the round 4 submission paper of BIKE [38].

5.1 QC-MDPC codes

Definition 9 (Circulant Matrices)**.** *A circulant matrix is a square matrix where each row is the rotation of one element to the right of the previous row.*

Now such a circulant Matrix A, which is completely defined by the first row $(a_0, a_1, \cdots, a_{r-1})$ can be mapped to a polynomial $\phi(A) = a_0 + a_1 \cdot X + \cdots + a_{r-1} \cdot X^{r-1}$ in a polynomial ring $\mathcal{R} = \mathbb{F}_2[X]/[X^r - 1]$. The matrix operation now can be seen as polynomial operations, where the transpose operation is defined for an $a = a_0 + a_1 \cdot X + \cdots + a_{r-1} \cdot X^{r-1}$ as $a^T = a_0 + a_{r-1} \cdot X + \cdots + a_1 \cdot X^{r-1}$.

Definition 10 (Block-Circulant Matrices)**.** *A block-circulant matrix is a circulant matrix formed by circulant matrices of the same size. The order is the size of the circulant matrices and the index the number of circulant matrices in a row of the block-circulant matrix.*

Definition 11 (Quasi-Cyclic Codes)**.** *A quasi-cyclic code (QC code) is a code which uses a block-circulant matrix as generator matrix. For a $(n_0; k_0)$-QC code it holds that the length is $n = n_0 \cdot r$ and the dimension is $k = k_0 \cdot r$, where r is the order of the block-circulant/generator matrix.*

Definition 12 (Quasi-Cyclic Moderate-Density-Parity-Check Codes)**.** *A $(n_0; k_0; r; w)$-QC-MDPC code is a $(n_0; k_0)$ quasi-cyclic code, that accepts a parity-check matrix with constant row weight $w = \mathcal{O}(\sqrt{n})$.*

For the setting of BIKE it is assumed that these matrices are binary.

5.2 Notation and Specification

$$\begin{aligned}\mathcal{R} &= \text{Cyclic polynomial ring } \mathbb{F}_2[X]/(X^r - 1)\\ \mathcal{H}_w &= \{(h_0; h_1) \in \mathcal{R}^2 : |h_0| = |h_1| = w = 2\}\\ \mathcal{E}_t &= \{(e_0; e_1) \in \mathcal{R}^2 : |e_0| + |e_1| = t\}\\ M &= \{0,1\}^l\\ K &= \{0,1\}^l\end{aligned}$$

Four parameters need to be selected for BIKE.

1. The block size r, which should be a prime number such that 2 is primitive modulo r.
2. The row weight w, such that w is an even positive integer such that $w/2$ is odd.
3. The error weight t
4. The message and shared secret size l

For the three security levels 1, 3 and 5 the BIKE team proposed the following selection. "For all security levels, the key length parameter is fixed to l = 256 [38]."

Security	r	w	t
Level 1	12,323	142	134
Level 3	24,659	206	199
Level 5	40,973	274	264

Table 3: Proposed parameter selection for BIKE

The parameter r needs to be selected as prime to prevent squaring attacks. It is also important to choose r, such that 2 is primitive modulo r, as this prevents any potential structure. Additionally, one should look out for r, where the hamming weight of $(r-2)$ is small. This makes the polynomial inversion algorithm [20] more efficient. In the case of the proposed parameter selection, none $|(r-2)|$ is greater than 5. The result of this is, that every polynomial h of even weight in the cyclic polynomial ring $\mathbb{F}_2[X]/(X^r - 1)$ is invertible, which is necessary for the Key Generation. So it is additional necessary that w is even, so that $|h| = w/2$ is uneven. If it is desired to use a custom selection of parameters, one should be careful and consider the mentioned properties. Additionally, one should pick l, w, t and r to be compliant with the security properties as defined in section 5.8.

5.3 Key Generation

The key generation algorithm is generating the secret key consisting of the tuple (h_0, h_1, σ) and the public key h. It is executed first by the initiating party, sending the public key to the other communication party. h_0 and h_1 are picked out of the set $\mathcal{H}_w$, with the distribution

D, described in section 5.7. h is then calculated by inverting h_0 and multiplying it with h_1. Since inverting a polynomial is time-consuming, the BIKE team presented an efficient polynomial inversion algorithm suitable for this use case, as seen in section 6.5.

KeyGen : $() \mapsto (h_0, h_1, \sigma), h$
Output: $(h_0, h_1, \sigma) \in \mathcal{H}_w \times \mathcal{M}, h \in \mathcal{R}$
1: $(h_0, h_1) \xleftarrow{D} \mathcal{H}_w$ [1]
2: $h \leftarrow h_1 h_0^{-1}$
3: $\sigma \xleftarrow{\$} \mathcal{M}$

5.4 Encapsulation

The Encapsulation algorithm takes the public key and generates a ciphertext c and the additional secret key K. This will be executed by the other communication party, therefore not by the initiating party. The ciphertext needs to be sent to the initiating party. $\mathbf{H}, \mathbf{L}$ and $\mathbf{K}$ are hash functions, based on the SHA family, which are defined in section 5.7.

Encapsulation : $h \mapsto K, c$
Input: $h \in \mathcal{R}$
Output: $K \in \mathcal{K}, c \in \mathcal{R} \times \mathcal{M}$
1: $m \xleftarrow{\$} \mathcal{M}$
2: $(e_0, e_1) \leftarrow \mathbf{H}(m)$
3: $c \leftarrow (e_0 + e_1 h, m \oplus \mathbf{L}(e_0, e_1))$
4: $K \leftarrow \mathbf{K}(m, c)$

5.5 Decapsulation

The Decapsulation algorithm takes the secret key tuple (h_0, h_1, σ) as well as the ciphertext c and also generates the additional secret key K. This is the last step of the key encapsulation mechanism, done by the initiating party. For practical use it is important that the algorithm refuses to decapsulate more than one incoming ciphertext, for one given key pair.

Decapsulation : $(h_0, h_1, \sigma), c \mapsto K$
Input: $(h_0, h_1, \sigma) \in \mathcal{H}_w \times \mathcal{M}, c \in \mathcal{R} \times \mathcal{M}$
Output: $K \in \mathcal{K}$
1: $e' \leftarrow \text{decoder}(c_0 h_0, h_0, h_1)$ [2]
2: $m' \leftarrow c_1 \oplus \mathbf{L}(e')$
3: **if** $e' = \mathbf{H}(m')$ **then** $K \leftarrow \mathbf{K}(m', c)$ **else** $K \leftarrow \mathbf{K}(\sigma, c)$

[1] D is a distribution, which is defined in the "Distribution D and Hash Function" section 5.7.

[2] Since the decoder can have a decoding failures, a special element is needed. If the decoder fails, the input for Hash $\mathbf{L}$ is then (0,0).

The decoder plays a crucial role in the Decapsulation algorithm, as it calculates the error $e = e'$ from the syndrome $s = c_0 h_0$ in polynomial form and the parity-check matrix H represented by the two polynomials h_0 and h_1, where it holds that $s = eH^T$. Therefore, the decoder solves the problem of Syndrome Decoding in a (2,1,r,w)-QC-MDPC code, see definition 13. Now both communication partner have the same shared secret key K and continue there conversation based on this secret key.

5.6 Decoder

The scientific team responsible for BIKE proposed different decoders throughout the competition. Its purpose is to solve the problem of syndrome decoding 13. This is based on the decisional Syndrome Decoding Problem, shown to be NP-complete. Three main criteria are influencing the design of such a decoder:

1. A sufficient low decoding failure rate (DFR), satisfying the security requirements of the usage of the KEM
2. A fixed runtime, not just an upper bound runtime, to avoid side-channel attacks
3. A performance, which is suitable for the desired target platform

In the following the latest two decoders being presented are going to be explained in more detail. There are many more decoder algorithm, some better at certain properties but therefore weaker in others.

Black-Gray-Flip

The BGF decoder (see page 27) starts with a loop, iterating over the number of iterations NbIter. In each loop pass, one Black-Gray-Flip iteration (BFIter) is done after calculating the threshold. This is done to categorize the potential flipping bits into black or gray, for very likely to switch in black and not so likely in gray. This is only important for the first iteration, where the procedure BFMaskedIter is called, flipping bits in e depending on the mask black or gray. Every iteration, the threshold is being calculated again and BFIter is being called, where only the first three lines are really important. The categorization into black or gray can be ignored, not being needed anymore after the first iteration.

NbIter is always instantiated with 5, as proposed by the BIKE team. τ is the threshold gap to determine the size of the gray set of positions, and set to 3 for an optimal decoder performance. The other parameters are according to the corresponding BIKE parameters.

The counter function $ctr(H; s; j)$, computes the number of unsatisfied parity-checks of j. It is the number of 1 (set bits) appearing in the same position of the vector s as well as in the j-th column of the matrix H.

The threshold selection function is defined by the desired security level 1,3 or 5, proposed by NIST in section 4.3.1. Note that there are other threshold selections possible, depending on the iteration i, but these were shown to not perform better for the BGF decoder. Therefore, the threshold can be precomputed and doesn't need to be evaluated in every iteration.

Algorithm 1 Black-Gray-Flip (BGF)

Require: $r, w, t, d = w/2, n = 2r;$ NbIter, τ, threshold (see text for details)
Input: $s \in \mathbb{F}_2^r, H \in \mathbb{F}_2^{r \times n}$
Output: $e \in \mathbb{F}_2^n$

$e \leftarrow 0^n$
for $i = 1, \ldots,$ NbIter **do**
    $T \leftarrow \text{threshold}(|s + eH^T|, i)$
    $e, \text{black}, \text{gray} \leftarrow \text{BFIter}(s + eH^T, e, T, H)$
    if $i = 1$ **then**
        $e \leftarrow \text{BFMaskedIter}(s + eH^T, e, \text{black}, (d+1)/2 + 1, H)$
        $e \leftarrow \text{BFMaskedIter}(s + eH^T, e, \text{gray}, (d+1)/2 + 1, H)$
if $s = eH^T$ **then**
    return e
else
    return $\perp$

procedure BFIter (s, e, T, H)
for $j = 0, \ldots, n-1$ **do**
    if $\text{ctr}(H, s, j) \geq T$ **then**
        $e_j \leftarrow e_j \oplus 1$
        $\text{black}_j \leftarrow 1$
    else if $\text{ctr}(H, s, j) \geq T - \tau$ **then**
        $\text{gray}_j \leftarrow 1$
return $e, \text{black}, \text{gray}$

procedure BFMaskedIter $(s, e, \text{mask}, T, H)$
for $j = 0, \ldots, n-1$ **do**
    if $\text{ctr}(H, s, j) \geq T$ **then**
        $e_j \leftarrow e_j \oplus \text{mask}_j$
return e

The DFR is an estimation for the BGF decoder based on the given specification and here represented by the symbol $\perp$. Furthermore, the key and message space l is fixed to 256, as proposed by the BIKE team.

