Exchanging Keys Using McEliece

Udveksling Af Nøgler Ved Brug Af McEliece

Rune N. T. Thorsen
201505509 - AU522740

Master's Thesis in Computer Science
June 2023

Supervisor:
Ivan Bjerre Damgård

Department of Computer Science
Aarhus University

Colophon

Exchanging Keys Using McEliece

— Udveksling Af Nøgler Ved Brug Af McEliece

Master's thesis by Rune N. T. Thorsen. Written under supervision by Professor Damgård, Department of Computer Science, Aarhus University.

Typeset by the author with LaTeX and the memoir document class, using Linux Libertine and Linux Biolinum 11.0/13.6pt.

Dedication

This work is dedicated to my father who was the first to spark my interest in computers and tell me about the world of cryptology.

Acknowledgements

I would like to thank my supervisor Ivan, because he has always been available, able to understand all of my struggles and answer mostly any question that I could throw at him. If not, he would rapidly be able to find an answer. I will also extend my gratitude to Mads for reading through my thesis and making suggestions as to where it could be improved. Alongside Mads I will thank Jakob for helping me with LaTeX. Lastly Magdalena, Michael and Nishka will also have to be thanked for providing a good day whenever I went to my office. I have enjoyed “going to work” with these people throughout the semester and especially also the professional sparring that have occurred between us.

Abstract (English)

In this thesis I will introduce linear error correcting codes and in particular Goppa codes with a focus on irreducible binary Goppa codes. These will be used later for building the original McEliece public key cryptosystem. After this, some security concerns of the original cryptosystem will be laid out and in particular, I will show that it does not live up to CCA2 security. This will be rectified in the case where it is to be used for key exchanging, first in the classical sense and then later in the sense of the quantum random oracle model.

Resumé (Dansk)

I dette speciale vil jeg introducere lineære fejlkorrigerende koder og i særdeleshed Goppa-koder med et fokus på irreducible binære Goppa-koder. Disse vil blive brugt senere til at bygge det originale McEliece offentlig nøgle kryptosystem. Herefter vil nogle sikkerhedsproblemer for det originale kryptosystem blive udlagt, og jeg vil især vise, at det ikke lever op til CCA2-standarden for sikkerhed. Dette vil blive rettet i forbindelse med nøgleudveksling, først i klassisk forstand og derefter senere i betydningen af den kvantetilfældige orakelmodel.

Preface

This thesis concludes my Master's degree in Computer Science at Aarhus University. It is with great satisfaction, that it has been done in the realm of cryptography, which itself is the reason that I wanted to study computer science in the first place. It has been a long and arduous journey, but finally I will have written a thesis and soon I will have completed my studies. This is then also the culmination of having spent 8 years at the same institution and having undertaken lots of things that were relevant to my studies and perhaps even more that were not. I am immensely grateful for all the people that I have met along the way. They must number in the hundreds if not almost thousands and have all had some sort of impact upon me. Having to leave Aarhus University seems a bit unreal. I would be lying, if I were to say, that I did not want to stay, but I must also say that I am glad to finalise my total of 21 years of education. Wish me luck and I will wish that upon you too.

Contents

Introduction

Suppose you have to send a message to your friend at the other part of town, but your mutual enemy works at the post office. You do not want this person to be able to steam your letters and read them. In this scenario you could send a manual to your friend, letting them know how to build a box and a padlock for the box. This box could then contain the message that you wish your friend to send to you. Your friend could build the box and the padlock and then put a message in the box and use the padlock to seal the box. The box is then sent to you and if you are smart, you would have designed the padlock such that only you know to build the key for it (even in the case that your friend and your enemy also have the instructions on how the padlock is built). This means that your enemy will not be able to sniff your messages that you friend sends to you and your friend would have to commit to the message, that he or she puts in the box. This is essentially what public key encryption is. Your friend could then send you instructions on how to build another padlock alongside how to make the key for it. Using this new padlock, you could send messages back and forth but only the two of you will have the key for unlocking the messages. This is the idea behind symmetric encryption schemes and is analogous to how public key encryption schemes are used in real life.

This thesis is then about one of the oldest public key encryption schemes, namely the McEliece public key encryption scheme. It has not yet been broken in a classical- or a quantum sense and so it is of great interest to the cryptographic community, as it seems that the emergence of general quantum computers is almost imminent.

Before having a look at that though, I will have to go over some coding theory and also lay out Goppa codes – in particular irreducible binary Goppa codes, as these are the ones that will be used for building the cryptosystem.

The cryptosystem does not however live up to modern standards of security of public key cryptosystems. This is not because it is old and outdated in a sense that current technology has surpassed it, but simply because it never did and back in the day it was most likely not something most people concerned themselves with. As I pointed out earlier, this was one of the oldest public key cryptosystems. This mistake can be rectified when using it for exchanging keys – even in the quantum sense – and I will show how.

Please note that all relevant notation will be introduced when necessary. The reader should notice that some of it might be overloaded, but whenever in doubt, please take context into account. Not every single piece of notation is formally introduced though. This is because there are several basic concepts of mathematics and computer science that I will presuppose that the reader has knowledge of and thus also know the common notation of.

As for the concepts that I presuppose that the reader is familiar with, these include some abstract algebra (including linear algebra), combinatorics, probability theory and complexity theory. The goal is that if a person with a bachelor's degree in computer science has had a course in abstract algebra (and maybe also one in cryptology), he or she would be able to follow along.

Chapter 1

Linear Error Correcting Codes

This chapter is built in just about the same way as chapter 3 in [31] (which is then also the source for the entire chapter). There is an exception to this however. This chapter excludes everything after section 3.2 in [31]. This is because I will limit my focus to just introducing linear error correcting codes, since this is what is needed in order to understand chapter 2.

In section 1.1 I will quickly introduce why error correcting codes are of desire, besides being useful for understanding Goppa codes. This is followed by section 1.2 where I will introduce the notion of block codes and some important definitions in relation to them. Later in section 1.3 I will use what has already been introduced along with some more definitions to fully define what a linear error correcting code is. All of these introductions will also lead to an important result regarding a special property of linear error correcting codes.

1.1 Why Error Correcting Codes?

Suppose a message is sent over a *noisy channel*. This noisy channel will have a non-zero probability of introducing errors at every position in the received message. For instance one might have a bit-flip in a particular bit in said message. This is the problem that error correcting codes tries to solve. All such errors are assumed to be independent though. That is for a message of length q it is assumed that an error that has been introduced in position i in such a received message has no influence on the remaining $q - 1$ symbols of the message. The remaining symbols of the message will thus still have a probability of an error with equal probability (that is to say that any and all symbols in a message sent over a noisy channel will have a uniform probability of having an error introduced).

1.2 Block Codes

Throughout this chapter it will be assumed that information is coded using some arbitrary alphabet $\mathcal{Q}$ with q distinct symbols. By introducing the notation that $|\bullet|$ denotes the function that takes a set as an argument and returns the amount of distinct elements in the set, it must then be true that $|\mathcal{Q}| = q$.

Suppose you have blocks of n symbols that can all be decoded independently. Call n the *block length* or *word length* (or even just *length*) and call the blocks *codewords*. A code that can decode these blocks independently will be called a *block code*. Now codewords of this form will have the general notation $\mathbf{x} \in Q^n$ meaning that $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ is a codeword of length n but is at the same time to be considered as a vector with entries corresponding to the symbols in the given word.

In order to further investigate some interesting properties of block codes, some more notation is needed. Let the vector $(0, 0, \cdots, 0)$ be denoted by $\mathbf{0}$. This new notation helps us introduce our first set of definitions, that of the Hamming distance and of the Hamming weight.

Definition 1.1 (Hamming Distance And Hamming Weight) *Let* $\mathbf{x}, \mathbf{y} \in Q^n$ *then the* distance *(or* Hamming distance*)* $d(\mathbf{x}, \mathbf{y})$ *of* $\mathbf{x}$ *and* $\mathbf{y}$ *is defined as*

$$d(\mathbf{x}, \mathbf{y}) = |\{i \mid 1 \le i \le n,\ x_1 \ne y_i\}|.$$

Now the weight *(or* Hamming weight*)* $w(\mathbf{x})$ *of* $\mathbf{x}$ *can be defined in terms of the distance function:*

$$w(\mathbf{x}) = d(\mathbf{x}, \mathbf{0}).$$

It is not hard to see that the hamming distance is a metric on Q^n. Already now the relevance of this metric can be seen from what I started this chapter out with: if a message is sent over a noisy channel, then the Hamming distance can be used to measure how many errors are in a received message over said channel (provided one has access to both the original and the received message of course).

Now let C be a *code* that is a nonempty proper subset of the alphabet Q'', in a more mathematical notation one would then write $\emptyset \ne C \subset Qn$. In the case that $|C| = 1$ the code is *trivial*. If $q = 2$ then the code is called *binary*, if $q = 3$ then the code is *ternary*, etc. Now I am ready to move on a bit further again. Time for some more definitions. This first one should be intuitive enough.

Definition 1.2 (Minimum Distance And Minimum Weight) *The* minimum distance *of a nontrivial code* C *is*

$$\min\{d(\mathbf{x}, \mathbf{y}) \mid \mathbf{x} \in C,\ \mathbf{y} \in C,\ \mathbf{x} \ne \mathbf{y}\}.$$

The minimum weight *of* C *is*

$$\min\{w(\mathbf{x}) \mid \mathbf{x} \in C,\ \mathbf{x} \ne \mathbf{0}\}.$$

This next definition is rather interesting though and at first glance it might not be clear what it actually is that is being defined.

Definition 1.3 (Information Rate) *If* $|Q| = q$ *and* $C \subset Q''$ *then*

$$R = n^{-1} \log_q |C|$$

is called the information rate *of* C.

Definition 1.3 is worth having a short look at. The information rate that is being defined is essentially just a measure of how much of a message is not redundant. That is it is a fraction that represents how much of a given received message is useful information with the remainder of the message being used to do error correction.

Remember that in definition 1.1 there is a clear definition of something called the Hamming distance. It might differ somewhat from what everyone is used to think of as a distance, but if one accepts that it is indeed a distance, then one can also use it to define other things that can be defined from such a measurement. One of them is a radius.

Definition 1.4 (Covering Radius) *If $C \subset Q^n$ then the* covering radius $\rho(C)$ *of C is*

$$\max\{\min\{d(\mathbf{x},\mathbf{c}) \mid \mathbf{c} \in C\} \mid \mathbf{x} \in \mathbb{Q}^n\}.$$

Now let ρ be a radius of a sphere $B_\rho(\mathbf{x})$ with center $\mathbf{x}$. The sphere can be described as the set

$$\{\mathbf{y} \in Q^n \mid d(\mathbf{x},\mathbf{y}) \leq \rho\}.$$

It is even possible to define a set of such spheres as $\left\{B_\rho(\mathbf{c}) \,\middle|\, \mathbf{c} \in C\right\}$. Imagine if you will that ρ is the largest integer radius such that the spheres are disjoint. Now it must be true that either $d = 2\rho + 1$ or $d = 2\rho + 2$. To see this consider two spheres that expand equally much from their respective centres until they are adjacent. Now since they must not overlap either they just exactly do not (leading to the case where $d = 2\rho + 1$) or they start overlapping and must both contract 1 unit length (leading to the case where $d = 2\rho + 2$).

Now the covering radius is the smallest ρ such that that the spheres in the set $\left\{B_\rho(\mathbf{c}) \,\middle|\, \mathbf{c} \in C\right\}$ cover all of Q^n. If $\rho(C) = \rho$ then the code C is said to be *perfect*. This leads to definition 1.5.

Definition 1.5 (Perfect Code) *A code $C \subset Q^n$ with minimum distance $2e + 1$ is called a* perfect code *if every $\mathbf{x} \in Q^n$ has distance $\leq e$ to exactly one codeword.*

Definition 1.5 actually just implies that any perfect error correcting block code can correct up to e errors introduced by a noisy channel. Call such a code C that can correct up to e errors an *e-error-correcting code* and if it is also perfect, then call it a *perfect e-error-correcting code.*

Now denote by $\binom{\cdot}{\cdot}$ the binomial coefficient. The following result comes from combinatorics.

Corollary 1.6 (Sphere-Packing Condition) *If $C \subset Q^n$ is a perfect e-error correcting code then*

$$|C| \sum_{i=0}^{e} \binom{n}{i} (q-1)^i = q^n.$$

Corollary 1.6 basically just implies that it takes q^n spheres to cover all of Q^n. This means that there must be q^n different block codes on the alphabet Q^n that can correct up to e errors.

Note that a trivial code will be trivially perfect. This is because the distance between an element and itself is 0.

1.3 Linear Codes

Linear error correcting codes are essentially a special case of block codes. I will try to define what a linear code actually is, but doing so is not as easy as it was for block codes. That is because it is now required that the code in question needs to have some actual algebraic structure and not be treated as a general case, as block codes were. Fortunately this is entirely possible to do.