Security	threshold(S)	DFR
Level 1	$\max(\lfloor 0,0069722 \cdot S + 13,530 \rfloor, 36)$	2^{-128}
Level 3	$\max(\lfloor 0,005265 \cdot S + 15,2588 \rfloor, 52)$	2^{-192}
Level 5	$\max(\lfloor 0,00402312 \cdot S + 17,8785 \rfloor, 69)$	2^{-256}

Table 4: Threshold function and decoding failure rate for BGF Decoder

New Bit-Flipping Decoder

Later in round 4 this decoder was again changed to New Bit-Flipping Decoder, making the decoder simpler and providing better resistance to weak key attacks [69].

Algorithm 2 New BIKE Decoder

Require:
Input: $s \in \mathbb{F}_2^r$, $H \in \mathbb{F}_2^{r \times n}$
1: $\tilde{e} \leftarrow 0^n; \tilde{s} \leftarrow s$
2: **for** $i = 1, \ldots,$ NbIter **do**
3: $\quad T \leftarrow \text{threshold}(i, s, \tilde{s})$
4: $\quad$ **for** $j = 0, \ldots, n-1$ **do**
5: $\quad\quad \sigma_j \leftarrow \text{ctr}(H, \tilde{s}, j)$
6: $\quad$ **for** $j = 0, \ldots, n-1$ **do**
7: $\quad\quad$ **if** $\sigma_j \geq T$ **then**
8: $\quad\quad\quad \tilde{e}_j \leftarrow \tilde{e}_j \oplus 1$
9: $\quad\quad\quad \tilde{s} \leftarrow \tilde{s} - \text{col}(H, j)$
10: **return** $\tilde{e}$

function threshold$(i, s, \tilde{s})$
1: $T' \leftarrow f_t(|s|)$
2: $M \leftarrow (d+1)/2$
3: **if** $i = 1$ **then** $T \leftarrow T' + \delta$
4: **if** $i = 2$ **then** $T \leftarrow (2T' + M)/3 + \delta$
5: **if** $i = 3$ **then** $T \leftarrow (T' + 2M)/3 + \delta$
6: **if** $i \geq 4$ **then** $T \leftarrow M + \delta$
7: **return** $\max(f_t(|\tilde{s}|), T)$
$f_t(x) = 0,006258 \cdot x + 11,094; \delta = 3$
(Level 1)

The counter function stays the same as the one for BGF decoder. The black and grey categorization of error correction has been removed, making the algorithm simpler. Furthermore, the threshold value has been modified, which changes over the iterations.

5.7 Hash Functions and Distribution D

BIKE makes use of three Hash functions.

L

The function is defined as $\mathcal{R}^2 \rightarrow \mathcal{M}$. It takes the bit representation of two binary polynomials $e_0 = a_0 + a_1 \cdot X + \cdots + a_{r-1} \cdot X^{r-1}$ and corresponding for e_1. The coefficients of $a_0 + a_1 + \cdots + a_{r-1}$ are taken and concatenated with the coefficients from e_1, building the input for the standard SHA384. The output is defined as the $l = 256$ least significant bits of SHA384. The notation $\mathbf{L}(e_0, e_1)$ refers therefore to hashing an input of $\{0,1\}^{r+r}$ bits.

K

The function is defined as $\mathcal{M} \times \mathcal{R} \times \mathcal{M} \to \mathcal{K}$. The output is defined as the $l = 256$ least significant bits of SHA384. The notation $\mathbf{K}(m, c)$, where $c = (c_0, c_1)$, refers therefore to hashing an input of $\{0,1\}^{l+r+l}$ bits. Here the bit representation of m is concatenated first, then c_0 and at last c_1.

H

Hash function **H** is defined as $\mathcal{M} \to \mathcal{E}_t$ through calling algorithm "WSHAKE256-PRF" with the appropriate parameters:

Algorithm 3 WSHAKE256-PRF

Input: seed (32 bytes), len, wt
Output: A list (wlist) of wt distinct elements in $\{0, \cdots, len-1\}$

```
wlist← ()
s_0, ··· , s_{wt−1} ← SHAKE256-Stream(seed,32 · wt)
for i = (wt − 1), ··· , 1, 0 do
    pos ← i + ⌊(len − i)s_i/2^3 2⌋
    if pos ∈ wlist then
        wlist ← wlist, i
    else
        wlist ← wlist, pos
return wlist
```

The algorithm therefore outputs a list of length wt, with elements between 0 and len-1. This list can be interpreted in the setting of **H** to a binary polynomial, where each of the coefficients is 0 except the ones in the list. Therefore, **H** will call algorithm "WSHAKE256-PRF" with the parameters wt=t, len=2r and the seed which is the input of **H**. Algorithm "WSHAKE256-PRF" generates the list of distinct elements, through calling SHAKE256 with the seed and 32·wt. For clarification let the output list of algorithm "WSHAKE256-PRF" be $(3, 0, 7)$, therefore the hamming weight of the corresponding polynomial is 3. This list is then converted to the polynomial $a = 1 + 1 \cdot X^3 + 1 \cdot X^7$. In the case of hash function **H** the list first gets split up into two equal parts each of length r and then gets converted into polynomials.

Distribution D

The **KeyGen** algorithm makes use of distribution D to pick $h_0, h_1 \in \mathcal{H}_w$. This distribution can reuse **Algorithm 3**, by calling it with the input len=r and wt=w/2. Since this produces only one polynomial, two calls are required, one for h_0 and one for h_1. "The corresponding distribution is biased, however the impact of this bias on security is hardly measurable [38]", and resolves in less code duplication.

5.8 Security Claims

The BIKE team presented an IND-CCA proof of the KEM of BIKE under four assumptions. To reach λ bits of (classical) IND-CCA security, it must hold that:

1. $QCSD_{r,t}$ offers λ bits of security
2. $QCCF_{r,w}$ offers λ bits of security
3. $|M| = 2^l \geq 2^\lambda$
4. DFR(decoder)$\leq 2^{-\lambda}$

1. and 2. refers to the hardness of the problems Quasi-Cyclic Syndrome Decoding 13 and Quasi-Cyclic Codeword Finding 14. For the general case of Syndrome Decoding 3. is fulfilled by the proposed parameter settings, since $l = 256$ should be used every time and 4. is elaborated more in the next section. The best known solver for problems 1. and 2. are mostly influenced by the parameters t and w, therefore these should be picked first. On this selection r should then be picked, under consideration of a low DFR.

Definition 13 ((2,1)-QC Syndrome Decoding - QCSD$_{r,t}$)**.** *Given $(h, s) \in \mathcal{R}_{odd} \times \mathcal{R}_{p(t)}$ and an integer $t > 0$, there exists $(e_0, e_1) \in \mathcal{E}_t$, such that $e_0 + e_1 h = s$.*

Definition 14 ((2,1)-QC Codeword Finding - QCCF$_{r,w}$)**.** *Given $h \in \mathcal{R}_{odd}$ and an even integer $w > 0$, with /2 odd, then there exists $(h_0, h_1) \in \mathcal{H}_w$, such that $h_1 + h_0 h = 0$.*

The parity of the weight matters, therefore it needs to be included in the problem statements. Here only the relevant problems for BIKE are noted.

5.8.1 Decoding Failure Rate

As mentioned before, the decoder (family) has a certain decoding failure rate (DFR). A low DFR, required for CCA security, can not be computed or estimated by simulation. This is due to the fact that the high block size r leads to an excessively large number of potential outcomes. A simulation would only represent a tiny fraction of the total possibilities, and therefore would not be a reliable representation. Instead, simulations of smaller block sizes r are taken, where the outcomes are more representative. For these smaller r the simulations are representative. The DFR is then calculated for various small values of r, and the results are plotted to create a curve showing the relationship between DFR and r. This curve can be extrapolated to estimate the DFR for larger block sizes.

Definition 15 (Decoding Failure Rate)**.** *Given a decoder, which takes as input a syndrome s and a parity-check matrix, the decoding failure rate is*

$$DFR(decoder) = PR[(e_0, e_1) \neq decoder(s, h_0, h_1)]$$

where $s = e_0 h_0 + e_1 h_1$.

To note is that there is no proof of an upper bound for the Decoding Failure Rate, therefore any IND-CCA proof of BIKE is under the assumption that BGF is IND-CCA secure. “Consequently, the BIKE instantiation with the BGF decoder does not make a formal claim for IND-CCA security, although by any practical considerations, this is probably the case [38].”

6 BIKE in EasyCrypt

EasyCrypt is a formal verification framework for cryptographic proofs that leverages automatic reasoning tools to assess the security of cryptographic algorithms and protocols. The increasing complexity of cryptographic algorithms and protocols makes it progressively more challenging to guarantee their security. Additionally, the sheer number of proofs has risen significantly and are "often too complex for humans to go through." "The problem is that as a community, we generate more proofs than we carefully verify (and as a consequence some of our published proofs are incorrect)", Shai Halevi wrote [25]. To address these issues, EasyCrypt and other theorem provers have been developed.

Implementing something in EasyCrypt can be difficult for beginners. In the following there will be a rough introduction to EasyCrypt by the example of the BIKE KEM *BIKE.ec* and the *Utils.ec* file. For more information the master thesis of Tejas Anil Shah [54] is highly recommend or the chapter "EasyCrypt: A Tutorial" from the book "Foundations of Security Analysis and Design VII" [8]. The first one is suited for beginner, never worked with a theorem prover before, with much more explanation and a broad introduction. The second one is pretty good to get a good and fast introduction into EasyCrypt.

For any detailed information about EasyCrypt, the official GitHub repository is the best source. Not only is the official documentation here to find, but also existing data type and many example implementations can be found there. When needing help with installation the tutorial of Alley Stoughton [65] is recommended.

All the implementation files can be found in the GitLab repository [1]. The main components for this bachelor thesis are:

1. The key encapsulation mechanism consisting of Key Generation, Encapsulation and Decapsulation
2. The Bit-Flipping Algorithms, which are used for decoding of the syndrome
3. The implementation of Polynomial Inversion Algorithm from [20]
4. The different security levels for parameter configuration
5. The Utils File, which consists of the main data types, libraries, further parameters, distributions and custom operations

The following section will provide a brief overview of the implementation of the BIKE KEM in EasyCrypt, with an explanation of what can be changed in terms of decoder and parameters. The *Utils.ec* file will be presented, giving more insights into EasyCrypt, as this file provides the utilities necessary for the program to function. Then the implementation of the two decoder will be discussed, followed by the implementation of the polynomial inversion algorithm. Finally, a brief overview of the security levels and the parameter configuration will be presented. For the following implementation, bits are represented as a vector in the base of $\mathbb{F}_2$. In mathematical sense it doesn't make any difference, but it was easier this way to work with it.