First have a group Q and a subgroup C of Q^n as a code. This is called a *group code.* Actually Q is the field $\mathbb{F}_q$ where $q = p^r$ with p being a prime number. As before we will now again have that Q^n is an n-dimensional vector space, equal to $\mathbb{F}_q^n$. Now I am already prepared to define what is meant by a linear code.

Definition 1.7 (Linear Code) *A* q-ary linear code C *is a linear subspace of* $\mathbb{F}_q^n$*. if* C *has dimension* k *then it is called an* $[n, k]$ *code.*

Now such codes that also has minimum distance d will be called $[n, k, d]$-codes, so this notation means that an $[n, k, d]$-code will be a k-dimensional linear code of length n with minimum distance d. Linear codes also have something called a generator matrix, the definition of which is below.

Definition 1.8 (Generator Matrix) *A* generator matrix G *for a linear code* C *is a* k *by* n *matrix in which the rows are a basis of* C.

Another way to say the same thing is to say that G is a $k \times n$ matrix for which it holds that C is spanned by its rows or that C is simply equal to its row space. That is to say that if G is a generator matrix for C then $C = \left\{ \mathbf{a}G \mid \mathbf{a} \in Q^k \right\}$. Let I_k denote the k by k identity matrix, then when $G = \begin{pmatrix} I_k & P \end{pmatrix}$ it is said to be in *standard form* or *reduced echelon form.* Whenever G is in standard form then the first k symbols of a codeword are called *information symbols.* As the name implies these symbols carry information and the remaining symbols will then serve as *parity check symbols* and are determined by the information symbols. This of course means that the code has $n - k$ parity check symbols. This gives an information rate of $\frac{k}{n}$, since k out of n symbols actually carry information.

Suppose you have two codes C_1 and C_2. If there exists a fixed permutation that when applied to all the codewords of C_1 gives C_2, then the two codes are equally good. Such two codes are called *equivalent.* Sometimes this notion of equivalence is extended to also include a permutation on the symbols of Q. Since Gauss-Jordan elimination is a thing, it is a well known fact from linear algebra that any code will be equivalent to another code that has a generator matrix in standard form.

Now, one should be able to easily separate the information symbols and the redundant symbols. Such codes that have this property are called *separable.* In general if a code C has k information symbols and if $|C| = q^k$ and there is exactly one codeword for every possible choice of entries in the k symbols then the code is said to be *systematic* on k positions. So an $[n, k]$ code is systematic on at least one k-tuple of positions.

Since linear codes are just a special case of block codes, a linear code C has minimum distance $d = 2e + 1$, so it is able to correct up to e errors in a received word. If however $d = 2e$ then an error pattern of weight e will always be detected. Then in the more general case one can say that, if C has M words, then one must check a total of $\binom{M}{2}$ pairs of codewords in order to find d. The following result shows that the work is a bit easier for linear codes.

Theorem 1.9 *For a linear code C the minimum distance is equal to the minimum weight.*

Proof

For a linear code it holds that

$$\begin{aligned} d(\mathbf{x}, \mathbf{y}) &= d(\mathbf{x} - \mathbf{y}, \mathbf{0}) \\ &= w(\mathbf{x} - \mathbf{y}) \end{aligned}$$

and if

$$\begin{aligned} x &\in C, \\ y &\in C \end{aligned}$$

then

$$\mathbf{x} - \mathbf{y} \in C. \qquad \blacksquare$$

Let $\langle \bullet, \bullet \rangle$ denote the inner product between two vectors and then the definition of a so-called dual code can be introduced.

Definition 1.10 (Dual Code) *If C is an $[n, k]$ code then the* dual code $C^\perp$ *is defined as*

$$C^\perp = \left\{ \mathbf{y} \in \mathbb{F}_q^n \,\middle|\, \forall x \in C \text{ then } \langle \mathbf{x}, \mathbf{y} \rangle = 0 \right\}.$$

This dual code $C^\perp$ is also in and of itself a linear code, just as our C is — it is an $[n, n-k]$ code. One might easily think that it is an orthogonal complement to C, but this is not the case. Since the codes are defined over a finite field Q, the subspaces of C and $C^\perp$ could have an intersection larger than $\{\mathbf{0}\}$ and they could also be equal to each other. In the case that $C = C^\perp$ then C is called a *self-dual* code.

Let $\bullet^T$ denote the transpose of a matrix and let $G = \begin{pmatrix} I_k & P \end{pmatrix}$ be the generator matrix for C in the standard form once again, then $H = \begin{pmatrix} -P^T & I_{n-k} \end{pmatrix}$ is the generator matrix for the dual code $C^\perp$. This is because H is of just the right size and rank and has the right entries to fulfill that $G^{-1} = H^T$ which gives that $GH^T = 0$, which is all that is required. This also means that every codeword $\mathbf{a}G$ has inner product 0 with every row of H. Put in another way:

$$\mathbf{x} \in C \Leftrightarrow \mathbf{x}H^T = 0. \tag{1.11}$$

All of the $n-k$ linear equations in eq. (1.11) must be satisfied by all possible codewords in C.

If $\mathbf{y} \in C^\perp$ then the equation $\langle \mathbf{x}, \mathbf{y} \rangle = 0$ (which should hold for every $\mathbf{x} \in C$) found in definition 1.10 is called a *parity check (equation). H* is then called the *parity check matrix* of C. It should be clear now, that the code generated by H is the dual code of C, which also becomes apparent after noting that H^T is the right kernel of C.

Definition 1.12 (Syndrome Of A Linear Code) *If C is a linear code with parity check matrix H then for every $\mathbf{x} \in Q^n$, $\mathbf{x}H^T$ is called the* syndrome *of* $\mathbf{x}$.

Please note that the covering radius $\rho(C)$ of an $[n,k]$ code (see definition 1.4) is the smallest integer such that any (column-)vector in Q^{n-k} can be written as the sum of at most ρ columns of H.

It should be very clear by now that any codeword $\mathbf{x} \in C$ is characterised by syndrome $\mathbf{0}$, since this just follows from definition.

The importance of the syndrome comes from the aid it provides in decoding a received $\mathbf{x}$. Since C is a subgroup of Q^n then Q^n can be partitioned into cosets of C. If two vectors $\mathbf{x}$ and $\mathbf{y}$ have the same syndrome, then they must both be elements of the same coset. Likewise, if the two vectors are part of the same coset, then they must both have the same syndrome. In total one can then write

$$\mathbf{x}H^T = \mathbf{y}H^T \Leftrightarrow \mathbf{x}, \mathbf{y} \in C.$$

So if a vector $\mathbf{x}$ is received that has error pattern $\mathbf{e}$, then $\mathbf{x}$ and $\mathbf{e}$ will have the same syndrome, since the errors in $\mathbf{e}$ are in the same positions as they are in $\mathbf{x}$. This also means that in order to achieve a maximum likelihood decoding of $\mathbf{x}$ one must first find and choose a vector $\mathbf{e}$ in the same coset as $\mathbf{x}$ and then decode $\mathbf{x}$ as $\mathbf{x} - \mathbf{e}$. In this case the vector $\mathbf{e}$ is called the *coset leader.*

This is where the great benefit of introducing an algebraic structure to codes first appears. For an $[n,k]$ code over $\mathbb{F}_q$ there are a total of q^k codewords and q^n possible received messages. Assuming that the information rate is reasonably high, then the receiver only needs to know the q^{n-k} possible coset leaders corresponding to all possible syndromes and since the information rate is high, then q^{n-k} is much smaller than q^n. If the code had no algebraic structure, then one would have to list the most likely transmitted word for every possible received word $\mathbf{x}$, so this is a huge improvement!

Now consider a code C that has minimum distance $d = 2e + 1$. In this case every error pattern of weight $\leq e$ must be the unique coset leader of some coset because two vectors both with weight $\leq e$ must have distance $\leq 2e$ and are therefore elements of different cosets. Now, if C is a perfect code, then there simply are not any other coset leaders. In the case that C has minimum distance $2e + 1$ and all coset leaders have weight $\leq e + 1$, then C is called *quasi-perfect.* The covering radius will be the weight of a coset leader with maximum weight.

Sometimes it might be useful to actually add an extra symbol to every codeword in C according to some natural rule. The following definition gives a common way of doing so.

Definition 1.13 (Extended Code) *Let C be a code of length n over the alphabet $\mathbb{F}_q$. The* extended code *$\overline{C}$ of C is then*

$$\overline{C} = \left\{ (c_1, c_2, \cdots, c_n, c_{n+1}) \;\middle|\; (c_1, \cdots, c_n) \in C, \; \sum_{i=1}^{n+1} c_i = 0 \right\}.$$

Now if C is a linear code with generator matrix G and parity check matrix H, then $\overline{C}$ will also have a generator matrix $\overline{G}$ and a parity check matrix $\overline{H}$. $\overline{G}$ is simply made from G by adding a column such that all the columns columns of $\overline{G}$ adds up to 0 and $\overline{H}$ is produced from H as the following matrix:

$$\overline{H} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ & & & & 0 \\ & & H & & 0 \\ & & & & \vdots \\ & & & & 0 \end{pmatrix}.$$

Lastly let me note that if C is a binary code with an odd minimum distance d, then $\overline{C}$ has minimum distance $d + 1$, since all weights and distances for $\overline{C}$ are even.

Chapter 2

Goppa Codes

Goppa codes were first introduced by V. D. Goppa in 1970 [20]. In this chapter I will start off by defining Goppa codes and then I will show how to build an irreducible binary Goppa code. These are of a particular importance, since they are the type of Goppa codes used in chapter 3 and they have some nice properties that are good to have when it comes to cryptography. I will show how to build these irreducible binary Goppa codes before moving my focus to the minimum distance of these and then how to decode them.

As will be revealed the lower bound is easy to calculate for irreducible binary Goppa codes, they also allow efficient error correction, provided one knows the generating polynomial and lastly if one does not know the generating polynomial, then there is yet to be discovered an efficient algorithm that can do the error correction [17].

This is also the chapter where the importance of chapter 1 is first seen, as Goppa codes are a special case of linear codes [17][31, p. 140]. For a full introduction to coding theory and Goppa codes I would have to recommend a textbook on the subject though.

2.1 Definitions

In this section I will introduce Goppa codes in a general sense and then continue on with how to build an irreducible binary Goppa code including the parity check matrix and the generator matrix for such a code.

2.1.1 Goppa Codes in General

Let $g(z)$ be a monic polynomial of degree t over the field $\mathbb{F}_{q^m}$ (with m and t both being positive integers) and let $L = \{\gamma_0, \gamma_1, \cdots, \gamma_{n-1}\} \subset \mathbb{F}_{q^m}$ such that $|L| = n$ and $g(\gamma_i) \neq 0$ for $0 \leq i \leq n-1$. Now call g the *Goppa polynomial* and in order to emphasise the importance of L, call the elements of L the *code support*[17, 31].

Now the definition of a Goppa code (found in [31, p. 140]) is ready to be introduced.

Definition 2.1 (Goppa Code) *A* Goppa code $\Gamma(L, g)$ *with Goppa polynomial* g *and code support* L *is the set of codewords* $\mathbf{c} = (c_0, c_1, \cdots, c_{n-1})$ *over the alphabet* $\mathbb{F}_q$ *for which the following holds:*

$$\sum_{i=0}^{n-1} \frac{c_1}{z - \gamma_i} \equiv 0 \pmod{g(z)}. \tag{2.2}$$

Now remember how linear codes had something called a syndrome and how Goppa codes are linear codes. Well Goppa codes must also have a syndrome then. Actually equation eq. (2.2) hints at the existence and the definition of the syndrome. The following definition of the syndrome is found in [17].

Definition 2.3 (Syndrome of a Goppa code) *Let* $\mathbf{c} = (\mathbf{c}_0, \cdots, \mathbf{c}_{n-1}) \in \mathbb{F}_q^n$*. Then the* syndrome of a Goppa code*,* $S_\mathbf{c}$*, is given by*

$$S_\mathbf{c}(z) = -\sum_{i=0}^{n-1} \frac{\mathbf{c}_i}{g(\gamma_i)} \frac{g(z) - g(\gamma_i)}{z - \gamma_i} \pmod{g(z)}.$$

The important thing here is that this can lead to another way of actually defining Goppa codes as the set of all vectors $\mathbf{c}$ that fulfills that

$$S_\mathbf{c}(z) = 0$$

or equivalently

$$S_\mathbf{c}(z) \equiv \sum_{i=0}^{n-1} \frac{\mathbf{c}_i}{z - \gamma_i} \equiv 0 \pmod{g(z)}. \tag{2.4}$$

2.1.2 How To Build An Irreducible Binary Goppa Code

Now, since I am dealing with binary Goppa codes, a Goppa polynomial over the field $\mathbb{F}_{2^m}$ is needed, so let $\mathbb{F}_{2^m}[X]$ denote the polynomial ring over this field. In this subsection, I will follow [17], but anywhere that I do not, I will point it out. The purpose of this is to define irreducible binary Goppa codes. The definition should be reminiscent of the previous general definition with some changes and specifications that are of utmost importance for further analysis.

First of all one starts with specifying the Goppa polynomial to be used, so let

$$g(X) = \sum_{i=0}^{t} g_i X^i \in \mathbb{F}_{2^m}[X]$$

be a monic polynomial of degree t. This is the Goppa polynomial that will be used to create a *binary Goppa code.* If this polynomial is also irreducible, then one will obtain an *irreducible binary Goppa code.*

Next a code support is needed. Just as before, it is imperative that this consists of elements γ_i such that $g(\gamma_i) \neq 0$ for all $0 \leq i \leq n-1$. So

$$\mathbf{L} = (\gamma_0, \cdots, \gamma_{n-1}) \in \mathbb{F}_{2^m}^n.$$

Now the preliminaries for the definition of an irreducible binary Goppa code is ready to be introduced.

Definition 2.5 (Irreducible Binary Goppa Code) *Let $g(X) \in \mathbb{F}_{2^m}[X]$ be an irreducible binary polynomial and let $\mathbf{L} \in \mathbb{F}_{2^m}^n$ be a code support for the polynomial g. An* irreducible binary Goppa code *is then the set of all vectors $\mathbf{c} = (\mathbf{c}_0, \cdots, \mathbf{c}_{n-1}) \in \mathbb{F}_2^n$ such that*

$$S_{\mathbf{c}}(X) = 0$$

holds in the polynomial ring $\mathbb{F}_{2^m}[X]$ or equivalently fulfills that

$$S_{\mathbf{c}}(X) \equiv \sum_{i=0}^{n-1} \frac{\mathbf{c}_i}{X - \gamma_i} \equiv 0 \pmod{g(X)}.$$

Let $\mathcal{G}$ denote such an irreducible binary Goppa code and thus it is obtained that

$$\begin{aligned}\mathcal{G}(\mathbf{L}, g(X)) &= \left\{\mathbf{c} \in \mathbb{F}_2^n \mid S_{\mathbf{c}}(X) = 0\right\} \\ &= \left\{\mathbf{c} \in \mathbb{F}_2^n \mid S_{\mathbf{c}}(X) \equiv 0 \pmod{g(X)}\right\}.\end{aligned}$$

An important note when it comes to these types of codes is that if $g(X)$ is indeed irreducible and has degree $t > 1$, then $g(\gamma) \neq 0$ for all $\gamma \in \mathbb{F}_{2^m}$ [27, p. 162]. This means that $\mathbf{L}$ may contain all elements of $\mathbb{F}_{2^m}$.

Next up in order to have a useful code, one must also have a parity check matrix, H, for it. It must be required that any codeword $\mathbf{c} \in \mathcal{G}(\mathbf{L}, g(X))$ fulfills that

$$\mathbf{c} \in \mathcal{G}(\mathbf{L}, g(X)) \Leftrightarrow H\mathbf{c}^T = \mathbf{0}. \tag{2.6}$$

So start off by noticing that

$$\frac{g(X) - g(\gamma_i)}{X - \gamma_i} = \sum_{j=0}^{t} g_j \frac{X^i - \gamma_i^j}{X - \gamma_i} = \sum_{s=0}^{t-1} X^s \sum_{j=s+1}^{t} g_j \gamma_i^{j-1-s}$$

for all $0 \leq i \leq n-1$. It follows that

$$\sum_{i=0}^{n-1} \left(\frac{1}{g(\gamma_i)} \sum_{j=s+1}^{t} g_j \gamma_i^{j-1-s} \right) \mathbf{c}_i = 0.$$

Van Lint points out that this means that the vector

$$\left(\frac{1}{g(\gamma_0)} \cdot \frac{g(X) - g(\gamma_0)}{X - \gamma_0}, \cdots, \frac{1}{g(\gamma_{n-1})} \cdot \frac{g(X) - g(\gamma_{n-1})}{X - \gamma_{n-1}} \right)$$

with each entry interpreted as a column vector is a parity check matrix in a way [31, p. 140] (his argument actually follows from a discussion of BCH codes, but that is way beyond the scope of this thesis). Note that since the entries of H are elements of the extensions field $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$. If one interprets $\mathbb{F}_{2^m}$ as a vector space over $\mathbb{F}_2$ with m as dimension, then h can be written as a matrix over $\mathbb{F}_2$ of dimension $mt \times n$. This leads to the following parity check matrix.

$$H = \begin{pmatrix} g_t g(\gamma_0)^{-1} & \cdots & g_t g(\gamma_{n-1})^{-1} \\ (g_{t-1} + g_t \gamma_0) g(\gamma_0)^{-1} & \cdots & (g_{t-1} + g_t \gamma_{n-1}) g(\gamma_{n-1})^{-1} \\ \vdots & \ddots & \vdots \\ \left(\sum_{j=1}^{t} g_j \gamma_0^{j-1} \right) g(\gamma_0)^{-1} & \cdots & \left(\sum_{j=1}^{t} g_j \gamma_{n-1}^{j-1} \right) g(\gamma_{n-1})^{-1} \end{pmatrix}.$$

In order to emphasise the way in which H fulfills eq. (2.6) one can then write out $H = XYZ$ with

$$X = \begin{pmatrix} g_t & 0 & 0 & \cdots & 0 \\ g_{t-1} & g_t & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_1 & g_2 & g_3 & \cdots & g_t \end{pmatrix}, \; Y = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \gamma_0 & \gamma_1 & \cdots & \gamma_{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_0^{t-1} & \gamma_1^{t-1} & \cdots & \gamma_{n-1}^{t-1} \end{pmatrix} \text{ and}$$

$$Z = \begin{pmatrix} \frac{1}{g(\gamma_0)} & & & \\ & \frac{1}{g(\gamma_1)} & & \\ & & \ddots & \\ & & & \frac{1}{g(\gamma_{n-1})} \end{pmatrix}.$$

Notice that it follows from 2.6 that the row space of H must be dual to the Goppa code. Call this row space V with basis a. Call the dual of this vector space V^* and compute a basis for this called b. The basis vectors in b is then equal to the rows of the generator matrix G. This is also easy to see from how I in section 1.3 wrote that for linear codes it must be true that $GH^T = 0$ implying that $G = \left(H^T\right)^{-1}$ (which should give a rather easy way of actually making G from H or vice versa).

Since H is an $mt \times n$ matrix, the matrix G will have dimension $n \times k$ with $k \geq n - mt$ again being consistent with the fact that it is an $[n, k]$ linear code – more specifically G generates an irreducible binary $[n, k]$ Goppa code with Generator matrix G, parity check matrix H and with

$$k \geq n - mt. \tag{2.7}$$

2.2 Minimum Distance In Irreducible Binary Goppa Codes

In this section I will follow [17] in trying to find the minimum distance for an irreducible binary Goppa code.

Let $\mathcal{G}(\mathbf{L}, g(X))$ be an irreducible binary Goppa code. Again $g \in \mathbb{F}_{2^m}$ is the irreducible binary Goppa polynomial used and the used code support is $\mathbf{L} = \{\gamma_0, \cdots, \gamma_{n-1}\} \in \mathbb{F}_{2^m}^n$ such that $g(\gamma_i) \neq 0$ and $\mathcal{G}(\mathbf{L}, g(X))$ consists of codewords $\mathbf{c} = (\mathbf{c}_0, \cdots, \mathbf{c}_{n-1}) \in \mathbb{F}_2^n$.

Define $\mathcal{T}_\mathbf{c} = \{i \;:\; \mathbf{c}_i = 1\}$ and

$$\sigma_\mathbf{c}(X) = \prod_{j \in \mathcal{T}_\mathbf{c}} \left(X - \gamma_j\right) \in \mathbb{F}_{2^m}[X].$$

Let $\bullet \setminus \bullet$ denote the set minus operator. Now the differential coefficient of $\sigma_\mathbf{c}(X)$ will then be

$$\sigma_\mathbf{c}'(X) = \sum_{i \in \mathcal{T}_\mathbf{c}} \prod_{j \in \mathcal{T}_\mathbf{c} \setminus \{i\}} \left(X - \gamma_j\right).$$

Using eq. (2.4) one then gets that

$$\sigma_\mathbf{c}(X) S_\mathbf{c}(X) \equiv \sigma_\mathbf{c}'(X) \pmod{g(X)}.$$

From here it can be seen that $\sigma_\mathbf{c}$ has roots in any γ_i for all $0 \leq i \leq n-1$ and since $g(\gamma_i) \neq 0$ (again for all $0 \leq i \leq n-1$) it follows that the two polynomials $g(X)$ and

$\sigma_{\mathbf{c}}(X)$ has no common roots and thus must be relatively prime. So $\sigma_{\mathbf{c}}(X)$ must be invertible modulo $g(X)$ with inverse $\sigma_{\mathbf{c}}^{-1}(X)$ and then

$$\sigma_{\mathbf{c}}'(X)\,\sigma_{\mathbf{c}}^{-1}(X) \equiv S_{\mathbf{c}}(X) \pmod{g(X)}. \tag{2.8}$$

It then follows that

$$\forall \mathbf{c} \in \mathbb{F}_2^n \;:\; \mathbf{c} \in \mathcal{G}(\mathbf{L}, g(X)) \Leftrightarrow \sigma_{\mathbf{c}}'(X) \equiv 0 \pmod{g(X)}.$$

Note that $\mathbb{F}_{2^m}$ has the property that $i\sigma_i X^{i-1} = 0$ for every even i and thus the polynomial $\sigma_{\mathbf{c}}'(X) = \sum_{i=1}^{n} i\sigma_i X^{i-1}$ must be a perfect square. Now, $g(X)$ being irreducible also means that

$$\forall \mathbf{c} \in \mathbb{F}_2^n \;:\; \mathbf{c} \in \mathcal{G}(\mathbf{L}, g(X)) \Leftrightarrow \sigma_{\mathbf{c}}'(X) \equiv 0 \pmod{g^2(X)}.$$

Let $\deg(\bullet)$ denote the function that returns the degree of a given polynomial. It follows that for any codeword $\mathbf{c} \in \mathcal{G}(\mathbf{L}, g(X)) \setminus \{\mathbf{0}\}$, the weight of the codeword can be described by

$$w(\mathbf{c}) = \deg(\sigma_{\mathbf{c}}(X)) \geq 1 + \deg\left(\sigma_{\mathbf{c}}'(X)\right) \geq 2\deg(g(X)) + 1.$$

This proves the following result.

Theorem 2.9 (Minimum distance of of an irreducible binary Goppa code) *Let $\mathcal{G}(\mathbf{L}, g(X))$ be an irreducible binary Goppa code with g being an irreducible binary Goppa polynomial of degree t. Then the* minimum distance *of this Goppa code is $2t + 1$.*

This means that any irreducible binary Goppa code is an e-error-correcting code with $e = t = \deg(g)$. In section 2.3 I will go over how to find the errors and how to correct them.

2.3 Decoding Irreducible Binary Goppa Codes

Let me start this section off with refreshing the reader's memory on some important aspects of abstract algebra and then move on to how irreducible binary Goppa codes can be decoded.

2.3.1 Some Preliminary Observations

First off comes two well-known results from abstract algebra (the second one might better be known as a special case of Sylow's theorems). For proofs that I have left out and relevant definitions I will refer the reader to [27]. Let G and H be groups with H being a subgroup of G and denote by G/H the *set of left cosets* of H.

Theorem 2.10 (Lagrange) *If $G \subseteq G$ is a subgroup of a finite group G, then*

$$|G| = |G/H|\,|H|.$$

Theorem 2.10 implies that the order of a subgroup divides the order of the group.

Theorem 2.11 (Cauchy) *Let G be a finite group and p be a prime. If p divides the order of G, then G has an element of order p.*

Now, Let $\mathbb{K}$ be a field with multiplicative identity element $1_{\mathbb{K}}$ and additive identity element $0_{\mathbb{K}}$. Then the characteristic of the field $\mathbb{K}$ is defined in the following way.

Definition 2.12 (Field Characteristic) *The* field characteristic *of a field $\mathbb{K}$, denoted $char(\mathbb{K})$, is the smallest number p such that $p \cdot 1_{\mathbb{K}} = 0_{\mathbb{K}}$, where*

$$p \cdot 1_{\mathbb{K}} = \underbrace{1_{\mathbb{K}} + \cdots + 1_{\mathbb{K}}}_{p \text{ times}} = 0_{\mathbb{K}}.$$

If there exists no such p, then $char(\mathbb{K}) = 0$.