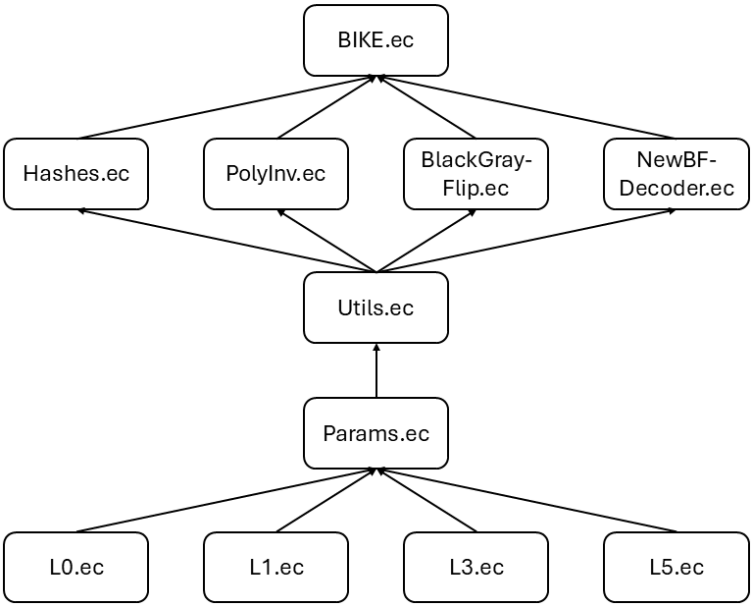

Figure 1: Inheritance tree of the implementation

6.1 KEM

The BIKE KEM algorithm consists of three primary algorithms: `keyGen`, `encaps` and `decaps`. In EasyCrypt file typically starts with importing the needed modules and libraries, done by `import`. The prefix `require` updates the namespace, enabling the imported modules to be used without their prefix. Comments are opened with `(*` and closed with `*)`, same for single and multi line. A class is defined by `module` and a procedure inside a `module` by `proc`. Typically a statement ends with a point or a semicolon, depending on the context.

```
(* CHANGEME: Replace with "NewBFDecoder" if needed *)
require import BlackGrayFlip.

require import AllCore Utils Hashes PolyInv.
import Matrix PolyRing Params.

module Bike = {
    proc keyGen(): (Utils.R * Utils.R) * M * Utils.R = { ··· }

    proc encaps(pk : Utils.R): K * (Utils.R * M) = { ··· }

    proc decaps( sk : (Utils.R * Utils.R), sigma : M, c : Utils.R *
       M ): K = { ··· }
}.
```

BIKE.ec

Since two decoder implementations are available, the user can choose which one to use by changing the `import` statement and the function call inside `proc keyGen`. This `proc` is the Key Generation algorithm, as defined in section 5.3. The brackets before the colon defines the input, the brackets after the colon the output. Therefore, it has no input and produces the secret key $(h_0, h_1) \in R, \sigma \in M$, and the public key $h \in R$. It says that the output is `R` and `M`, which are data `types`, further defined in the *Utils.ec* file, see section 6.2. Here `Utils.R` needs to be written to clarify which `R` is to be used, since `R` has more than one definition in the current namespace.

```
proc keyGen(): (Utils.R * Utils.R) * M * Utils.R = {

    var tmp;
    var h0, h1 : polyXnD1;
    var sk : (polyXnD1 * polyXnD1);
    var pk;
    var sigma : M;

    (*(h0, h1) D <- Hw*)
    tmp <@ Algorithm3.main(sigma, r, floor (w%r/2%r));
    h0 <- intlistToPoly tmp;
    tmp <@ Algorithm3.main(sigma, r, floor (w%r/2%r));
    h1 <- intlistToPoly tmp;
    sk <- (h0, h1);
    (*h <- h1h0^-1*)
    h0 <@ Algorithm2.invert(sk.`1);
    pk <- (h1 * h0);
    (*σ $<- M*)
    sigma <$  duniformM;

    return (sk, sigma, pk);
  }
```
BIKE.ec

Inside a `proc` the variables are defined by `var` and assignments are done by `<-`. One can define the type of `var` by `:` followed by the `type`. Besides from `type` there are also operations, denoted as `op`, which can be viewed as mathematical operations. Such an operation is `intlistToPoly`, which is defined in *Utils.ec*, converting a list of integers to a polynomial. Even basic operations like addition are considered operations in EasyCrypt. To view the definition of an operation, one can execute the `print` command.

```
    print ( + ).
```

The output of each + is being shown, that specifies the addition for different types.

```
op (+) (x y : polyXnD1) : Subtype.sT =
  pinject (prepr x + prepr y)%BasePoly.PolyComRing.
(* Matrices.(+) *)
op (+) (m1 m2 : matrix) : QuotientMat.Subtype.sT =
  offunm
    (fun (i j : int) => (m1.[i, j] + m2.[i, j])%ZR, max (rows m1) (
       rows m2),
     max (cols m1) (cols m2)).
(* Vectors.(+) *)
op (+) (v1 v2 : vector) : QuotientVec.Subtype.sT =
  offunv (fun (i : int) => (v1.[i] + v2.[i])%ZR, max (size v1) (size
      v2)).
(* Xint.(+) *)
abbrev (+)  : xint -> xint -> xint = xadd.
(* Real.(+) *)
abbrev (+)  : real -> real -> real = CoreReal.add.
(* Int.(+) *)
abbrev (+)  : int -> int -> int = CoreInt.add.
```

Typically, an op is defined by obtaining the input parameters after the op name. However, this can be changed to inline, as seen by addition with brackets.

Also, proc calls can be done by <@, here by calling `Algorithm3.main` defined in *Hashes.ec*. Another proc call is `Algorithm2.invert`, the polynomial inversion algorithm defined in *PolyInv.ec*. The last thing to call is <$, for calling out distributions over a given type. Here it is the uniform distribution over M, defined in *Utils.ec*.

Integers are a basic type in EasyCrypt, yet there are many more such as Reals, Booleans, Vectors and Polynomials. Many of them are defined in the in EasyCrypt standard library, imported by import `AllCore`. These modules consist of numerous lemmas and axioms defining the types. Conversions between these data types can be challenging, but the *Utils.ec* file provides many custom helper functions, specified for this. For the example of Integer to Real conversion, one can simply say for a variable `x` of type Integer, `x\%r` to convert it to a Real. The other way around is also possible but not so easy. Since not every real number is also an integer a function is needed mapping real to int. This can be done by rounding the real number, therefore calling the op `floor`, resulting not in a one-to-one mapping. But since it is known that w is even, $w/2$ is also an integer, so no information loss will take place here. Sometimes one want to make a tuple of two variables, just like $sk = h_0, h_1$. To access a certain element of the tuple, one can use the dot operator, like `sk.‘1` for the first element, here h_0.

6.2 Utilities

This file contains all important data types, libraries, further parameters, distributions and custom operations. In this implementation, primarily the vectors and polynomials are used,

besides the core data types like Integers and Reals. The majority of data types have already many utility functions and operation, like addition, subtraction or concatenation for vectors. Nevertheless, this is not the case for all data types, and frequently not for all the operations that are required. Polynomials have two already existing implemented variants, the basic polynomial defined by *Poly.eca* and an extension to consider rings with polynomials by *PolyReduce.eca*. For vector representation the *DynMatrix.eca* is used, representing a dynamic matrix and vector implementation with a greater number of utility functions and operations than those available in the *Matrix.eca* theory. One should be careful when using the dynamic variant, since it does not check for compliance of dimensions of the matrices and vectors, which can lead to errors, as example for matrix multiplication. The suffix *a* in the file name stands for abstract, meaning that when importing this file, certain things need to be defined by the user.

```
require import AllCore List Distr IntDiv Params IntMin.

require ROM PolyReduce BitWord DynMatrix RealExp.

(* Polynom Ring *)
clone import PolyReduce.PolyReduceZp as PolyRing
with op n = r, op p = 2
proof *.
realize ge2_p. smt. qed.
realize gt0_n. smt. qed.

(* Matrix *)
clone import DynMatrix as Matrix with
op ZR.(+) = PolyRing.Zp.(+),
type ZR.t <= PolyRing.Zp.
```
Utils.ec

Here the data types are imported, but since there is more than one theory defined in *PolyReduce.eca* and *DynMatrix.eca*, the user needs to define which one to `import`. Then `clone` is used to clone the abstract theory, and `as` keyword to rename it. For the polynomials, the exponent $n = r$ of the polynomial $[X]/(X^r - 1)$ needs to be defined and shown that it is greater than 0. Also, it is a polynomial ring with base $\mathbb{F}_2$, specifically $\mathbb{F}_2[X]/(X^r - 1)$, where $p = 2$.

```
clone import PolyReduce.PolyReduceZp as PolyRing
with op n = r, op p = 2
```
Utils.ec

Since the field is initiated with 2, the `smt` solver of EasyCrypt can prove the property $p = 2 \Rightarrow p \geq 2$ by itself, as well as the property $n > 0$ with $n = r$. The `smt` solver is a solver for satisfiability modulo theories, here in the case of EasyCrypt, Z3 is used, a "state-of-the art theorem prover from Microsoft Research" [18].

Next the three primarily used data types are defined, the polynomial `R` and the vectors `M` and `K`, just for easier access and more readable code.

```
(* Cyclic polynomial ring *)
type R = polyXnD1.
(* message space {0, 1}^l *)
type M = vector.
(* shared key space {0, 1}^l *)
type K = vector.
```
Utils.ec

The following section defines all custom operations `op` needed for the implementation. Many are operations for converting data `types`, like `polyToVec` or `vecToPoly`, which transforms a polynomial to a vector and vice versa. The `op` `toDecoder`, converting three polynomials from the Decapsulation algorithm to a vector and a block circulant matrix. As in the case of `proc` the input of an `op` is defined by the brackets before the colon and the output by what is after the colon. This can also be left empty and EasyCrypt derives them itself. The actual function is then defined after the `=` sign.

```
(* From Polynom to Vector *)
op polyToVec (x : polyXnD1) : vector = offunv ((BasePoly.of_poly (
   prepr x)), poly_length).

(* From Vector to Polynom *)
op vecToPoly (x : vector) : polyXnD1 = polyLX (tolist x).

(* From Vector to Circulant Matrix *)
op vecToCircMatr (x : vector) : matrix = offunm ((fun j i : int => x
   .[(i-j) %% (size x)]), size x, size x).

...

(* Converts from BIKE to Decoder *)
op toDecoder (s h0 h1 : R) : vector * matrix = (polyToVec s, (
   vecToCircMatr (polyToVec h0)) || (vecToCircMatr (polyToVec h1))).

...
```
Utils.ec

The operator `vecToCircMatr` converts a vector to a circulant matrix. The size of the input vector x is used to define the size of the matrix `size x, size x`. And the matrix is filled with the elements of the vector x, where the (i,j)-element of the resulting matrix is specified by the $(i-j)$ *mod* (size x) element of x, with `%%` is the modular operator. The `||` operator is used to concatenate two matrices, defined by the *DynMatrix.eca* theory. But also some more complex operators like `i2b`, converting an integer into a bit, here illustrated through a vector $\mathbb{F}_2$, are defined in *Utils.ec*.

```
(* Little helper to create an empty list of type int *)
op empty_int_list : int list = [].

op bitSizeComp (x : int) = (x=0) ? 1 : (argmax (fun i => 2^i) (fun j
    => j<=x)) + 1.

op i2b_iter (x : int * int list) = (x.`1 = 0) ? (x.`1, x.`2) : (x.`1
    %/ 2, (x.`1 %% 2) :: x.`2).

op i2b (x : int) = oflist (IntListToR (iter (bitSizeComp x) i2b_iter
    (x,empty_int_list)).`2 ).                                    Utils.ec
```

The op `bitSizeComp` computes the length of the bit representation of an integer x, by finding the maximum element y, where $2^y \leq x$. The op `i2b_iter` is the core iteration, dividing the integer x by 2 (`%/` is division with remainder) and adding $x \mod 2$ in front of the list. Lastly the op `i2b` wraps the list and calls the iteration `i2b_iter`. Also, the `if` condition was changed to a shorthand variant `?`, which is a ternary operator, returning the first argument if the condition is true and the second, which is after the colon, if it is false.