Since $\mathbb{F}_{2^m}$ is a finite field it cannot have characteristic 0 (because all natural numbers are a product of prime numbers). This means that $char(\mathbb{F}_{2^m}) > 0$ and then $1_{\mathbb{F}_{2^m}}$ will generate an additive subgroup of some order p. Theorem 2.10 states that this p will divide the order of the whole group $\mathbb{F}_{2^m}$, which is 2^m. Now the job is to find p. Notice that 2^m is a multiple of the prime number 2. Theorem 2.11 then implies that the additive subgroup will have an element of order 2. Let $a \in \mathbb{F}_{2^m}$ be such an element then this means that

$$a + a = a\left(1_{\mathbb{F}_{2^m}} + 1_{\mathbb{F}_{2^m}}\right) = 0_{\mathbb{F}_{2^m}} \Rightarrow 1_{\mathbb{F}_{2^m}} + 1_{\mathbb{F}_{2^m}} = 2_{\mathbb{F}_{2^m}} = 0_{\mathbb{F}_{2^m}} \tag{2.13}$$

and thus $char(\mathbb{F}_{2^m}) = 2$.

Now I am ready to introduce the Frobenius automorphism.

Definition 2.14 (Frobenius Automorphism) *Let $\mathbb{F}$ be a field of field characteristic p and let $\alpha \in \mathbb{F}$. The* Frobenius automorphism *on $\mathbb{F}$ is the map*

$$\begin{aligned} \phi : \mathbb{F} &\to \mathbb{F}, \\ \alpha &\mapsto \alpha^p. \end{aligned}$$

Remember that an automorphism is an isomorphism from an object to itself whilst preserving the structure of said object. The following corollary is then achieved.

Corollary 2.15 *If $x \in \mathbb{F}_{2^m}$, then the map*

$$\begin{aligned} \mathbb{F}_{2^m} &\to \mathbb{F}_{2^m} \\ x &\mapsto x^2 \end{aligned}$$

is the Frobenius automorphism on $\mathbb{F}_{2^m}$ and therefore every element $y \in \mathbb{F}_{2^m}$ has a unique square root.

Now the Frobenius map

$$\begin{aligned} \mathbb{F}_{2^m}[X] &\to \mathbb{F}_{2^m}[X] \\ f(X) = \sum_{i=0}^{n} f_i X^i &\mapsto (f(X))^2 = \sum_{i=0}^{n} f_i^2 X^{2i} \end{aligned}$$

is an injective ring homomorphism. Its image $\mathbb{F}_{2^m}\left[X^2\right]$ is a set of polynomials being perfect squares of the ring $\mathbb{F}_{2^m}[X]$ and because this is its image, the map is not surjective.

2.3.2 Showing That A Decoding Algorithm Exists

Let $\mathcal{G}(\mathbf{L}, g(X))$ be an irreducible binary Goppa code with code support $\mathbf{L} = \{\gamma_0, \cdots, \gamma_{n-1}\}$ and irreducible binary Goppa polynomial $g \in \mathbb{F}_{2^m}[X]$ of degree t. Suppose you receive a message $\mathbf{m} = (\mathbf{m}_0, \mathbf{m}_1, \cdots, \mathbf{m}_{n-1}) \in \mathbb{F}_2^n$ that is supposed to be a codeword $\mathbf{c} = (\mathbf{c}_0, \mathbf{c}_1, \cdots, \mathbf{c}_{n-1}) \in \mathcal{G}(\mathbf{L}, g(X))$, but it is sent over a noisy channel that introduced errors. Let $\mathbf{e} = (\mathbf{e}_0, \mathbf{e}_1, \cdots, \mathbf{e}_{n-1}) \in \mathbb{F}_2^n$ be an error vector with $w(\mathbf{e}) \leq t$ and let $\bullet \oplus \bullet$ be the binary bitwise XOR operation. Then

$$\mathbf{m} = \mathbf{c} \oplus \mathbf{e}.$$

One might also write

$$\mathbf{e} = \mathbf{m} - \mathbf{c}.$$

The problem is now to get back the codeword $\mathbf{c}$ knowing only $\mathbf{m}$.

Start off by computing the syndrome $S_{\mathbf{m}}(X)$ but notice first that $S_{\mathbf{c}}(X) \equiv 0 \pmod{g(X)}$ so that

$$\begin{aligned} S_{\mathbf{m}}(X) &\equiv S_{\mathbf{e}}(X) \\ &\equiv \sum_{i=0}^{n-1} \frac{\mathbf{e}_i}{X - \gamma_i} \pmod{g(X)}. \end{aligned}$$

By definition this can be computed using the parity check matrix H for $\mathcal{G}(\mathbf{L}, g(X))$ or by using just the received message $\mathbf{m}$.

Now let $\mathcal{T}_{\mathbf{e}} = \{i \ : \ \mathbf{e}_i = 1\}$ and define the *error locator polynomial* $\sigma_{\mathbf{e}}(X)$ and its *companion polynomial* $\sigma'_{\mathbf{e}}(X)$ as

$$\begin{aligned} \sigma_{\mathbf{e}}(X) &= \prod_{j \in \mathcal{T}_{\mathbf{e}}} \left(X - \gamma_j\right) \in \mathbb{F}_{2^m}[X], \\ \sigma'_{\mathbf{e}}(X) &= \sum_{i \in \mathcal{T}_{\mathbf{e}}} \prod_{j \in \mathcal{T}_{\mathbf{e}} \setminus \{i\}} \left(X - \gamma_j\right). \end{aligned}$$

Notice how $\sigma_{\mathbf{e}}(X)$ and $\sigma'_{\mathbf{e}}(X)$ have no common factors meaning that they are relatively prime and how $\deg(\sigma_{\mathbf{e}}(X)) \leq t$ whilst $\deg\left(\sigma'_{\mathbf{e}}(X)\right) \leq \deg(\sigma_{\mathbf{e}}(X))$. Observe also that

$$S_{\mathbf{e}}(X)\,\sigma_{\mathbf{e}}(X) \equiv \sigma'_{\mathbf{e}}(X) \pmod{g(X)}. \tag{2.16}$$

Suppose some algorithm could give the monic polynomial $a(X)$ of lowest degree such that $a(X) \neq 0$ and a polynomial $b(X)$ of lower degree such that

$$S_{\mathbf{e}}(X)\,a(X) \equiv b(X) \pmod{g(X)}. \tag{2.17}$$

Given eq. (2.8), eq. (2.17) can be rewritten as

$$\sigma'_{\mathbf{e}}(X)\,\sigma_{\mathbf{e}}^{-1}(X)\,a(X) \equiv b(X) \pmod{g(X)}$$

and multiplying by $\sigma_{\mathbf{e}}(X)$ and rearranging this gives

$$b(X)\,\sigma_{\mathbf{e}}(X) - \sigma'_{\mathbf{e}}(X)\,a(X) \equiv 0 \pmod{g(X)}.$$

Since the degree of the left hand side is less than $\deg(g(X))$, it must be 0. Then since $\sigma_{\mathbf{e}}(X)$ and $\sigma'_{\mathbf{e}}(X)$ are relatively prime $\sigma_{\mathbf{e}}(X)$ must divide $a(X)$, but given that $a(X)$ is of least degree, it follows that $a(X) = \sigma_{\mathbf{e}}(X)$. This means that once $a(X)$ is found, then $\sigma_{\mathbf{e}}(X)$ and then also $\sigma'_{\mathbf{e}}(X)$ can be found. Once these polynomials are known then the positions $0 \leq i \leq n-1$ for which $\mathbf{e}_i \neq 0$ can be extracted and the vector $\mathbf{e}$ then becomes known. The existence of such an algorithm that leads to eq. (2.17) would prove that there is indeed a way to decode irreducible binary Goppa polynomials. As will be revealed shortly there is actually one based on the extended euclidean algorithm for univariate polynomials that will perform exactly the task that we want it to [31, pp. 144-145][17].

2.3.3 The Decoding Algorithm

I will now follow [17] and go through why their algorithm works, finishing up with showing the actual algorithm itself. Then comes a minor discussion about its running time.

First of all compute the syndrome using the parity check matrix as described previously, define the error locator polynomial and arrive at eq. (2.16) as described previously.

Split up $\sigma_{\mathbf{e}}(X)$ into squares and non-squares yielding

$$\sigma_{\mathbf{e}}(X) = \alpha^2(X) + X\beta^2(X). \tag{2.18}$$

In section 2.3.1 I laid out how $char(\mathbb{F}_{2^m}) = 2$ and due to eq. (2.13) I then get that $\sigma'_{\mathbf{e}}(X) = \beta^2(X)$. This means that eq. (2.16) can be rewritten as

$$\beta^2(X)(XS_{\mathbf{e}}(X) + 1) \equiv \alpha^2(X)S_{\mathbf{e}}(X) \pmod{g(X)}. \tag{2.19}$$

Assume that $\mathbf{e}$ does not constitute a codeword in $\mathcal{G}(\mathbf{L}, g(X))$ so $S_{\mathbf{e}}(X) \not\equiv 0 \pmod{g(X)}$. This means that there must exist an inverse of $S_{\mathbf{e}}(X)$ modulo $g(X)$. Set $T(X) = S_{\mathbf{e}}^{-1}(X)$ and multiply this into eq. (2.19) to achieve

$$\beta^2(X)(X + T(X)) \equiv \alpha^2(X) \pmod{g(X)}. \tag{2.20}$$

Corollary 2.15 comes in handy now. Let $\tau(X) \in \mathbb{F}_{2^m}[X]$ be the unique square root of the polynomial $T(X) + X$ such that $\tau(X)\tau(X) \equiv T(X) + X \pmod{g(X)}$. Now take the square root of eq. (2.20) and arrive at

$$\beta(X)\tau(X) \equiv \alpha(X) \pmod{g(X)}. \tag{2.21}$$

All of this has so far just been rewriting what has already been introduced. Note that $\tau(X)$ and $g(X)$ is already known and then the problem is now to find a unique pair $\alpha(X)$ and $\beta(X)$ of least degree so that eq. (2.21) is fulfilled. This is where the extended euclidean algorithm for univariate polynomials comes into play.

Remember that by assumption $\deg(\sigma_{\mathbf{e}}(X)) \leq t$. This means that $\deg(\alpha(X)) \leq \left\lfloor \frac{t}{2} \right\rfloor$ and $\deg(\beta(X)) \leq \left\lfloor \frac{t-1}{2} \right\rfloor$. This also means that the extended euclidean algortihm can actually be used.

The extended euclidean algorithm for univariate polynomials produces polynomials $\alpha_k(X) + \beta_k(x)\tau_k(X) \equiv 0 \pmod{g(X)}$ in each iteration with $\deg(\beta_k(X)) =$

$\deg(g(X)) - \deg(\alpha_{k-1}(X))$. This is of particular importance, because after each iteration the degree of β increases and the degree of α decreases. This means that there comes a point where both polynomials are below the bounds that have just been laid out. So running this algorithm until the first iteration after $\deg(\alpha_k(X)) = \left\lfloor \frac{t+1}{2} \right\rfloor$ yields the polynomial $\alpha_k(X)$ with

$$\deg(\alpha_k(X)) \leq \left\lfloor \frac{t+1}{2} \right\rfloor - 1 \leq \left\lfloor \frac{t}{2} \right\rfloor .$$

In this iteration a $\beta_k(X)$ is also achieved that will have

$$\begin{aligned}\deg(\beta_k(X)) &= \deg(g(X)) - \deg(\alpha_k(X)) \\ &\leq t - \left\lfloor \frac{t+1}{2} \right\rfloor = \left\lfloor \frac{t-1}{2} \right\rfloor .\end{aligned}$$

So now set $\alpha(X) = \alpha_k(X)$ and $\beta(X) = \beta_k(X)$ in eq. (2.21) and the only remaining problem is to find the roots of eq. (2.18), which will lead to the vectors $\mathbf{e}$ and $\mathbf{c}$.

The final and full algorithm due to [17] can be found in algorithm 2.22.

2.3.3.1 Time Complexity Of The Decoding Algorithm

Now for the running time of algorithm 2.22. This will be done according to [17].

Remember how the irreducible binary Goppa code is an $[n, k]$ Goppa code. Well This means that if the $S_{\mathbf{m}}(X)$ is computed using the parity check matrix H then this step in the algorithm takes $(n-k)n$ binary operations. In order to compute $T(X)$ the extended euclidean algorithm is used. Since $g(X)$ is a polynomial of degree t with coefficients of size m this step takes $\mathcal{O}\left(t^2m^2\right)$ operations. The same is true for the later uses of the same algorithm. Computing the square root of $T(X) + X$ is a linear mapping on the ideal $\mathbb{F}_{2^m}/g(X)$ so this step also takes $\mathcal{O}\left(t^2m^2\right)$ operations. The last step of determining the the zeroes of $\sigma_{\mathbf{e}}(X)$ is the hardest though and takes $n\left(tm^2 + tm\right)$ binary operations. Note however that $mt \geq (n-k)$ so the time complexity of the algorithm ends up being

$$\mathcal{O}\left(ntm^2\right).$$

Algorithm 2.22 Decoding Algorithm for An Irreducible Binary Goppa Code

```
**Input:** An irreducible binary Goppa code $\mathcal{G}(\mathbf{L}, g(X))$ and a vector $\mathbf{m} = \mathbf{c} \oplus \mathbf{e}$ with $\mathbf{c}$ being a codeword and $\mathbf{e}$ an error vector.

**Output:** the codeword $\mathbf{c}$ and the error vector $\mathbf{e}$.

$S_{\mathbf{m}}(X) = \sum_{i=0}^{n-1} \frac{\mathbf{m}_i}{X-\gamma_i} \pmod{g(X)}$        ▷ Compute the syndrome of $\mathbf{m}$
(or use the parity check matrix $H$)

if $S_{\mathbf{m}}(X) \equiv 0 \pmod{g(X)}$ then
    Return $(\mathbf{m}, \mathbf{0})$        ▷ There are no errors, $\mathbf{m}$ is a codeword
else
    $T(X) = S_{\mathbf{m}}^{-1}(X) \pmod{g(X)}$        ▷ There are errors, $\mathbf{m}$ is not a codeword
    $\tau(X) \equiv \sqrt{T(X) + X} \pmod{g(X)}$
end if

/*Extended euclidean algorithm*/
$i = 0; r_{-1}(X) = \alpha_{-1}(X) = g(X); r_0(X) = \alpha_0(X) = \tau(X); \beta_{-1}(X) = 0; \beta_0(X) = 1$
while $\deg(r_i(X)) \geq \left\lfloor \frac{t+1}{2} \right\rfloor$ do
    $i = i + 1$
    Determine $q_i(X), r_i(X)$ such that $r_i(X) = r_{i-2}(X) - q_i(X) r_{i-1}(X)$ and
        $\deg(r_i(X)) < \deg(r_{i-1}(X))$
    $\beta_i(X) = \beta_{i-2}(X) + q_i(X) \beta_{i-1}(X)$
    $\alpha_i(X) = r_i(X)$
end while

$\sigma(X) = a^2 \left((\alpha_i(X))^2 + X(\beta_i(X))^2\right)$ with $a \in \mathbb{F}_{2^m}$ such that $\sigma(X)$ is monic

for $i = 0$ to $n - 1$ do        ▷ Determination of zeroes of $\sigma_{\mathbf{e}}(X)$
    if $\sigma(\gamma_i) = 0$ then
        $\mathbf{e}_i = 1$
    else
        $\mathbf{e}_i = 0$
    end if
    $\mathbf{c} = \mathbf{m} \oplus \mathbf{e}$
end for

Return $(\mathbf{c}, \mathbf{e})$
```

Chapter 3

The Original McEliece Cryptosystem

In 1978 R. J. McEliece was the first to propose a public key cryptosystem based on coding theory. This is in contrast to more well known encryption schemes such as RSA, that is based on the integer factorisation problem. More specifically, McEliece proposed using irreducible binary Goppa codes [33]. Given how early this proposal was made, it is thus one of the oldest asymmetric cryptosystems and remains yet to be broken, provided one is careful in choosing the security parameters used. The advantage of code-based cryptography lies in faster en- and decryption, so the question now is; why did it not gain as much traction as RSA or other public key cryptosystems based on number-theoretical problems? Well the answer lies in the big keys used. Nowadays this is not much of an issue though, since storage capacity has increases vastly since 1978 and so has computing power. Thus the McEliece public key cryptosystem has now become quite practical.

Having introduced linear codes in section 1.3 and then also irreducible binary Goppa codes in section 2.1 alongside its decoding in section 2.3 I am now finally ready to introduce the McEliece public key cryptosystem in section 3.2 and the security of it in section 3.3. First however I will have to introduce just a bit more coding theory in section 3.1, so that it becomes apparent where the idea behind the cryptosystem comes from.

3.1 Some Problems From Coding Theory

In this section I will mostly follow [17] and define some problems related to linear codes. This will serve as a preparation for the discussion of the McEliece cryptosystem.

First off a look at a problem that has been researched in the past. It is the problem of decoding a general linear code. But before looking at that, a clarification about notational matters need to be disclosed. So let C be an arbitrary $[n, k]$ linear code over the field $\mathbb{F}$ with $\mathbf{y} \in \mathbb{F}^n$ as defined in chapter 1. By definition this code will have a generator matrix, G, and parity check matrix H. Let also d denote a distance function (as usual I will be using the Hamming distance from definition 1.1 – though other notions of distance do exist) and let w denote the hamming weight function from definition 1.1.

The problem is then as follows.

Problem 3.1 (General Decoding Problem For Linear Codes) *Find* $\mathbf{x} \in C$ *such that* $d(\mathbf{y}, \mathbf{x})$ *is minimal.*

Now let $d_{\min}$ denote the minimum distance of the linear code C and let $\mathbf{e}$ be an error vector of weight $w(\mathbf{e}) \leq t = \left\lfloor \frac{d_{\min}-1}{2} \right\rfloor$. If $\mathbf{x} \in C$ then there must exist a unique solution to the general decoding problem for $\mathbf{y} = \mathbf{x} + \mathbf{e}$. In this case, C will be a t-error-correcting-code.

Now Berlekamp et al. managed to prove that problem 3.1 is actually NP-hard [7], but Vardy later improved upon this leading to the following result.

Theorem 3.2 *Problem 3.1 is NP-complete.*

Proof See [45]. ∎

One could also easily imagine another problem regarding finding codewords in a linear code of a particular weight. This is stated here.

Problem 3.3 (Subspace Weights) *Let* $\mathbf{w} \in \mathbb{N}$. *Find* $\mathbf{x} \in C$ *such that* $d(\mathbf{0}, \mathbf{x}) = \mathbf{w}$.

The hope here is that a cryptosystem can be built upon these problems. For problem 3.1 this is not too hard to imagine, given theorem 3.2. For problem 3.3 it follows from the following result.

Theorem 3.4 *Problem 3.3 is NP-Hard.*

Proof See [7]. ∎

There is just one more problem that might be of interest based upon the equivalence of codes. Such a notion was briefly mentioned in section 1.3 but here I will give a more proper introduction to it, starting with permutation equivalence. Remember that for any $\mathbf{x} \in C$, $\mathbf{x}$ is written as an n-tupel: $\mathbf{x} = (x_0, \cdots, x_{n-1})$ and then a *permutation*, denoted π, of the *permutation group* S_n, will just shuffle around the entries in $\mathbf{x}$ according to some fixed rule. Denote by π^{-1} the inverse permutation.

Definition 3.5 (Permutation Equivalent) *Two* $[n, k]$ *codes,* C *and* C', *over a field,* $\mathbb{F}$, *are called* permutation equivalent *if there exists a permutation,* π, *of the permutation group* S_n *over* n *elements such that*

$$C' = \pi(C) = \left\{ \pi^{-1}(\mathbf{x}) \,\middle|\, \mathbf{x} \in C \right\}.$$

The subgroup of S_n that keeps C fixed is then called $Aut(C)$.

The more formal version of the definition of equivalence is then ready to be introduced. Let $\bullet \circ \bullet$ denote the entry wise vector multiplication, ϕ be a field automorphism of the field $\mathbb{F}$ and φ be the function that applies ϕ to all entries in a vector.

Definition 3.6 (Equivalence) *Two* $[n, k]$ *codes,* C *and* C', *over a field,* $\mathbb{F}$, *are said to be* equivalent *if there exists a permutation,* π, *of the permutation group* S_n *over* n, *an* n*-tupel* $\mathbf{a} = (a_0, \cdots, a_{n-1}) \in \mathbb{F}^n$ *and a field automorphism,* ϕ *of* $\mathbb{F}$ *such that*

$$\mathbf{x} \in C \Leftrightarrow \varphi\left(\pi^{-1}(\mathbf{a}) \circ \pi^{-1}(\mathbf{x})\right) \in C'.$$

Notice how in the case where $\mathbb{F} = \mathbb{F}_2$ the definition of permutation equivalency is the same as the definition of equivalency.

Let the two codes C and C' have generator matrices G and G' respectively. Now the final problem of this section can be introduced.

Problem 3.7 *Given two generator matrices G and G', decide if the codes generated by the matrices are permutation equivalent or not.*

3.2 Description Of The McEliece Cryptosystem

As mentioned earlier in this chapter, the McEliece cryptosystem is based upon problems from coding theory. To be more precise, it is based upon the decoding problem of a large linear code [26]. This is a special case of problem 3.1 and the next section, I will reveal just how this problem is used.

In [33] the cryptosystem is first proposed and I will describe it here. Notice that it uses irreducible binary Goppa codes, but in [17, p. 6] it is pointed out that "any subclass of the class of alternant codes could be used. However, it might not reach the desired security". Goppa codes are a class of alternant codes, but as the quote above states, one still has to be careful when trying to build one for use in the McEliece cryptosystem – in particular one should just stick to the irreducible binary Goppa codes introduced in section 2.1.2. I will get back to this in the next chapter.

Just as with any public key cryptosystem, this cryptosystem utilises a trapdoor function. In this case the trapdoor will be the knowledge of the used Goppa polynomial [17].

Now the cryptosystem can be defined as follows.

System Parameters: $n, t \in \mathbb{N}$ where $t < n$.

Key Generation: Given the parameters n, t generate the following matrices:

G: $k \times n$ generator matrix of an irreducible binary $[n, k]$ Goppa code $\mathcal{G}$ which can correct up to t errors.
S: $k \times k$ random binary non-singular matrix.
P: $n \times n$ random permutation matrix.

Then compute the $k \times n$ matrix $G' = \mathrm{S}G\mathrm{P}$.

Public Key: (G', t).

Private Key: $(\mathrm{S}, D_{\mathcal{G}}, \mathrm{P})$ where $D_{\mathcal{G}}$ is an efficient decoding algorithm for $\mathcal{G}$ (see e.g. algorithm 2.22).

Encryption: To encrypt a plaintext $\mathbf{m} \in \{0, 1\}^k$ choose a vector $\mathbf{z} \in \{0, 1\}^n$ of weight t randomly and compute the ciphertext $\mathbf{c}$ in the following way:

$$\mathbf{c} = \mathbf{m}G' \oplus \mathbf{z}. \tag{3.8}$$

Decryption: To decrypt a ciphertext **c** start by calculating

$$\mathbf{c}\mathrm{P}^{-1} = \mathbf{m}\mathrm{S}G \oplus \mathbf{z}\mathrm{P}^{-1}.$$

Then apply the decoding algorithm $D_{\mathcal{G}}$ to this. Because $\mathbf{c}\mathrm{P}^{-1}$ has a hamming distance of t to the Goppa code, the codeword obtained is

$$\mathbf{m}\mathrm{S}G = D_{\mathcal{G}}\left(\mathbf{c}\mathrm{P}^{-1}\right).$$

Now one can now compute the plaintext **m** as

$$\mathbf{m} = (\mathbf{m}\mathrm{S}G)\, G^{-1}\mathrm{S}^{-1}.$$

3.3 Security- And System Parameters Of The Original McEliece Cryptosystem

In this section I will first introduce a basic concept of information theory in section 3.3.1 and continue on with discussing how this can be used to let the security of the McEliece cryptosystem be expressed by a single system parameter in section 3.3.2.

3.3.1 Shannon Entropy

Here I will just quickly define something called entropy, that Shannon introduced in 1948 [39]. Let $[\bullet, \bullet]$ denote a *closed interval.* Let also A be a discrete random variable, that assumes values in the alphabet $\mathcal{A}$ and is distributed according to some probability distribution $P : \mathcal{A} \to [0, 1]$.

Definition 3.9 (Shannon Entropy) *The* entropy *(or* Shannon entropy*) of a variable is given by*

$$H(A) = \sum_{a \in \mathcal{A}} P(a) \log\left(\frac{1}{p(x)}\right).$$

When dealing with bit strings, one would use base 2 for the logarithm in definition 3.9. Note that the entropy can be interpreted as the level of uncertainty as to the possible outcome of the variable A (this is also where the idea and name comes from – it is simply borrowed from statistical physics).

3.3.2 Security In Terms Of The System Parameters

A question that might arise is, why would one possibly consider the cryptosystem just described in section 3.2 as secure? Well, have a look at the following problem.

Problem 3.10 (McEliece Problem) *Suppose you have a McEliece public key* $\left(G', t\right)$ *where* $G' \in \{0,1\}^{k \times n}$ *and a ciphertext* $\mathbf{c} \in \{0,1\}^n$*. Now find the unique message* $\mathbf{m} \in \{0,1\}^k$ *such that* $d\left(\mathbf{m}G', \mathbf{c}\right) = t$.