Additionally, the counter function `ctr` is defined here, which is used by the decoder. The counter function computes the number of unsatisfied parity-checks of j. It counts up when a 1 appears in the same position in the vector s as well as the j-th column of the matrix H. The op `ctr` is not complex, and unwraps the input to two integer lists. The actual function for countering `listcount` is then called with the two lists `x,y` as input, counting the output up for every element which is equal to 1 in both lists `(w=1 && v=1)? 1 : 0`.

```
op listcount(x y : int list) : int =
    with x = [], y = [] => 0
    with x = [], y = w :: ws => 0
    with x = w :: ws, y = [] => 0
    with x = w :: ws, y = v :: vs => ((w=1 && v=1) ? 1 : 0) +
       listcount ws vs.

op ctr (H : matrix, s : vector, j : int) : int = listcount (
   RListToInt (tolist (col H j))) (RListToInt (tolist s)).
...                                                              Utils.ec
```

The last thing defined in *Utils.ec* are the distributions. The hash functions **K** and **L** give out $\mathcal{K}$ and $\mathcal{M}$ both in $\{0,1\}^l$, therefore needing a uniform distribution over $\{0,1\}^l$.

```
(* Distribution for Vector of length l and Base F_2 *)
op duniformM = Matrix.Vectors.dvector Zp.DZmodP.dunifin l.       Utils.ec
```

6.3 Hashes

To implement the hash functions **K**, **L** and the **Algorithm 3** the concept of a Random Oracle Model (ROM) was used. Such a ROM is used instead of the implementation of the actual hash functions, making proofs easier and independent of possible interference between a particular hash function and the rest of the algorithm. The corresponding theory can be found under *ROM.eca* in the EasyCrypt standard library. The definition for the hash function **K** is shown here, the one for **L** is similar but with other input and output parameter.

```
(* Specializing the hash function K *)
clone ROM as ROK with
  type in_t    <- M * Utils.R * M,
  type out_t   <- K,
  op   dout _  <- dvector Zp.DZmodP.dunifin 1.

import ROK.
import Lazy.

module K = {
  proc init(): unit = { LRO.init(); }
  proc hash(x:M * Utils.R * M): K = { var y; y <@ LRO.o(x); return y
     ; }
}.
```
Hashes.ec

`in_t` is the input parameter, `out_t` the output parameter and `dout` the distribution of the output. The "hashing" can then be done by calling the `proc hash` with the input parameter $\mathcal{M} \times \mathcal{R} \times \mathcal{M}$ and the output is then uniformly distributed over $\mathcal{K}$. Also, the **Algorithm 3** WSHAKE256-PRF 5.7 is defined here, which is used for the definition of the distribution `D` and for hash function **H**. The call of the hash function SHAKE256 is done by `SingleStream .hash`, where `SingleStream` is the random oracle outputting 32 bit vectors of uniform distribution. The implementation of the hash function **H**, calls `Algorithm3` with the parameters `len=2r` and `wt=t`.

```
module Algorithm3 = {
    proc main(seed : vector, len : int, wt : int): int list = {

      var j <- 0;
      var tmp;
      var i <- wt-1;
      var pos : int;

      (* wlist <- () *)
      var wlist : int list;
      (* The least significant bit is given first *)
      var s : vector list;
      (* s_0,...,s_wt-1 <- SHAKE256-Stream(seed,32*wt) *)
      while (j<wt) {
        tmp <@ SingleStream.hash(seed,j);
```

```
        s <- rcons s tmp;
        j <- j + 1;
      }

      (* for i = (wt - 1), . . . , 1, 0 do *)
      while (0<(i+1)) {
        (* pos <- i + | (len-i) * s_i / 2^32 | *)
        pos <- i + floor((len - i)%r * (b2i (nth (zerov 0) s i))%r /
            (2^32)%r);
        (* wlist <- wlist, (pos ∈ wlist) ? i : pos *)
        if (mem wlist pos) {
          wlist <- rcons wlist i;
        } else {
          wlist <- rcons wlist pos;
        }
        i <- i - 1;
      }
      (* return wlist *)
      return wlist;
    }
}.
```

Hashes.ec

Therefore the Hash function `H` looks as follows:

```
(* Specializing the hash function H *)
module H = {
  proc hash(x) = {
    var tmp;
    var h0;
    var h1;
    tmp <@ Algorithm3.main(x, 2*r, t);
    h0 <- intlistToPoly (take r tmp);
    h1 <- intlistToPoly (drop r tmp);
    return (h0,h1);
  }
}.
```

Hashes.ec

Additionally, an alternative definition is given, called `H2`. It can be found at the bottom of the file *Hashes.ec*, and is done with the random oracle model. The corresponding distribution for this ROM is `distribution_over_E_t` defined in *Utils.ec*.

6.4 Decoder

The decoder for BIKE is not set and can be changed to any other algorithm, the properties are defined in section 5.6. Both decoders are implemented, to change them, only the import statement and the function call inside the `proc` in the *BIKE.ec* file need to be changed.

Black-Gray-Flip

The main function of the BGF, which has no name in the BIKE paper, is now called `proc main` and is at the end of the `module`, due to compiler restrictions. As said before, the *DynMatrix.eca* has a lot of utility operations defined. Calculations like $s + eH^T$ are defined as:

```
seH <- row ((rowmx s) + (rowmx e) * (trmx H)) 0;
```
BlackGrayFlip.ec

`trmx` computes the transpose of a matrix and `rowmx` the row matrix of a vector. The vectors e and s are converted to row matrices, to make the multiplication with the transposed matrix H^T logical possible. Since this results in a row matrix, it needs to be converted back to a column vector by `row`, where the 0 at the end is defining which row. This statement is done beforehand to make the code more readable. Since not specified in the paper, it is assumed that the vectors `black` and `gray` are instantiated with 0^n. Given that the elements of the vectors are in $\mathbb{F}_2$, the XOR operation is done by applying the addition operator `+` defined in the finite field $\mathbb{F}_2$. The symbol for a decoding failure now defined as `emptyv`. Therefore, the return statement is defined as:

```
return (QuotientMat.(==) (tofunm (rowmx s)) (tofunm ((rowmx e) * (
    trmx H)))) ? e : emptyv;
```
BlackGrayFlip.ec

Because there are multiple equivalence checks in the namespace, the concrete `==` needs to be specified by `QuotientMat.==` followed by the arguments. The `tofunm` function converts a matrix to a function from (int,int) to the corresponding element in the matrix, which is needed for the equivalence check. Also, the `if` condition was again changed to a shorthand variant `?`. Since `var` are needed to be defined before anything else each `proc` starts with every needed variable. Loops are limited to `while`, changing the loop conditions a bit. It is important to increase the loop condition in every iteration by `i <- i + 1;`, to avoid an infinite loop.

```
(* The main Black-Gray-Flip algorithm *)
proc main(s : vector, H : matrix) : vector = {
    var e, black, gray, seH : vector;
    var i <- 1;
    var t : int;

    (* e <- 0^n *)
    e <- zerov Params.n;
    (* precomputing s + e * H^T *)
    seH <- row ((rowmx s) + (rowmx e) * (trmx H)) 0;
    (* for i = 1,..., NbIter do *)
    while(i < NbIter+1) {
        ...
        i <- i + 1;
    }
}
```
BlackGrayFlip.ec

NewBFDecoder

Since the threshold function has become more complex it was moved to the *NewBFDecoder.ec* file. The $f_t(x)$ function is defined in the security level file of *L0.ec* and *L1.ec*, as well as δ which is here called `delt`. This is due the missing specification of these Levels in the BIKE presentation slides for the new decoder [64]. So this decoder is working **only for the security level 0 and 1**. No implementation details will be shown here, since no new concepts are used for EasyCrypt and no changes have been made to the published slides. Furthermore, it is assumed that the parameters will be the same for *NewBFDecoder.ec* as for *BlackGrayFlip.ec*.

6.5 Polynomial Inversion

```
require import AllCore Utils RealExp.

import Params PolyRing IntDiv.


module Algorithm2 = {
    proc invert(a : R) : R = {

        var i <- 1;
        var g : R;
        var f <- a;
        var result <- a;

        while (i < floor(ln (r-2)%r)+1) {
            g <- f ^ (2 ^ (2 ^ (i-1)));
            f <- f * g;
            if (Zp.asint (Matrix.Vectors.get (i2b (r-2)) i) = 1) {
                result <- result * f ^ (2 ^ ((r-2) %% 2^i));
            }
            result <- result * result;
            i <- i + 1;
        }
        return result;
    }
}.
```

PolyInv.ec

The implementation of the polynomial inversion algorithm is done in `Algorithm2` (see above) and according to the **Algorithm 2** in the specification [20]. The algorithm works by raising a polynomial x to the power of 2^k for some k, which can be done efficiently, therefore resulting in a lower runtime. This is especially important since polynomial inversion usually is a time consuming operation [20].

The algorithm starts by initializing variables i, g, f, and *result*. Here, f is set to the input polynomial a, and *result* will eventually hold the inverse of a. The main part of the algorithm

is a loop that continues as long as i is less than $\lfloor \log_2(r-2) \rfloor + 1$. In each iteration, g is computed by raising f to the power of $2^{2^{(i-1)}}$. The variable f is updated by multiplying it by g, effectively squaring f. `if` the i-th bit of $r-2$ (in its binary representation) is 1, the algorithm performs an additional multiplication by raising f to a power based on $r-2$ mod 2^i. This ensures that the inversion is computed correctly using the binary decomposition of $r-2$.