Clearly if one is able to solve problem 3.1 then one is also able to solve problem 3.10. The reverse cannot be said though. Solving problem 3.10 would not solve problem 3.1 except for in a certain class of codes. Notice that the matrix S disguises the underlying Goppa code as a general linear code. As is usual, one can conjecture the decoding problem for large linear codes to be a hard problem to solve. This means that problem 3.10 can also be conjectured to be hard [17, 26, 34]. Thus the following is obtained

Conjecture 3.11 *Problem 3.10 is hard to solve.*

What exactly is meant by problem 3.10 being hard to solve then? Well it means that any probabilistic adversary trying to break the cryptosystem by solving problem 3.10 in polynomial time will do so with only negligible probability in the security parameter(s). Remember that the used irreducible binary Goppa polynomial g is over the field $\mathbb{F}_{2^m}$. From eq. (2.7) it follows that

$$k = n - mt + a \tag{3.12}$$

where $a \in \mathbb{N} \cup \{0\}$. Intuitively in order to achieve maximum entropy on the encrypted message, it will be required that all bits of $\mathbf{m}G'$ has an equal chance of getting flipped or not flipped. This means that the vector $\mathbf{z}$ should have Hamming weight $\frac{n}{2}$, but since $w(\mathbf{z}) = t$ this means that $t = \left\lfloor \frac{n}{2} \right\rfloor$ since t must be a natural number. This value for t does however not satisfy $2t + 1 \leq n$, which is needed in order to provide adequate error correction. The easy solution to this is to just let t be as close to this bound as possible, so t is then determined by

$$t = \left\lfloor \frac{n-1}{2} \right\rfloor. \tag{3.13}$$

Now eq. (3.12) becomes

$$k = n - \left\lfloor \frac{mn - m}{2} \right\rfloor + a. \tag{3.14}$$

Now, if $t > 1$ and n is chosen to be maximal, then since $|\mathbb{F}_{2^m}| = 2^m$

$$n = 2^m$$

and so

$$k = 2^m - \left\lfloor \frac{m2^m - m}{2} \right\rfloor + a.$$

This does however mean that m cannot be chosen to be very large. There is still a way out though, because n does not have to be chosen to be maximal and so one still has eq. (3.14). For some fixed m this then implies that n can be used to define a lower limit for k and can thus be used as the sole security parameter of the cryptosystem. The only downside to this is that it limits n from above, but as was just seen, this is somewhat desirable.

So what is now meant by the probability of a successful attack being negligible in n? Well it means that the probability of a successful attack is inversely proportional to some function in n that grows faster than any polynomial in n (a polynomial because it is assumed that the adversary works in polynomial time). This in turn implies that as n grows larger, the probability of a successful attack asymptotically approaches 0 faster than the multiplicative inverse of any polynomial in n. Allow me to formally introduce this notion of a negligible probability, since it will also be useful in later discussions.

Definition 3.15 (Negligible Probability) *Let $y \in \mathbb{N}$ be a natural number, $\epsilon(y)$ be a probability expressed in terms of the parameter y and $p(y)$ be any polynomial in the same parameter y. If*

$$\epsilon(y) \leq \frac{1}{p(y)}$$

for all large enough y, then the probability $\epsilon(y)$ is said to be negligible in y.

This in turn leads to another more formalised version of conjecture 3.11.

Conjecture 3.16 *Any probabilistic polynomial time adversary that tries to solve problem 3.10 will do so with only negligible probability in n.*

Notice that t can be chosen to not satisfy eq. (3.13) $\left(\text{being chosen to be below } \left\lfloor \frac{n-1}{2} \right\rfloor\right)$ and now the security will have to be stated in terms of not only n, but also k and t.

Chapter 4

The Need For Better Security

In this chapter I will outline some security concerns regarding the original McEliece public key cryptosystem that was introduced in section 3.2.

The main takeaway of section 4.1 is that it is indeed imperative, that the Goppa codes used will be an irreducible binary Goppa code built in exactly the same way as described in section 2.1.2.

In section 4.2 I will give a rundown of some attacks on the ciphertext produced by the original McEliece public key cryptosystem. Some of those attacks are to be considered critical and thus it will automatically be revealed that this ultimately leads to a need for a better and more rigorous security. This security notion that is needed will also be briefly introduced in this section.

4.1 Attacking The Structure Of The McEliece Cryptosystem

I mentioned in section 3.2 how in [17] it is pointed out that any Goppa code can be used for the original McEliece public key cryptosystem. It is indeed true that any one of these work, but I also noted, that the security might be compromised when one is not using an irreducible binary Goppa code. In this section I will reveal just how one might accidentally compromise the security of the cryptosystem.

Let me just start this section off with something completely different in section 4.1.1. This will serve as preliminaries in order to understand the further goal of this section in section 4.1.2. Since the attack in section 4.1.2 can be overcome, I will not go into full detail about it. I will instead keep to describing it in essence.

4.1.1 The Importance Of $\mathbf{L}$, P and S

Here I will follow [17] in trying to explain the significance of $\mathbf{L}$, P and S. In doing so it will automatically also be explained why the private key is what it is (that is, why $D_{\mathcal{G}}$ and P must be kept secret). As a shorthand, I will write "PKC" instead of "public key cryptosystem".

For the McEliece PKC one needs to use an irreducible binary Goppa code $\mathcal{G}(\mathbf{L}, g(X))$ with code support $\mathbf{L}$ and Goppa polynomial g. Suppose a public key of the McEliece PKC is built alongside a corresponding private key. Since P is a random permutation, then even if the public key is known to an adversary, then **L** can still be revealed without introducing any security problems, provided that P is indeed kept secret (it is actually often the case that **L** is revealed for normal applications). The reason for this is that the knowledge of P will be needed for decryption and for any $\gamma \in \mathbf{L}$ it follows by definition that $g(\gamma) \neq 0$, so g can not be reconstructed using only the knowledge of **L** alone. Taken together this means that one would still have to brute force the generator matrix G of $\mathcal{G}$, even if **L** is known. This also serves as an argument as to why g must be kept secret and why it is the trapdoor in the McEliece PKC.

If P is revealed, then so is $G'\mathrm{P}^{-1} = \mathrm{S}G$. Now suppose g is unknown and let H' be the systematic dual matrix of $\mathrm{S}G$. Then from definition 2.5 it follows that $S_{\mathbf{c}}(X) = 0$ for all binary vectors $\mathbf{c}$ where $H'\mathrm{P}^{-1}\mathbf{c}^T = 0$ (remember that the underlying Goppa code is defined as the set of the $\mathbf{c}$'s that fulfill this property). If the underlying Goppa code is revealed, then an adversary would be able to do efficient error correction and so the underlying Goppa code can not be allowed to be compromised. So revealing P would break the McEliece PKC and thus P must be kept secret.

Unlike P, S does not aid in hiding the secret Goppa polynomial g being used. As Canteaut and Chabaud put it, "the invertible matrix S has no cryptographic function; it only assures for McEliece's system that the public matrix is not systematic otherwise most of the bits of the plain-text would be revealed" [12, p. 4]. As of yet there is no known way efficiently to recover H from knowing $\mathrm{S}^{-1}G'$ only. An important note here is that this statement is only made about the original McEliece PKC.

4.1.2 Weak Keys

In 1998, Loidreau and Sendrier proposed a way to identify weak keys [32]. Specifically they found a structural attack on the original McEliece PKC that works, if the used Goppa polynomial is a "binary" polynomial. What is meant by this is that the Goppa polynomial, g, is over the field $\mathbb{F}_2$ – i.e. $g \in \mathbb{F}_2[X]$.

In [38] an algorithm known as the *Support Splitting Algorithm (SSA)* is presented. It can be used to determine whether or not two codes are permutation equivalent and the automorphism group of a code.

Now the idea behind the attack is as follows. Use the SSA as an oracle to determine whether or not the Goppa polynomial used is of the aforementioned form. This greatly reduces the search space of a brute force attack on the private key and so, if the used Goppa polynomial is found to be over the field $\mathbb{F}_2$, then one can simply brute force search for a Goppa polynomial that generates the used Goppa code. Once the underlying Goppa code is known, then P can be revealed and the cryptosystem is then broken.

The easy way to deal with this attack is to just simply not use such weak Goppa codes. The attack technically can be generalised to work if the used Goppa polynomial is over any subfield of $\mathbb{F}_{2^m}$, but in the general case an exhaustive search simply takes too long and if g is not of the weak form previously described, then the amount of polynomials in the equivalence classes found are simply too low to reduce the search space enough to make the attack viable in practice [17, 26].

4.2 Ciphertext Only Attacks On The McEliece Cryptosystem

In the beginning of the chapter I promised, that I would introduce a security notion in this section. Before beginning to talk about non-critical- and critical ciphertext only attacks, I will have to introduce this security notion. This will provide a basis for why the attacks described in this section can be divided into critical- and non-critical attacks.

Suppose an adversary who wants to recover a message from its ciphertext only, has access to a decryption oracle. The adversary may not query the oracle on the target ciphertext, but is given the plaintext that is the result of decrypting any chosen ciphertext. The adversary then hopes that this can result in a plaintext that can be used to recover the target plaintext. This is called an *adaptive chosen ciphertext attack* (*CCA2*). The security definition that is aspired to can then be introduced.

Definition 4.1 (CCA2 Security) *A cryptosystem is said to be* secure against adaptive chosen ciphertext attacks *(*CCA2 secure*) if all probabilistic polynomial time adversaries has only negligible probability of deciphering a given ciphertext, through the use of an adaptive chosen ciphertext attack.*

Usually this would be stated in another (yet more formal) way however. In order to understand this other way of stating the definition, some more introduction to this field of study will be needed. So let $\Pi = (G, E, D)$ be a public key cryptosystem with generating algorithm G, that generates the public key, pk, and secret key, sk, encryption algorithm E and decryption algorithm D. For this to be useful when transferring messages, it is required that such a cryptosystem would fulfill that for any encryption of a plaintext, correct decryption is possible. This means that if x is a plaintext and (pk, sk) is a key pair output by G, then $D_{sk}\left(E_{pk}(x)\right) = x$ (where E_{pk} denotes encryption using the public key and D_{sk} denotes decryption using the secret key).

Now call the adversary A and suppose A is in a case where it communicates with one of two different types of oracles, O_0 and O_1. Let $P(A, 0)$ be the probability that A guesses being in the case, where A is talking to oracle O_0 and let $P(A, 1)$ be the probability that A guesses being in the case, where A is talking to oracle O_1. Let $|\bullet|$ denote the absolute value of a number (please note that this notation is overloaded and looks exactly the same as the notation of the function that returns the number of elements in a set). The advantage of A can then be introduced.

Definition 4.2 (Advantage Of An Adversary) *Let A be an adversary that communicates with an oracle and let O_0 and O_1 be oracles. The* advantage of A, Adv_A, *is then defined as*

$$Adv_A(O_0, O_1) = |P(A, 0) - P(A, 1)|.$$

Let $O^{\text{CCA2}}_{\text{ideal}}$ and $O^{\text{CCA2}}_{\text{real}}$ be oracles and let k be the security parameter of Π. Consider then the following two scenarios:

The Ideal World Both the adversary A and oracle $O^{\text{CCA2}}_{\text{ideal}}$ gets the security parameter k as input.

1. $O_{\text{ideal}}^{\text{CCA2}}$ runs $G(k)$ to get (pk, sk) and gives pk to A.
2. A may submit a string $\mathbf{y}$ to $O_{\text{ideal}}^{\text{CCA2}}$, which will return $D_{sk}(\mathbf{y})$ to A. This step can be repeated as many times as A wants.
3. A then chooses a plaintext $\mathbf{x}$ and gives to an encryption oracle O_E. The oracle responds with $\mathbf{y}_0 = E_{pk}(\mathbf{r})$, where $\mathbf{r}$ is a randomly chosen plaintext of the same length as $\mathbf{x}$.
4. A may now again submit a chosen string $\mathbf{y}$ to $O_{\text{ideal}}^{\text{CCA2}}$, but is not allowed to submit the string $\mathbf{y} = \mathbf{y}_0$. If A follows the rules, then $O_{\text{ideal}}^{\text{CCA2}}$ will return $D_{sk}(\mathbf{y})$ to A. Again this step can be repeated as long as A wants to.
5. A now outputs a bit b, signifying A's guess as to whether or not A is in this scenario.

The Real World Both the adversary A and oracle $O_{\text{real}}^{\text{CCA2}}$ gets the security parameter k as input.

1. $O_{\text{real}}^{\text{CCA2}}$ runs $G(k)$ to get (pk, sk) and gives pk to A.
2. A may submit a string $\mathbf{y}$ to $O_{\text{real}}^{\text{CCA2}}$, which will return $D_{sk}(\mathbf{y})$ to A. This step can be repeated as many times as A wants.
3. A then chooses a plaintext $\mathbf{x}$ and gives it to $O_{\text{real}}^{\text{CCA2}}$. $O_{\text{real}}^{\text{CCA2}}$ responds with $\mathbf{y}_0 = E_{pk}(\mathbf{x})$.
4. A may now again submit a chosen string $\mathbf{y}$ to $O_{\text{real}}^{\text{CCA2}}$, but is not allowed to submit the string $\mathbf{y} = \mathbf{y}_0$. If A follows the rules, then $O_{\text{real}}^{\text{CCA2}}$ will return $D_{sk}(\mathbf{y})$ to A. Again this step can be repeated as long as A wants to.
5. A now outputs a bit b, signifying A's guess as to whether or not A is in this scenario.

Any time that a random oracle is used, then it is said that the work is done in the *random oracle model* (*ROM*). The formal version of definition 4.1 is then ready to be introduced.