6.6 Security Levels and Parameter Configurations

The BIKE team presented for the BGF parameter selections for the three security levels. The definition for these are in the corresponding *L.ec* files. For example, the *L3.ec* file is used for security level 3. Additionally, the file *L0.ec* is added. This file is using tiny parameters, which are not secure, but are used for testing purposes, making runtime and debugging easier and much faster. Note that only *L0.ec* and *L1.ec* have parameter configuration for *NewBFDecoder.ec*.

```
(* This is the parameter setting for the desired Security Level 3 *)
require import AllCore.

(* System Parameter *)
op r = 24659.

op w = 206.

op t = 199.

op l = 256.

(* Decoder Settings *)
op NbIter = 5.

(* Special Settings for BGF *)
op tau = 3.

op threshold (s : int , i : int) : int = max (floor(0.005265*s%r
   +15.2588)) 52.
```
L3.ec

To change the security level used by the scheme, one simply needs to change the `export` statement in the *Params.ec* file. On default, it is the testing setting L0.

```
(* For another Security Level change export *)
require export L0.
```
Params.ec

Some `lemmas` are defined in order to ensure the parameter selection is valid. That $w \approx \sqrt{n}$ is omitted, since the statement is too imprecisely.

```
op d = w %/ 2.

lemma w_even: 2 %| w.
smt. qed.

lemma uneven_w_half: !(2 %| d).
smt. qed.
```
Params.ec

In `lemma w_even` it was proven that w is indeed even, and `uneven_w_half` shows that d is not even, since $d = w/2$, which is a necessary condition for w. The following proof was contributed by Prof. Dr. Dominique Unruh, who helped me a lot throughout the implementation of BIKE in EasyCrypt.

```
lemma r_prime: prime r.
  pose P := (fun q, -r <= q => q %| r => `|q| = 1 \/ `|q| = r).
  have step : forall t, -r <= t => P t => (forall q, q <= t-1 => P q
     ) => forall q, q <= t => P q.
    smt.
  have Hmain: forall q, q <= r => P q.
    rewrite /r.
    do (apply step; [ by trivial | by trivial | simplify ]).
    smt.
  rewrite /prime.
  progress.
  have Hbound: q <= r. smt.
  have Hnegbound: -r <= q.
    have H2: -r %% -q = 0.
    rewrite modzN; smt.
    smt.
  apply Hmain; trivial.
qed.
```
Params.ec

The induction based `lemma r_prime` proves, that for every integer t tinier than r, it holds that t does not divide r or is equal to 1. By showing this for every number q smaller than r, it is shown that r has no divider expect from 1 and is therefore a prime number. `P` is the predicate, defining, that when a q is in the range $-r$ to r, and q divides r, then q is either 1 or r. The next step is then showing that `P` holds for every t smaller than r, with induction initialized by `have step`. The main induction is then `Hmain`, later applied by `apply Hmain; trivial`.

6.7 Challenges in the Implementation

EasyCrypt is a powerful tool for formal verification, but it has its challenges. Restrictions by the compiler, the abstractness of the language, and the lack of documentation were some of the main challenges faced during the implementation. But also the absence of real iterations, apart from while loops inside a `proc`, and the limited recursion, which seems to be possible

only with user-defined types like lists, is a big challenge. It is not possible to just execute the program and see if the output is correct. No real debugging is possible, and the only way to check the correctness is by proving it, which is even more difficult than the implementation itself. The base for proving has been set with this thesis. The proving itself needs to be done in the next step. Most of the time was spent on data type conversion. The availablility of complex data types and structures is needed for the implementation of the BIKE algorithm. Some base implementation of these data types are available by the EasyCrypt community, but needed to be extended for the required type conversions.

7 Conclusion

After introducing the core concepts of cryptography, RSA was presented as a widely used public key cryptographic system. However, as shown by Shor's algorithm, RSA is not secure against quantum computers. Therefore in the section 4.2 a short introduction to quantum computing was given, showing the concepts of quantum computing. Also additional reading material was provided in A[43], to further dig deeper into the topic of quantum computing. Then Shor's algorithm was explained in a short manner, with a full in depth explanation in the Appendix. Shor's algorithm results in an efficient algorithm for factoring large numbers, which is the basis for the security of RSA, therefore making RSA insecure against quantum computers. That's why quantum computers are a threat to algorithms like RSA, and the need for new cryptographic algorithms is rising. After this motivation, the NIST post-quantum cryptography competition was introduced, with the goal of finding these new cryptographic algorithms that are secure against quantum computers. The selection principles and the remaining candidates and finalists were presented, with BIKE as one of the finalists. Besides BIKE, which uses codes for the underlying mathematical structure, there are also lattice-based, elliptic curve based and other algorithms. The BIKE algorithm was explained in detail, including the underlying mathematics, as well as a short course into the analyses of the decoder performance.

The goal of the thesis was the implementation of the BIKE key encapsulation mechanism in the theorem prover EasyCrypt. To achieve this, the BIKE scheme was implemented in EasyCrypt, including the Key Generation, Encapsulation and Decapsulation algorithms. Also all security properties of the scheme have been implemented, using the provided parameter configurations for security levels 1, 3, and 5, as well as some additional parameters for testing purposes. For these parameters the neccessary constraints were proven. The polynomial inversion algorithm was implemented, which is used in the KeyGen algorithm. Two decoder were implemented, the Black-Gray-Flip and the NewBFDecoder, which is the new decoder for BIKE. The Hash functions were realised with the Random Oracle Model, and **Algorithm 3** WSHAKE256-PRF was implemented for the hash function **H** and the distribution `D`. In addition, operations for collaboration with the algorithm have been implemented and made available in the *Utils.ec* file.

Explained by the implementation of the BIKE scheme, the theorem prover EasyCrypt was introduced, with its basic syntax and structure. As shown, with the build in smt-solver Z3, EasyCrypt can prove many properties by itself.

7.1 Discussion

BIKE is a promising candidate for the NIST post-quantum cryptography competition, with its security based on the hardness of decoding random linear codes. Although it is not proven yet that the problems $QCCF_{r,w}$ and $QCSD_{r,t}$ 5.8 are difficult to be solved, the underlying basic mathematical structure is well understood and has been used in other cryptographic systems. The analysis of the selected decoder is crucial for the security of the scheme, and the new decoder is a promising approach to improve the performance of the algorithm. The freedom of choice for the decoder is a big advantage, with the possibility to change the

security by changing the decoder, without changing the whole scheme. Also the decoder can be changed in order to adapt BIKE to different use cases, where the original use case is for synchronous communication protocols such as TLS. But the freedom of choice can also be seen as a disadvantage, since the security of the scheme is dependent on the decoder, and a wrong choice can lead to a security breach.

A big influence on BIKE performance has the polynomial inversion algorithm, which is used in the KeyGen algorithm. The provided algorithm is efficient and fast, but only suitable for the use case of a polynomial ring over $\mathbb{F}_2$, which is the case for BIKE. Through the correct selection of the parameter r, the algorithm is provided with the correct ring size, such that there is always a solution for the inversion. Therefore the provided algorithm is a good choice for the BIKE scheme.

EasyCrypt is a powerful tool for formal verification, and with the right knowledge and experience, it is a good choice. The implementation in EasyCrypt is very challenging because of the lack of documentation. The provided documentation is outdated and does not cover the current state of the language. Other frameworks like Coq, which has a bigger community and more documentation, could have probably been an alternative choice.

Lastly the implementation of the BIKE scheme in EasyCrypt was a success. Although implementations in C and a hardware implementation are already realised, they do not provide the framework for formal verification of the security scheme. This base is now set with the implementation within this thesis.

It is currently not recommend to implement BIKE into an existing real world system, due to the not finished discussion over the decoder and the therefore missing parameter selection for the new decoder. A final report to the end of round 4 of the NIST PQC competition is outstanding, and the final decision for the new decoder is not made. If BIKE is going to be standardized, the "best" decoder will be provided there, with the corresponding parameter selection and a final security analysis.

7.2 Future Work

From the provided fundamental implementation, extended proofs over certain security properties of BIKE can be made. One can compare the two decoders, the Black-Gray-Flip and the NewBFDecoder, to see which of them has a lower decoding failure rate. Even the IND-CCA security of BIKE can be proven, which is done by the BIKE team only with 3 constraints. Still missing are the definitions for security level 3 and 5 of the new decoder, which are not provided by the BIKE team till now. No matter if done in EasyCrypt or not, BIKE still needs more analysis and proving to be sure of its security. Especially the decoder and it's performance plays a crucial role, as well as the selection of the parameters. This however does also apply to the other candidates and even finalists of the NIST PQC competition. It happens that a cryptographic scheme is broken after years of usage, therefore testing and analyzing is crucial for the security of the scheme, on the base that is set within this thesis.

A Quantum Computing

As mentioned, a classical computer uses classical bits, which can be either 0 or 1, whereas a quantum computer uses quantum bits (qubits), consisting of quantum particles. This allows a qubit to be in any state between 0 and 1, resulting in an infinite number of potential states. It is common to write a quantum state in ket-Notation, also called Dirac-Notation.

$$\alpha \cdot |0\rangle + \beta \cdot |1\rangle$$

where α and β are the amplitudes. In order to be a valid quantum state it needs to hold that the amplitudes

$$\alpha, \beta \in \mathbb{C} : |\alpha|^2 + |\beta|^2 = 1$$

A simple example for a quantum state is $1 \cdot |0\rangle + 0 \cdot |1\rangle$, where it is fully in the classical possibility 0, representing the classical bit with the value 0. Another example would be $\frac{1}{\sqrt{2}} \cdot |0\rangle + \frac{1}{\sqrt{2}} \cdot |1\rangle$. This time the quantum state is not decided between 0 and 1, but it is 0 and 1 at the same time, a so-called superposition. The state of the qubit is then with a probability of $|\frac{1}{\sqrt{2}}|^2 = \frac{1}{2}$ in 0 and with $|\frac{1}{\sqrt{2}}|^2 = \frac{1}{2}$ in 1. The probability of a quantum state to be in a certain state is therefore depending on the amplitudes. For the general case $\alpha \cdot |0\rangle + \beta \cdot |1\rangle$ it would mean that the probability of being in 0 is $|\alpha|^2$ and $|\beta|^2$ for being in 1.
The superposition could be also represented as:

$$\frac{1}{\sqrt{2}} \cdot |0\rangle + \frac{1}{\sqrt{2}} \cdot |1\rangle = \frac{1}{\sqrt{2}} \cdot (|0\rangle + |1\rangle)$$

It can also be written in vector basis:

$$\frac{1}{\sqrt{2}} \cdot |0\rangle + \frac{1}{\sqrt{2}} \cdot |1\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

where the first vector entry represents the classical possibility 0 and the bottom entry represents 1. For this specific case of a quantum state, there is also the shorthand notation $|+\rangle$. In similar logic $|-\rangle$ represents the quantum state $\frac{1}{\sqrt{2}} \cdot |0\rangle - \frac{1}{\sqrt{2}} \cdot |1\rangle$ [28].

In order to deepen the principles explained so far and to analyze the behavior of quantum states under observation, we will examine the following publicly known experiment.

A.1 Schrödinger's cat:

In front of us is a box, which is completely lightproof. There are also no other ways to get information from inside the box, else then opening the lid. Inside the box is a cat, a radioactive sensor with a toxic gas ampule attached and a radioactive atom. At the start of the experiment the cat is perfectly healthy and alive. The radioactive atom when decayed, triggers the sensor, releasing the toxic gas, resulting in the death of the cat.

The catch: No one knows when this radioactive atom is going to decay. And so nobody knows if the cat is still alive or not. "There, metaphorically speaking it is possible for a cat to be alive and dead at the same time. It is indeterminate whether it is dead or alive [28]."

With a certain probability p the atom decays, resulting in the death of the cat, and with the probability $1-p$ it is stable, and the cat is alive.

The bridge to the quantum computer setting can now be closed by considering this atom as the quantum bit, with the two classical possibilities being "stable" or "decayed" and the corresponding amplitudes $\sqrt{p}$ and $\sqrt{1-p}$. So at this point, it is unknown if the cat is still alive or not, since it is unknown in which state the radioactive atom is. The only possible way to actually get information, is to open the lid. When doing so, it becomes clear whether the cat is dead or alive, which also reveals whether the radioactive atom has decayed or not. It is only at this point that the radioactive atom has "decided" in which state it is, and therefore also changed its state to one of two possible post-measurement states (p.m.s), written in ket notation:

With a probability of p is the p.m.s $1 \cdot |\text{decayed}\rangle + 0 \cdot |\text{stable}\rangle$ and with a probability of $1-p$ is the p.m.s $0 \cdot |\text{decayed}\rangle + 1 \cdot |\text{stable}\rangle$.

We take from this, that by observing a quantum state it is forced to choose one state, and therefore resolves the superposition. It is really important for this, that no interaction between the quantum particle and the outside world has occurred. Else some kind of measurement would take place, ruining the superposition of the particle [28].

The objective is to utilize the quantum computer to perform a specific calculation. It is therefore crucial to gain an understanding of how to manipulate the state of a quantum particle in a desired manner.

Calculating with Quantum States:

In a classical probabilistic system one could change the system by applying a stochastic matrix to it. The equivalence for a quantum system is applying a unitary matrix to a quantum state. It changes a quantum state with the amplitude condition into another quantum state, where the amplitude condition is also fulfilled.

Definition 16. *A matrix A is a unitary matrix, iff one of these two conditions hold:*

1. *For all quantum states x it holds that $Ax = y$, where y is also again a valid quantum state*

2. *$A^\dagger \cdot A = I$, where $A^\dagger$ is the complex conjugate matrix and I is the Identity matrix*

Lemma 1. *From the definition of unitary matrices 16 it follows that for any quantum state x and any unitary matrix B it holds that*

$$BB^\dagger = I$$

where I is the corresponding Identity Matrix, and therefore it also holds

$$xBB^\dagger = x$$

With the Identity matrix we already have the first unitary matrix, which maps a quantum state x to itself.

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

A very important unitary is the Hadamard gate. Gate, because just like in a classical circuit the unitaries are logical operators on quantum bits.

$$H = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$$

It changes a quantum state which is currently in a classical position into one in superposition.

$$H \cdot |0\rangle = H \cdot 1 \cdot |0\rangle + 0 \cdot |1\rangle = \frac{1}{\sqrt{2}} \cdot |0\rangle + \frac{1}{\sqrt{2}} \cdot |1\rangle = |+\rangle$$

[28][43]

The forthcoming experimental procedure may not present an immediately apparent logical rationale, a viewpoint I initially shared. However, it is crucial to note that mathematical principles remain invariant regardless of our intuitive understanding. Additionally, it is an experimental study, the results of which align with the initial hypothesis [21].

A.2 Elitzur-Vaidman bomb tester:

The goal of the experiment is to determine whether a bomb is present within a black box without opening it. Beginning with a lightproof box again, but this time it is possible to interact with it, by sending single photons through the box [21]. Photons are single light quantum particles, forming together the light as we know it. If from a lightbulb, laser or the sun, every light consists of many photons, which individually follow the rules of quantum physics [48].

Inside the box is a beam splitter, that splits up a beam of light, or in our case just one photon. For the original experiment there was a half-silvered plates used. One photon is sent through the box and, upon hitting the beam splitter, has an equal probability of being transmitted or reflected, moving in either direction. The unitary equivalence of this process is the Hadamard matrix, changing the quantum state from a classical position into a superposition. This is due to the fact that the photon is a quantum particle, and thus follows the principles of quantum physics. So the photon gets transmitted and will be sent through the “upper” path, and also it will be reflected and will be sent through the “lower” path. For those who have reservations about the functioning of such a photon, it is recommended to look at the double-slit experiment [71].

The photon will be reflected by a mirror and subsequently directed through another beam splitter, regardless of whether it is sent through the “lower” or “upper” path. The second beam splitter is again either reflecting or transmitting the photon. Following the second beam splitter, two photon detectors, labelled A and B, are employed to determine whether a photon has reached one of the designated points. A measures if the photon is in the upper

path, B measures the lower path. Now comes the bomb, which is potentially added to the lower path after the first beam splitter, but before the second beam splitter. It doesn't matter in which path it is, so it is assumed to be in the lower path. The bomb is equipped with a photon sensor, which is designed to detect the passage of photons and subsequently detonate. The setup is complete, it's time to send a photon through the box.

1. Setup with a bomb

A photon is characterized by two distinct classical states: one representing motion upwards, and the other representing motion to the right. These states are represented by the digits 0 and 1, respectively. At the start a photon will be sent through the quantum circuit, with the initial state $|0\rangle$. Upon passing the first beam splitter is can either get transmitted or reflected. When being transmitted, the photon will be going through the lower path, resulting in the bomb to explode and photon not continuing on its way. The probability for this is $\frac{1}{2}$. When going through the upper path on the other hand, nothing happens until the second beam splitter. Then it will be split again by the second beam splitter, with again a 50% chance each. The last step are the detectors, where with a probability of $\frac{1}{4}$ A will measure the photon, and with probability $\frac{1}{4}$ B will measure. So there are 3 possible outcomes (P.m.s) when there is a bomb:

-	Chance
No Photon detected	1/2
Photon detected at A	1/4
Photon detected at B	1/4

Table 5: Elitzur-Vaidman bomb tester outcomes with a bomb

where "No Photon detected" means it will not get measured since it hit the bomb, which then exploded.

2. Setup but without a bomb

Again a photon is sent through the quantum circuit with the initial state $|0\rangle$. This time there is no bomb on the lower wire. So the photon gets send through both beam splitter directly after each other, without any intervening events. And from 1 it follows since $H = H^\dagger$ that $H \cdot H^\dagger = I$, so the photon comes back out in state $|0\rangle$, which was also the input value of the quantum state. Then comes the measurement, where only detector A will measure the photon, since it corresponds to $|0\rangle$. The resulting P.m.s are:

-	Chance
No Photon detected	0
Photon detected at A	1
Photon detected at B	0

Table 6: Elitzur-Vaidman bomb tester outcomes without a bomb

A comparison of the two tables grand an information gain. The data indicate that detection of a photon by detector B necessarily implies the presence of a bomb in the box. This

conclusion follows from the observation that detector B would be unable to register any measurement in the absence of the bomb within the box, with a probability of $\frac{1}{4}$. Consequently, this experiment demonstrates the capacity to measure the presence of an object without interacting with it, called interaction-free measurement in quantum mechanics.

Now one may say, that the chance of explosion is too high and too insecure for a real application, since the probability for an explosion is $\frac{1}{2}$. Fortunately there is an easy fix which drops the chance of explosion close to zero. For this the half-silvered plates needs to be changed, in such a way that the chance for a single photon to be transmitted is way higher than being reflected. So the changed unitary will look like

$$H_1 = \begin{pmatrix} a & (i \cdot b) \\ (i \cdot b) & -a \end{pmatrix}$$

for the first beam splitter and for the second beam splitter like

$$H_2 = \begin{pmatrix} b & (i \cdot a) \\ (i \cdot a) & -b \end{pmatrix}$$

for $a, b \in \mathbb{R}_{>0}$ and a much bigger than b.
The outcomes are

$$H_2 \cdot H_1 \cdot |0\rangle = H_2 \cdot (a|0\rangle + ib|1\rangle) \xrightarrow[\text{and can be omitted}]{|1\rangle \text{ hits the bomb}} H_2 \cdot a|0\rangle = ab|0\rangle + ia^2|1\rangle$$

where $|0\rangle$ is the outcome for measuring in B and $|1\rangle$ for measuring in A. The corresponding probabilities for the Post measurement states are:

-	Chance
No Photon detected	b
Photon detected at A	aa
Photon detected at B	ab

Table 7: Elitzur-Vaidman bomb tester outcomes when 2 photons are sent through

When repeating this experiment often enough, the probability of the outcomes only get better and the chance of exploding the bomb gets smaller each time [21].

A.3 Shor's Algorithm

In order to solve the problem of period finding, Shor used the concept of reduction. He presented a solution to another problem and then demonstrated how this solution could be adapted to be used for period finding. In this case, the problem of order finding was used as a basis for this approach.

Definition 17 (Order Finding Problem)**.** *Given a prime p, a generator g of the multiplicative group $(mod\, p)$ and an $x\ (mod\, p)$, find an r such that*

$$g^r \equiv x (mod\, p)$$

[55]

Reduction from order finding to integer factorization

Given: A positive odd integer N

Goal: Compute integer factorization of N

1. Pick a random $x \in \{1, \ldots, N-1\}$
2. Compute the order r_x of x with the order finding algorithm
3. Abort if r_x is odd or $x^{r_x/2} - 1 \equiv -1 \,(mod\, N)$
4. Compute $x = gcd(x^{r_x/2} - 1, N)$ and $y = gcd(x^{r_x/2} - 1, N)$
5. Check if x or y is a non-trivial factor of N. If so, the algorithm succeded [55][43].

Then the algorithm finds a factor of N with at least the probability of $1 - \frac{1}{2^k}$ where k is the number of distinct prime factors of N. This only counts under the assumption that N is odd and not a prime power.

What if N is a prime power?
The factorization of prime powers can be done efficiently using classical methods.

What if N is even?
Divide N by 2 until an uneven number comes out. Then apply Shor's algorithm for order finding.

Now that we now how to reduce the problem of integer factorization to order finding, we can take a look at the order finding algorithm.

Shors algorithm for order finding

Given: Random integer $x \in \{1, \ldots, N-1\}$ and odd N

Find: Order r of x mod N

1. Find a q with $2N^2 \leq q \leq 4N^2$
2. Change the input to superposition over all q

$$|0^q\rangle \Rightarrow \frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a\rangle$$

3. Compute $x^a \,(mod\, N)$ on an auxiliary wire

$$\frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a, x^a \,(mod\, N)\rangle$$

4. Apply the Quantum Fourier Transform

$$\frac{1}{q}\sum_{a=0}^{q-1} e^{2\pi iac/q}|c, x^a \,(mod\, N)\rangle$$

5. Measuring the wire. It would be enough to just measure the top wire, but for completeness, also the lower wire is measured. Looking now at a specific state k, then the probability for such a state k and all $a \equiv k \,(mod\, N)$ is:

$$\left|\frac{1}{q}\sum_{a:x^a \equiv x^k \,(mod\, N)} e^{2\pi iac/q}\right|^2$$

Lemma 2. *Given r is the order of x with respect to modulo N. When $x^a \equiv x^b \,(mod\, N)$ with $a, b \in \mathbb{N}$,then this is equal to*

$$x^a \cdot x^{-b} \equiv 1 \,(mod\, N) \Leftrightarrow x^{a-b} \equiv 1 \,(mod\, N)$$

And since r is the smallest $\mathbb{N}$, such that $x^r \equiv 1 \,(mod\, N)$ it follows that $a - b$ is a multiple of r and therefore $a - b = kr$ for some $k \in \mathbb{N} \Rightarrow a = b + kr \Rightarrow a \equiv b \,(mod\, r)$

Now this is not the wanted solution, so classical post-processing is necessary. Because of 2 the output of the algorithm can be rewritten to

$$\left|\frac{1}{q}\sum_{a:a \equiv k \,(mod\, r)} e^{2\pi iac/q}\right|^2$$

Then swapping a with $br + k$ results in

$$\left|\frac{1}{q}\sum_{b=0}^{\lfloor (q-k-1)/r \rfloor} e^{2\pi i(br+k)c/q}\right|^2$$