Definition 4.3 (CCA2 Indistinguishable) *A public key cryptosystem* Π *with security parameter k is said to be* indistinguishable against adaptive chosen ciphertext attacks *(*CCA2 indistinguishable*) if for all probabilistic polynomial time adversaries A, it holds that* $Adv_A\left(O_{\text{real}}^{\text{CCA2}}, O_{\text{ideal}}^{\text{CCA2}}\right)$ *is negligible in k.*

If a public key cryptosystem is CCA2 indistinguishable, then it must also be CCA2 secure and vice versa. For this reason, the terms "CCA2 indistinguishable" and "CCA2 secure" will be used interchangeably. Additionally, the shorthand *IND-CCA2* is also often used. This notion of security that arises from being CCA2 indistinguishable was first introduced in [37] and is now widely regarded as the standard security notion for public key encryption schemes.

4.2.1 Non-Critical Attacks

The attacks presented here are considered not critical, because they can simply be circumvented by enlarging the parameter size(s) of the cryptosystem [26].

4.2.1.1 Generalised Information-Set-Decoding Attack

This attack was actually first proposed by McEliece himself in the original paper [33]. It was later made more systematic and generalised in [28] by Lee and Brickell. By assuming knowledge of an upper bound on the distance to the next codeword, it tries to solve the used special case of problem 3.1.

The general idea of the attack is that one assumes that an adversary is given a generator matrix G' of a linear error-correcting code along with a ciphertext $\mathbf{c} = \mathbf{m}G' \oplus \mathbf{z}$, where $\mathbf{z}$ is an error vector of weight t. Now the adversary will randomly choose k columns of G' and $\mathbf{c}$. If there are no errors in the chosen columns of $\mathbf{c}$ and if the matrix generated from the k columns chosen from G' put together provide an invertible matrix, then $\mathbf{m}$ can easily be determined.

Let G'_k, $\mathbf{c}_k$ and $\mathbf{z}_k$ denote the k columns picked from G', $\mathbf{c}$ and $\mathbf{z}$ respectively with $\mathbf{m}$ being the message that one wants to transmit. They have the following relationship

$$\mathbf{c}_k = \mathbf{m}G'_k\mathbf{z}_k.$$

Now if $\mathbf{z}_k = \mathbf{0}$ and if G'_k is non-singular, then $\mathbf{m}$ can be recovered using eq. (3.8), because it implies that

$$\mathbf{m} = \mathbf{c}_k {G'_k}^{-1}.$$

Even if $\mathbf{z}_k \neq \mathbf{0}$ then one can always make a new guess among small Hamming weights, so assume that $w(\mathbf{z}_k)$ is small, then $\mathbf{m}$ can be recovered by guessing a new $\mathbf{z}_{k'}$. The recovered plaintext can then be verified by checking whether or not $w\left(\mathbf{c} \oplus \mathbf{m}G'\right) = w\left((\mathbf{c}_{k'} \oplus \mathbf{z}_{k'}) {G'_{k'}}^{-1} G' \oplus \mathbf{c}\right) = t$ [17, 26].

Now let $I \subset \{0, \cdots, n-1\}$ be a set fulfilling that $|I| = k$. The generalised version of this attack in then summarised in algorithm 4.4, found in [17].

Algorithm 4.4 GISD

Input: $n \times k$ generator matrix G', a cipher text $\mathbf{c} = \mathbf{m}G' \oplus \mathbf{z}$, where $\mathbf{m}$ is the plain-text and $\mathbf{z}$ is the error vector of weight t, a positive integer $j \leq t$.
Output: The plaintext $\mathbf{m}$.

```
while true do
    Choose randomly $I \subset \{0, \cdots, n-1\}$, where $|I| = k$.
    $Q_1 = {G'_I}^{-1}$; $Q_2 = Q_1 G'$
    $\mathbf{e} = \mathbf{c} \oplus \mathbf{c}_I Q_2$
    for $i = 0$ to $j$ do
        for all $\mathbf{z}_I$ with $w(\mathbf{z}_I) = i$ do
            if $w(\mathrm{e} \oplus \mathrm{z} Q_2) = t$ then
                Return $(\mathrm{c}_\mathrm{I} \oplus \mathrm{z}_I) Q_1$
            end if
        end for
    end for
end while
```

4.2.1.1.1 Time Complexity Of algorithm 4.4

Here I will somewhat follow [17, 28] and try to give an estimate of the amount of work required to do the attack described in algorithm 4.4.

First of all notice that the number of sets I with $|I| = k$ is $\binom{n}{k}$. Now the amount of these sets such that there are i errors in $\mathbf{c}_I$ is $\binom{t}{i}\binom{n-t}{k-i}$. This means that if there can be at most j errors in $\mathbf{c}_I$, then there is a total of $\sum_{i=0}^{j}\binom{t}{i}\binom{n-t}{k-i}$ of such sets I, that are relevant. So in order for the `while`-loop to select the right set I it will have to run an expected

$$T_j = \frac{\binom{n}{k}}{\sum_{i=0}^{j}\binom{t}{i}\binom{n-t}{k-i}}$$

number of times.

The `while`-loop has a body too however. The two `for`-loops will here be taken together. Notice that the amount of error vectors $\mathbf{z}_I$ with $w(z_I) \leq j$ is given by

$$N_j = \sum_{i=0}^{j}\binom{k}{i}.$$

In [28] the authors propose using $j = 2$ to minimize the expected work factor, but since the expected work factor is clearly proportional to $T_j \cdot N_j$, large enough parameters should still be enough to prevent this type of attack.

4.2.1.2 Finding-Low-Weight-Codeword Attack

Suppose you have an encrypted message $\mathbf{c} = \mathbf{m}G' \oplus \mathbf{z}$. Now you could generate the $(k+1) \times n$ matrix

$$\begin{pmatrix} G' \\ c \end{pmatrix}$$

and the minimum weight codeword of this would then be the error vector $\mathbf{z}$. The idea behind this attack is then to find this codeword (solving a special case of problem 3.3), provided that $w(\mathbf{z}) \leq \frac{t}{2}$ and that the minimum distance of G' is t. This would completely break the McEliece PKC, since the knowledge of $\mathbf{z}$, $\mathbf{c}$ and G' is enough to find the plaintext $\mathbf{m}$ from eq. (3.8) as

$$\mathbf{m} = (\mathbf{c} \oplus \mathbf{z})\, G'^{-1}.$$

Now Leon, Canteaut and Chabaud and also Stern tried to make algorithms that does this attack by finding a low-weight codeword among codeswords generated by an arbitrary generator matrix using a database obtained by pre-computation [11, 29, 41].

Assume that a code C is given by some generator matrix G. The algorithms then start off by searching for codewords of small weight in a restricted part of the code generated by G_S, where S is some random subset of $\{0, 1, \cdots, n-1\}$. These codewords are then expanded to codewords in C and then comes a check to see whether or not the codewords in C has the desired weight. The algorithms differ in the way that they choose the set S and the way in which they search for codewords generated by G_S.

In [13] the computational cost of Stern's algorithm is evaluated and it is found that the attack based on this algorithm is infeasible for appropriate parameters. Canteaut and Chabaud's algorithm does not improve much upon this result and hence their algorithm is still not good enough to break the original McEliece cryptosystem in practice [17]. As for Leon's algorithm, it is shown in [15] that large enough parameters make the attack require way too much computing time in practice.

4.2.1.3 Statistical Decoding

This type of attack was first proposed in 2001 by Al Jabri in [24] and was later improved upon by Overbeck in [35].

The idea that led to this attack was that the vectors from the dual space of a linear block code that are not orthogonal to the ciphertext reveals some information on the positions of the errors used in encryption in the McEliece PKC – again if the errors used in encryption is found, the McEliece PKC is broken. So an algorithm is proposed, that can find a sufficient number of vectors in the dual code of a certain weight. The algorithm tries to find the most likely places in which errors was introduced during encryption.

However the amount of precomputation the algorithm will have to do seems to make sure, that the original McEliece cryptosystem is safe against this attack for parameter size(s) that also resist the attack described in section 4.2.1.2. The precise running time of this attack is not clear however, since the true minimum distance of the dual code is something that is not known much about [30].

4.2.1.3.1 Iterative Decoding An alternative to the statistical decoding attack, namely an *iterative decoding attack*, was proposed by Fossorier et al. in [18]. They point out that this needs less precomputation, but the amount of precomputation needed is still large enough to make it infeasible.

4.2.2 Critical Attacks

The attacks presented in this section can not be overcome by simply enlarging the parameter size(s) of the cryptosystem and is thus considered critical.

All of them aim to reveal some information on the plaintext or the error vector, $\mathbf{z}$, used for encryption. Most of them do not reveal all of the plaintext, but all of them do not manage to retrieve the private key. They do however provide a reduction in the computational complexity of further subsequent attacks.

One thing they all have in common though, is that if CCA2 security for the McEliece cryptosystem can be obtained, then they can all be avoided. This provides enough reason for developing a conversion or transformation of the cryptosystem that manages to achieve this level of security [17, 26].

4.2.2.1 Known-Partial-Plaintext Attack

An adversary attacking the original McEliece PKC might use partial knowledge on the target plaintext to obtain the full plaintext, because such partial knowledge reduces the computational cost of the attack, which in turn is the same as reducing the parameters of the cryptosystem [13].

Let $\bullet \| \bullet$ denote the concatenation of two strings. Now the length of a message, $\mathbf{m}$, sent using the McEliece PKC will have length k and k can be split up into the sum of two numbers; $k = k_l + k_r$. So let $\mathbf{m}_l$ and $\mathbf{m}_r$ denote the leftmost k_l bits and the remaining k_r bits of the message respectively. Suppose an adversary knows $\mathbf{m}_r$ and wants to recover the remaining bits of the message contained in $\mathbf{m}_l$. Now let G'_l and G'_r denote the upper k_l rows and the remaining k_r lower rows of G' respectively. It can now be seen that

$$\begin{aligned} \mathbf{c} &= \mathbf{m}G' \oplus \mathbf{z} \\ &= \mathbf{m}_l G'_l \oplus \mathbf{m}_r G'_r \oplus \mathbf{z}, \\ \Rightarrow \mathbf{c} \oplus \mathbf{m}_r G'_r &= \mathbf{m}_l G'_l \oplus \mathbf{z} \end{aligned}$$

and if $\mathbf{c}' = \mathbf{c} \oplus \mathbf{m}_r G'_r$ then

$$\mathbf{c}' = \mathbf{m}_l G'_l \oplus \mathbf{z}.$$

So the McEliece PKC with parameters (n, k) is then equivalent to another instance of the same with parameters (n, k_l). This attack is not full in and of itself, so a further attack can then be used subsequently. If enough of the plaintext is known beforehand, then the reduction might just be enough to make even one of the non-critical attacks feasible in practice (e.g. the attacks in sections 4.2.1.1 and 4.2.1.2).

4.2.2.2 Related-Message Attack

Suppose some adversary wish to attack the McEliece cryptosystem and has access to two different ciphertexts, $\mathbf{c}_1, \mathbf{c}_2$, coming from two messages, $\mathbf{m}_1, \mathbf{m}_2$, encrypted using two different error vectors, $\mathbf{z}_1, \mathbf{z}_2$. If the adversary knows some some linear relation, $\delta(\mathbf{m}_1, \mathbf{m}_2) = \mathbf{m}_1 \oplus \mathbf{m}_2$, between the two plaintexts, then the adversary should be able to determine the error bits used in encryption [9]. This is because

$$\mathbf{c}_1 \oplus \mathbf{c}_2 \oplus \delta(\mathbf{m}_1, \mathbf{m}_2) G' = \mathbf{z}_1 \oplus \mathbf{z}_2, \tag{4.5}$$

so the adversary can now choose k coordinates in $\mathbf{c}_1 \oplus \mathbf{c}_2 \oplus \delta(\mathbf{m}_1, \mathbf{m}_2) G'$ that have the value 0 and use the attack described in section 4.2.1.1 to attack either $\mathbf{c}_1$ or $\mathbf{c}_2$.

Let $P(\bullet)$ denote a probability distribution and let the j'th entry in $\mathbf{z}_i$ be denoted $\mathbf{z}_{i,j}$. Now supposing that $w(\mathbf{z}_1) = w(\mathbf{z}_2) = t$ is far smaller than $\frac{n}{2}$, then since

$$P\left(1 = \mathbf{z}_{1,j} = \mathbf{z}_{2,j}\right) = \left(\frac{t}{n}\right)^2$$

it follows that there is a high probability that a 0-coordinate in the left-hand side of eq. (4.5) is also a 0-coordinate in both $\mathbf{z}_1$ and $\mathbf{z}_2$. This enables the adversary to efficiently guess error bits and so the attack from section 4.2.1.1 can be used.