After dissolving the bracket, $e^{2\pi ikc/q}$ can be ignored, because it can be factored out of the sum and has a magnitude of 1. Furthermore, we can exchange rc to its corresponding residue $\{rc\}_q$, which is in the range $-q/2 < \{rc\}_q \leq q/2$ and because of the residue also congruent to $rc \,(mod\, q)$.

$$\left|\frac{1}{q}\sum_{b=0}^{\lfloor (q-k-1)/r \rfloor} e^{2\pi ib\{rc\}_q/q}\right|^2$$

If $\{rc\}_q$ is small with relation to q, then we can use the Riemann summation method to convert it to an integral by change of variables $t = b/q$ which results in the integral:

$$\left|\int_0^{\frac{1}{q}\lfloor (q-k-1)/r \rfloor} e^{2\pi i\{rc\}_q t} dt\right|^2$$

This way it can be shown that the quantity can be asymptotically bounded below by $\frac{4}{\pi^2 r^2}$, so at least $\frac{1}{3r^2}$ if $|\{rc\}_q| \leq \frac{r}{2}$. Therefore, the probability of a seeing a given state $|c, x^k (mod n)\rangle$ is at least $\frac{1}{3r^2}$ if $\frac{-r}{2} \leq rc (mod 2^n) \leq \frac{r}{2}$. Or in other words, when given a d then $\frac{-r}{2} \leq rc - dq \leq \frac{r}{2}$. Rearranging rq results in $|\frac{c}{q} - \frac{d}{r}| \leq \frac{1}{2q}$. c and q are already known. And because $2n^2 \leq q$ and $r < n$ it follows that there is at most one fraction $\frac{d}{r}$, which satisfys the inequality 3.

Lemma 3. *If there is more than one solution then there are two $\frac{d_1}{r_1}$ and $\frac{d_2}{r_2}$ which satisfy the inequality. The difference of these two solutions would be*

$$\left|\frac{d_1}{r_1} - \frac{d_2}{r_2}\right| = \left|(\frac{c}{q} - \frac{d_1}{r_1}) - (\frac{c}{q} - \frac{d_2}{r_2})\right| \leq \frac{1}{2q} + \frac{1}{2q} = \frac{1}{q}$$

Now this multiplied by $r_1 \cdot r_2$ results in

$$|d_1 r_2 - d_2 r_1| \leq \frac{r_1 r_2}{q}$$

And since $r_1, r_2 < n$ and $2n^2 \leq q$ it follows that

$$\frac{r_1 r_2}{q} \leq \frac{n^2}{2n^2} = \frac{1}{2}$$

It is known that d_1, d_2, r_1, r_2 are all integers and therefore $|d_1 r_2 - d_2 r_1|$ is also an integer. The only integer solving the inequation $|d_1 r_2 - d_2 r_1| = \frac{1}{2}$ is 0, and therefore $d_1 r_2 = d_2 r_1$, which implies $d_1/r_1 = d_2/r_2$.

We can get $\frac{d}{r}$ now by the continues fraction expansion of $\frac{c}{q}$, and therefore also get r but only when d is relative prime to r. To see how often it is necessary to make these steps in order to find r with a sufficient high probability, we need to count the states $|c, x^a (mod N)\rangle$. There are $\phi(r)$ possible values for d, with each close to $\frac{c}{q}$ with $|\frac{c}{q} - \frac{d}{r}| \leq \frac{1}{2q}$. For x^k there are r possible values, since r is the order of x. Therefore, there are $r\ \phi(r)$ possible states $|c, x^a\ (mod\ N)\rangle$. Since each of these states occurs with probability at least $\frac{1}{3r^2}$, r will be obtained with a probability of at least $\frac{\phi(r)}{3r}$. With theorem A.3.1 it is shown that $\frac{\phi(r)}{r} > \frac{k}{log\ log\ r}$ and therefore finding r at least $\frac{k}{log\ log\ r}$ times. So by repeating this experiment $\mathcal{O}(log\ log\ r)$ times, there is a high probability of getting the desired order r of x [55].

Theorem A.3.1. *$\underline{lim} \frac{\phi(n)\ log\ log\ n}{n} = e^{-C}$ where C is the Euler-Mascheroni constant and e is the Euler number.*

Therefore, for a sufficient high N, it holds that $\frac{\phi(n)}{n} \approx \frac{e^{-C}}{log\ log\ n}$. Subtracting $\epsilon > \left|\frac{\phi(n)}{n} - \frac{e^{-C}}{log\ log\ n}\right|$ from the right side results in $\frac{\phi(n)}{n} > \frac{e^{-C} - \epsilon}{log\ log\ n}$. For $k = e^{-C} - \epsilon$ it follows that $\frac{\phi(n)}{n} > \frac{k}{log\ log\ n}$. [27]

References

[1] Simon Bausch. *Implementation of the PQC algorithm BIKE in theorem prover EasyCrypt*. Nov. 12, 2024. URL: https://gitlab.com/Simon.Bausch/implementation-of-the-pqc-algorithm-bike-in-theorem-prover-easycrypt.

[2] Gorjan Alagic et al. *Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process*. NIST IR 8413-upd1. Includes updates as of 09-26-2022. National Institute of Standards and Technology, July 2022. URL: https://nvlpubs.nist.gov/nistpubs/ir/2022/NIST.IR.8413-upd1.pdf.

[3] José Bacelar Almeida et al. *Formally verifying Kyber Episode IV: Implementation Correctness*. Cryptology ePrint Archive, Paper 2023/215. 2023. URL: https://eprint.iacr.org/2023/215.

[4] Benny Applebaum, Boaz Barak, and Avi Wigderson. "Public-key cryptography from different assumptions". In: *Proceedings of the forty-second ACM symposium on Theory of computing*. 2010, pp. 171–180.

[5] Nicolas Aragon et al. *BIKE - Bit Flipping Key Encapsulation*. URL: https://bikesuite.org/ (visited on 10/26/2024).

[6] Manuel Barbosa et al. *Fixing and Mechanizing the Security Proof of Fiat-Shamir with Aborts and Dilithium*. Cryptology ePrint Archive, Paper 2023/246. 2023. URL: https://eprint.iacr.org/2023/246.

[7] Manuel Barbosa et al. *Machine-Checked Security for* XMSS *as in RFC 8391 and* SPHINCS$^+$. Cryptology ePrint Archive, Paper 2023/408. 2023. DOI: 10.1007/978-3-031-38554-4_14. URL: https://eprint.iacr.org/2023/408.

[8] Gilles Barthe et al. "EasyCrypt: A Tutorial". In: *Foundations of Security Analysis and Design VII: FOSAD 2012/2013 Tutorial Lectures*. Ed. by Alessandro Aldini, Javier Lopez, and Fabio Martinelli. Cham: Springer International Publishing, 2014, pp. 146–166. ISBN: 978-3-319-10082-1. DOI: 10.1007/978-3-319-10082-1_6. URL: https://doi.org/10.1007/978-3-319-10082-1_6.

[9] Mihir Bellare and Phillip Rogaway. "Introduction to modern cryptography". In: *Lecture Notes* (2001).

[10] Marcel Berger. *Geometry. I, II. Transl. from the French by M. Cole and S. Levy*. English. Universitext. Springer, Cham, 1987. ISBN: 3-540-11658-3; 3-540-17015-4.

[11] E. Berlekamp, R. McEliece, and H. van Tilborg. "On the inherent intractability of certain coding problems (Corresp.)" In: *IEEE Transactions on Information Theory* 24.3 (1978), pp. 384–386. DOI: 10.1109/TIT.1978.1055873.

[12] Albrecht Beutelspacher, Jörg Schwenk, and Klaus-Dieter Wolfenstetter. *Moderne Verfahren der Kryptographie: von RSA zu Zero-Knowledge und darüber hinaus*. Springer, 2022.

[13] Dan Boneh et al. "Twenty years of attacks on the RSA cryptosystem". In: *Notices of the AMS* 46.2 (1999), pp. 203–213.

[14] University of Cambridge and Technische Universität München. *Isabelle*. URL: https://isabelle.in.tum.de (visited on 10/28/2024).

[15] Wouter Castryck and Thomas Decru. *An efficient key recovery attack on SIDH (preliminary version)*. Cryptology ePrint Archive, Paper 2022/975. https://eprint.iacr.org/2022/975. 2022. URL: https://eprint.iacr.org/2022/975.pdf.

[16] T. Charles Clancy, Robert W. McGwier, and Lidong Chen. "Post-quantum cryptography and 5G security: tutorial". In: *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*. WiSec '19. Miami, Florida: Association for Computing Machinery, 2019, p. 285. ISBN: 9781450367264. DOI: 10.1145/3317549.3324882. URL: https://doi.org/10.1145/3317549.3324882.

[17] Mauro Conti, Nicola Dragoni, and Viktor Lesyk. "A Survey of Man In The Middle Attacks". In: *IEEE Communications Surveys and Tutorials* 18.3 (2016), pp. 2027–2051. DOI: 10.1109/COMST.2016.2548426.

[18] Microsoft Corporation. *Introduction*. URL: https://microsoft.github.io/z3guide/docs/logic/intro/ (visited on 10/28/2024).

[19] Joan Daemen and Vincent Rijmen. *The design of Rijndael*. Vol. 2. Springer, 2002.

[20] Nir Drucker, Shay Gueron, and Dusan Kostic. *Fast polynomial inversion for post quantum QC-MDPC cryptography*. Cryptology ePrint Archive, Paper 2020/298. 2020. URL: https://eprint.iacr.org/2020/298.

[21] Avshalom C Elitzur and Lev Vaidman. "Quantum mechanical interaction-free measurements". In: *Foundations of physics* 23 (1993), pp. 987–997.

[22] Pierre-Alain Fouque et al. *Key Recovery from Gram-Schmidt Norm Leakage in Hash-and-Sign Signatures over NTRU Lattices*. Cryptology ePrint Archive, Paper 2019/1180. 2019. URL: https://eprint.iacr.org/2019/1180.

[23] Eiichiro Fujisaki and Tatsuaki Okamoto. "Secure Integration of Asymmetric and Symmetric Encryption Schemes". In: *Journal of Cryptology* 26.1 (2013), pp. 80–101. ISSN: 1432-1378. DOI: 10.1007/s00145-011-9114-1. URL: https://doi.org/10.1007/s00145-011-9114-1.

[24] Andrea Galimberti et al. "FPGA implementation of BIKE for quantum-resistant TLS". In: *2022 25th Euromicro Conference on Digital System Design (DSD)*. 2022, pp. 539–547. DOI: 10.1109/DSD57027.2022.00078.

[25] Shai Halevi. "A plausible approach to computer-aided cryptographic proofs." In: *IACR Cryptology ePrint Archive* 2005 (Jan. 2005), p. 181.

[26] Darrel Hankerson and Alfred Menezes. "Elliptic Curve Cryptography". In: *Encyclopedia of Cryptography, Security and Privacy*. Ed. by Sushil Jajodia, Pierangela Samarati, and Moti Yung. Berlin, Heidelberg: Springer Berlin Heidelberg, 2019, pp. 1–2. ISBN: 978-3-642-27739-9. DOI: 10.1007/978-3-642-27739-9_245-2. URL: https://doi.org/10.1007/978-3-642-27739-9_245-2.

[27] G. H. Hardy and E.M Wright. *An Introduction to the theory of numbers / by G. H. Hardy and E. M. Wright*. Oxford: Clarendon Pr., 1960.

[28] Matthias Homeister. *Quantum Computing verstehen: Grundlagen – Anwendungen – Perspektiven*. German. 6th ed. Computational Intelligence. 78 b/w illustrations, 21 illustrations in colour. Wiesbaden: Springer Vieweg, 2022, pp. XIII, 336. ISBN: 978-3-658-36433-5. DOI: 10.1007/978-3-658-36434-2.

[29] Samuel Jaques and John M. Schanck. "Quantum Cryptanalysis in the RAM Model: Claw-Finding Attacks on SIKE". In: *Advances in Cryptology – CRYPTO 2019*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Cham: Springer International Publishing, 2019, pp. 32–61. ISBN: 978-3-030-26948-7.

[30] Thorsten Kleinjung et al. "Factorization of a 768-bit RSA modulus". In: *IACR Cryptology ePrint Archive* 2010 (2010), p. 6.

[31] Katharina Kreuzer. *Verification of Correctness and Security Properties for CRYSTALS-KYBER*. Cryptology ePrint Archive, Paper 2023/087. 2023. URL: https://eprint.iacr.org/2023/087.

[32] Oleksandr Kuznetsov et al. "Trade-offs in Post-Quantum Cryptography: A Comparative Assessment of BIKE, HQC, and Classic McEliece." In: *CQPC*. 2023, pp. 1–11.

[33] Miroslav Laššák and Štefan Porubský. "Fermat-Euler Theorem in Algebraic Number Fields". In: *Journal of Number Theory* 60.2 (1996), pp. 254–290. ISSN: 0022-314X. DOI: https://doi.org/10.1006/jnth.1996.0123. URL: https://www.sciencedirect.com/science/article/pii/S0022314X96901237.

[34] Mayank Mahajan et al. "Comparative Analysis of Bit Flipping Decoders in BIKE PQC". In: *Advances in Data-Driven Computing and Intelligent Systems*. Ed. by Swagatam Das et al. Singapore: Springer Nature Singapore, 2024, pp. 345–356. ISBN: 978-981-99-9531-8.

[35] Ueli M. Maurer and Stefan Wolf. "The Diffie–Hellman Protocol". In: *Designs, Codes and Cryptography* 19.2-3 (2000), pp. 147–171. DOI: 10.1023/A:1008302122286.

[36] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 1997.

[37] Daniele Micciancio and Shafi Goldwasser. *Complexity of Lattice Problems: A Cryptographic Perspective*. The Springer International Series in Engineering and Computer Science. Hardcover ISBN: 978-0-7923-7688-0, Softcover ISBN: 978-1-4613-5293-8, eBook ISBN: 978-1-4615-0897-7. New York, NY: Springer, 2002. ISBN: 978-0-7923-7688-0. DOI: 10.1007/978-1-4615-0897-7.

[38] Microsoft Research. *BIKE: Bit Flipping Key Encapsulation*. https://bikesuite.org/files/v5.0/BIKE_Spec.2022.10.10.1.pdf. Accessed on February 29, 2024. 2022.

[39] National Institute of Standards and Technology. *Module-Lattice-Based Digital Signature Standard*. Federal Information Processing Standards Publication 203. U.S. Department of Commerce, National Institute of Standards and Technology, 2024. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.204.pdf.

[40] National Institute of Standards and Technology. *Module-Lattice-Based Key-Encapsulation Mechanism Standard*. Federal Information Processing Standards Publication 203. U.S. Department of Commerce, National Institute of Standards and Technology, 2024. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.203.pdf.

[41] National Institute of Standards and Technology. *Stateless Hash-Based Digital Signature Standard*. Federal Information Processing Standards Publication 203. U.S. Department of Commerce, National Institute of Standards and Technology, 2024. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.205.pdf.

[42] National Institute of Standards and Technology. *Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process*. Call for Proposals. National Institute of Standards and Technology, Dec. 2016. URL: https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf.

[43] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[44] Dusan Kostic Nir Drucker Shay Gueron and Edoardo Persichetti. "On the applicability of the Fujisaki–Okamoto transformation to the BIKE KEM". In: *International Journal of Computer Mathematics: Computer Systems Theory* 6.4 (2021), pp. 364–374. DOI: 10.1080/23799927.2021.1930176. eprint: https://doi.org/10.1080/23799927.2021.1930176. URL: https://doi.org/10.1080/23799927.2021.1930176.

[45] Spencer Wilson Nir Drucker. *Additional implementation of BIKE (Bit Flipping Key Encapsulation)*. URL: https://github.com/awslabs/bike-kem (visited on 10/26/2024).

[46] Mohammad Reza Nosouhi et al. "Bit Flipping Key Encapsulation for the Post-Quantum Era". In: *IEEE Access* 11 (2023), pp. 56181–56195. DOI: 10.1109/ACCESS.2023.3282928.

[47] Mohammad Reza Nosouhi et al. "Weak-Key Analysis for BIKE Post-Quantum Key Encapsulation Mechanism". In: *IEEE Transactions on Information Forensics and Security* 18 (2023), pp. 2160–2174. DOI: 10.1109/TIFS.2023.3264153.

[48] Harry Paul. *Experimente und ihre Deutung*. Berlin, Boston: De Gruyter, 1985. ISBN: 9783112595824. DOI: doi:10.1515/9783112595824. URL: https://doi.org/10.1515/9783112595824.

[49] J.H. van de Pol. "Lattice-based Cryptography". M.Sc. Thesis. Eindhoven University of Technology, Department of Mathematics and Computer Science, 2024.

[50] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. Aug. 2018. DOI: 10.17487/RFC8446. URL: https://www.rfc-editor.org/info/rfc8446.

[51] Jan Richter-Brockmann. *Folding BIKE: Scalable Hardware Implementation for Reconfigurable Devices*. URL: https://github.com/Chair-for-Security-Engineering/BIKE (visited on 10/26/2024).

[52] Simon Rubinstein-Salzedo. *Cryptography*. 1st ed. Springer Undergraduate Mathematics Series. 10 b/w illustrations, 7 illustrations in colour. Cham: Springer, 2018, pp. XII, 259. ISBN: 978-3-319-94817-1. DOI: 10.1007/978-3-319-94818-8.

[53] Isa Sedlacek, Nicole Keller, and Karen Scarfone. *NIST IR 8240 : Status Report on the First Round of the NIST Post-Quantum Cryptography Standardization Process*. Tech. rep. 8240. National Institute of Standards and Technology, 2018. DOI: 10.6028/NIST.IR.8240.

[54] Tejas Anil Shah. "The Joy of EasyCrypt: The Least Painful Way to Learn Formal Verification of Cryptography". 30 ECTS. MA thesis. Tartu: University of Tartu, Faculty of Science and Technology, Institute of Computer Science, Computer Science Curriculum, 2022.

[55] P.W. Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[56] Rajeev Sobti and Ganesan Geetha. "Cryptographic hash functions: a review". In: *International Journal of Computer Science Issues (IJCSI)* 9.2 (2012), p. 461.

[57] National Institute of Standards and Technology. *Announcing Request for Nominations for Public-Key Post-Quantum Cryptographic Algorithms*. Dec. 20, 2016. URL: https://csrc.nist.gov/News/2016/Public-Key-Post-Quantum-Cryptographic-Algorithms (visited on 09/24/2024).

[58] National Institute of Standards and Technology. *NIST Announce the Release of DRAFT NISTIR 8105, Report on Post-Quantum Cryptography for Public Comment*. Tech. rep. Feb. 3, 2016. URL: https://csrc.nist.gov/News/2016/NIST-Announce-the-Release-of-DRAFT-NISTIR-8105 (visited on 09/24/2024).

[59] National Institute of Standards and Technology. *Post Quantum Cryptography Standardization: Announcement and outline of NIST's Call for Submissions*. Feb. 24, 2016. URL: https://csrc.nist.gov/Presentations/2016/Announcement-and-outline-of-NIST-s-Call-for-Submis (visited on 09/25/2024).

[60] National Institute of Standards and Technology. *Post-Quantum Cryptography PQC*. URL: https://csrc.nist.gov/projects/post-quantum-cryptography (visited on 10/26/2024).

[61] National Institute of Standards and Technology. *PQC Standardization Process: Third Round Candidate Announcement*. July 20, 2020. URL: https://csrc.nist.gov/News/2020/pqc-third-round-candidate-announcement (visited on 09/26/2024).

[62] National Institute of Standards and Technology. *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*. June 13, 2006. URL: https://csrc.nist.gov/pubs/sp/800/90/final (visited on 09/25/2024).

[63] National Institute of Standards and Technology. *Security (Evaluation Criteria)*. Sept. 25, 2024. URL: https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/evaluation-criteria/security-(evaluation-criteria)#FN2 (visited on 09/26/2024).

[64] Alley Stoughton. *BIKE - 5th NIST PQC Standardization Workshop*. Apr. 10, 2024. URL: https://bikesuite.org/files/5th-nist-workshop-bike-presentation.pdf (visited on 10/24/2024).

[65] Alley Stoughton. *EasyCrypt Installation Instructions*. URL: https://alleystoughton.us/easycrypt-installation.html (visited on 10/24/2024).

[66] New York Times. *Government Announces Steps to Restore Confidence on Encryption Standards*. Sept. 10, 2013. URL: https://archive.nytimes.com/bits.blogs.nytimes.com/2013/09/10/government-announces-steps-to-restore-confidence-on-encryption-standards/ (visited on 09/25/2024).

[67] Joshua J Tom et al. "Quantum computers and algorithms: a threat to classical cryptographic systems". In: *International Journal of Engineering and Advanced Technology* 12.5 (2023), pp. 25–38.

[68] Ming-Hsien Tsai et al. *Automatic Certified Verification of Cryptographic Programs with COQCRYPTOLINE*. Cryptology ePrint Archive, Paper 2022/1116. 2022. URL: https://eprint.iacr.org/2022/1116.

[69] Tianrui Wang, Anyu Wang, and Xiaoyun Wang. "Exploring Decryption Failures of BIKE: New Class of Weak Keys and Key Recovery Attacks". In: *Annual International Cryptology Conference*. Cham: Springer Nature Switzerland, 2023.

[70] Y. S. Weinstein et al. "Implementation of the Quantum Fourier Transform". In: *Phys. Rev. Lett.* 86 (9 Feb. 2001), pp. 1889–1891. DOI: 10.1103/PhysRevLett.86.1889. URL: https://link.aps.org/doi/10.1103/PhysRevLett.86.1889.

[71] Thomas Young. "The Bakerian Lecture. Experiments and calculations relative to physical optics". In: *Philosophical Transactions of the Royal Society* 94 (1804), pp. 1–16. DOI: 10.1098/rstl.1804.0001. URL: http://doi.org/10.1098/rstl.1804.0001.