4.2.2.2.1 Message Resend Attack A special case of the attack just described is one where the same message is encrypted twice (or more) using different error vectors. In this case $\mathbf{z}_1 \oplus \mathbf{z}_2$ can be recovered as

$$\mathbf{z}_1 \oplus \mathbf{z}_2 = \mathbf{c}_1 \oplus \mathbf{c}_2.$$

4.2.2.3 Reaction Attack

This attack is a weaker version of an adaptively chosen ciphertext attack, because it uses a weaker assumption than the one for the CCA2 security. This is so because the attack only depends on the adversary observing the reaction of the receiver, who has the private key, but no decryption oracle is provided to the adversary. This attack also tries to determine error bits in a sent ciphertext. The attack was presented by Hall et al. in [21].

The idea behind this attack is as follows. Suppose some adversary intercepts a sent ciphertext $\mathbf{c}$ and flips a few bits (maybe even just one) to create $\mathbf{c}'$. Then $\mathbf{c}'$ is sent along to the intended recipient and the adversary then starts observing the reaction coming from this very same recipient. There are two possible reactions to this:

Reaction A: The recipient returns a `repeat`-request to the adversary, because the bit flips amounts to adding further errors in the message, so the recipient cannot decode properly.

Reaction B: The recipient returns an `acknowledgment` or do nothing, since a proper plaintext was decrypted.

Obviously if the total amount of errors has not exceeded t after the bit flipping, then reaction B would be observed. If however reaction A is observed, then the flipped bits would not have been in error in the first place. This means that if reaction A is observed, then the adversary can repeat the process over and over until finally having found the error vector $\mathbf{z}$ used for encryption.

Now the probability of needing more than k rounds before finding an error position will be

$$\frac{\binom{n-k}{t}}{\binom{n}{t}}$$

so the attack from section 4.2.1.1 can now be used to break the cryptosystem.

4.2.2.4 Malleability Attack

The concept of *non-malleable cryptography* was first introduced in 1991 by Dolev et al. in [16]. In order to understand this attack, one first needs to understand what is meant by malleable cryptography.

Definition 4.6 (Malleable- And Non-Malleable Cryptography) *Let* $\mathbf{c}$ *be a ciphertext coming from the encryption of message* $\mathbf{m}$*. If* $\mathbf{c}$ *can be changed into another valid ciphertext,* $\mathbf{c}'$*, that is an encryption of* $\mathbf{m}'$*, which has a known relation to* $\mathbf{m}$*, then the used cryptosystem is said to be* malleable*. If this is not possible, then the used cryptosystem is said to be* non-malleable*.*

An adversary might take advantage of a cryptosystem being malleable and use it to try to break said cryptosystem. Here comes an interesting result to note.

Theorem 4.7 *If a cryptosystem is non-malleable, then it is also CCA2 secure. Likewise if a cryptosystem is CCA2 secure, then it is also non-malleable.*

Proof See [4]. ∎

Now in 2000, Hung-Min Sun deviced the following attack on the McEliece PKC in [42].

Note that if one adds codewords to a ciphertext coming from the McEliece PKC, then that would yield another valid ciphertext (in the original attack proposed, all codewords are actually added). The attack is then as follows.

Let $G'[i]$ denote the i'th row in the public matrix G', $\mathbf{m}_1$ be an arbitrary vector with entries in $\{0, 1\}$ such that $w(\mathbf{m}_1) \neq 0$ (implying $w(\mathbf{m}_1) \geq 1$) and $I = \{i_1, i_2, \cdots\}$ be a set of coordinates i_j that correspond to the value 1 in $\mathbf{m}_1$. For a plaintext $\mathbf{m}$ that is encrypted using the McEliece PKC, denote by $\mathbf{m}'$ the plaintext $\mathbf{m}' = \mathbf{m} \oplus \mathbf{m}_1$. Finally the error vector from the McEliece PKC will be denoted as $\mathbf{z}$ as usual. The changed ciphertext is then calculated as

$$\mathbf{c}' = \mathbf{c} \bigoplus_{i \in I} G'[i] = (\mathbf{m} \oplus \mathbf{m}_1) G' \oplus \mathbf{z} = \mathbf{m}' G' \oplus \mathbf{z}.$$

So the original McEliece PKC is not non-malleable. In the scenario of a chosen ciphertext attack given in the start of this section, the adversary can ask the oracle to decrypt $\mathbf{c}'$. The oracle will return $\mathbf{m}' = \mathbf{m} \oplus \mathbf{m}_1$. Since $\mathbf{m}_1$ is known, the adversary can then obtain the target plaintext as

$$\mathbf{m} = \mathbf{m}' \oplus \mathbf{m}_1.$$

This proves the following.

Theorem 4.8 *The McEliece cryptosystem is malleable.*

Which in turn leads to the following

Corollary 4.9 *The original McEliece PKC is not CCA2 secure.*

Proof This follows from theorems 4.7 and 4.8. ∎

Chapter 5

Achieving CCA2 Security

In 1999 Fujisaki and Okamoto proposed a way to turn a weak public key cryptosystem into one that is IND-CCA2 in the random oracle model [19]. Their work was later improved upon in [22], where such a transformation has been broken down into several smaller transformations, achieving modularity. Any such transformation will usually be called an *FO*-transformation. Since this later paper improves upon the original work, I will follow this instead of the original in sections 5.6 and 5.7 and show how to build a version of the McEliece PKC that is CCA2 secure in section 5.8. The purpose of this chapter is then to show that key exchanges using the McEliece PKC can be done with CCA2 security. Before showing so though, there are five small sections of introduction that are needed if one is to understand the full picture.

5.1 Some Additional Security Notions

Much as in the start of section 4.2 I will here go over some security definitions that will be needed later in order to understand, what it actually is that this chapter aims to do.

5.1.1 One-Way Under Chosen Plaintext Attacks

There are several different ways of stating the security definition that I will attempt to arrive at in this section. For now I will follow the way that it is defined in [22]. So let $\Pi = (G, E, D)$ be a public key cryptosystem defined in the usual way with plaintext space $\mathcal{M}$ and let A be an adversary and O be an oracle. Consider the following game.

OW − CPA Both the adversary A and the oracle O gets the security parameter k as input.

1. O runs $G(k)$ to get (pk, sk) and gives pk to A.
2. O then chooses a valid plaintext $\mathbf{m}$ and encrypts it to $\mathbf{c} = E_{pk}(\mathbf{m})$ and gives $\mathbf{c}$ to A.
3. A then has to output a message $\mathbf{m}'$ and give it to O.

4. O checks if $D_{sk}(\mathbf{c}) = \mathbf{m}'$ and outputs 1 if this is true and 0 if this is not true.

The adversary A only wins the game if O outputs 1 in the last step of the game. Again O will implicitly reject messages (by outputting 0) if $\mathbf{m}'$ is not a valid plaintext.

The adversarial advantage of adversary A in the game $\mathrm{OW}-\mathrm{CPA}$ done one cryptosystem Π will be denoted by $Adv_{\Pi}^{\mathrm{OW-CPA}}(A)$. The game is usually denoted by $\mathrm{OW}-\mathrm{CPA}_{\Pi}^{A}$. In general any game Γ played by some algorithm A on cryptosystem Π will be denoted Γ_{Π}^{A} and the adversarial advantage in a given game Γ played by adversary A on cryptosystem Π will be denoted by $Adv_{\Pi}^{\Gamma}(A)$.

Notice that the advantage of an adversary in a given type of attack is just the probability that the adversary will succeed in its attack. It follows that

$$Adv_{\Pi}^{\mathrm{OW-CPA}}(A) = P\left(\mathrm{OW}-\mathrm{CPA}_{\Pi}^{A} \Rightarrow 1\right).$$

Which in turn gives rise to yet another security definition.

Definition 5.1 (OW − CPA) *Let* $\Pi = (G, E, D)$ *be a public key cryptosystem with key generating algorithm G, encryption algorithm E, decryption algorithm D and security parameter k. Let also A be an adversary to the system. If* $Adv_{\Pi}^{\mathrm{OW-CPA}}(A)$ *is negligible in k, then* Π *is said to be* one-way under chosen plaintext attacks *(*$\mathrm{OW}-\mathrm{CPA}$*).*

5.1.2 Plaintext Checking Attacks

Here I will follow [22] and give a formal introduction of what is called a *one-wayness under plaintext checking attack* (*OW-PCA*). This will lead to another notion of security – but this time the security notion is not a standard one.

Let $\Pi = (G, E, D)$ be a public key cryptosystem with security parameter k as in section 4.2, let A be an adversary and let O be an oracle. Assume that A has access to an oracle O_{PCO} that takes a message $\mathbf{m}$ and a ciphertext $\mathbf{c}$ and checks if $D_{sk}(\mathbf{m}) = \mathbf{c}$ and implicitly rejects non-valid plaintexts. Whenever some algorithm A has access to some oracle O, then denote it by A^{O}. The attack is then modelled as the following game.

OW − PCA Both the adversary $A^{O_{\mathrm{PCO}}}$ and the oracle O gets the security parameter k as input.

1. O runs $G(k)$ to get (pk, sk) and gives pk to $A^{O_{\mathrm{PCO}}}$.
2. O then chooses a valid plaintext $\mathbf{m}$ at random and encrypts it with the public key to get $\mathbf{c} = E_{pk}(\mathbf{m})$ and gives $\mathbf{c}$ to $A^{O_{\mathrm{PCO}}}$.
3. $A^{O_{\mathrm{PCO}}}$ now has to output a message $\mathbf{m}'$ and gives it to O.
4. O checks if $D_{sk}(\mathbf{c}) = \mathbf{m}'$ and outputs 1 if this is true and 0 if this is not true.

The adversary A only wins the game if O outputs 1 in the last step of the game. Please note that the definition of O in this game is such that it returns 0 if $\mathbf{m}'$ is not a valid message.

This gives the following.

$$Adv_{\Pi}^{\mathrm{OW-PCA}}\left(A^{O_{\mathrm{PCO}}}\right) = P\left(\mathrm{OW-PCA}_{\Pi}^{A^{O_{\mathrm{PCO}}}} \Rightarrow 1\right).$$

Now the non-standard security definition, that I mentioned earlier, can be introduced.

Definition 5.2 (OW − PCA) *A public key cryptosystem* Π *with security parameter k is said to be* one-way under plaintext checking attacks *(*OW − PCA*) if for all probabilistic polynomial time adversaries A, it holds that* $Adv_{\Pi}^{\mathrm{OW-PCA}}\left(A^{O_{\mathrm{PCO}}}\right)$ *is negligible in k.*

5.2 Hash Functions

Hash functions are defined as below.

Definition 5.3 (Hash Functions) *A* hash function *is a function that can map data of arbitrary size to an output that has a fixed number of bits in length.*

The output of a hash function may be called a *digest*, *hash codes*, *hash value* or just *hash*.

Now let A be an adversary and let h be a hash function.

Definition 5.4 (Collision) *Suppose A has access to a hash function h. Suppose also that A outputs two strings* $\mathbf{m}, \mathbf{m}'$*. If* $\mathbf{m} \neq \mathbf{m}'$ *whilst* $h(\mathbf{m}) = h(\mathbf{m}')$*, then A has found a* collision *of the hash function h.*

Definition 5.5 (Collision Intractable) *Let A be a probabilistic polynomial time adversary. If A can find a collision for the hash function h with only negligible probability, then the hash function h is said to be* collision intractable.

I have already used the term one-way. Let me just quickly introduce one-way functions formally though. A function f is said to be *one-way* if it is easy to compute on every input, but hard to invert given the image of a random input. Note that the words "easy" and "hard" here are to be understood in the realm of computational complexity.

The following theorem is well-known and rather interesting.

Theorem 5.6 *Collision intractable hash functions are one-way functions.*

5.3 Code Based Games

Sometimes it might be useful to introduce something called *code based games*. The games are essentially algorithms (some even with subroutines), but before introducing them, some notation is needed.

For a set S, let $x \overset{\$}{\leftarrow} S$ denote the uniform sampling of $x \in S$. If the sampling is done according to some distribution $\mathcal{D}$ however, then it will be written as $x \leftarrow \mathcal{D}$. Now when having a boolean statement B, $[\![B]\!]$ will return 1 if B is `true` and 0 if B is `false`.

Now as for algorithms, if they are given input x, then their deterministic computation will be written $y = A(x)$. As a general rule though, they will usually be assumed to be working probabilistic and thus the computation will be denoted $y \leftarrow A(x)$.

Hash functions $h : \mathcal{D}_h \to \mathcal{V}(h)$ will at times be modelled as random oracles. These will have a list of queries that they have been issued as a hash list $\mathcal{L}_h$, which contains tuples $(x, h(x))$ of arguments $x \in \mathcal{D}_h$ and answers $h(x)$. If $x \notin \mathcal{D}_h$, then the convention will be that $h(x) = \perp$ (where $\perp$ is used as a special symbol to denote a rejection).

The games themselves are constructed following [6, 40] with boolean flags initialised to `false`, numerical types to 0, sets to the empty set $\emptyset$ and strings to the empty string Λ. They will terminate, once they have returned some output.

The games will as usual be done by some adversary on a cryptosystem, so let the public key cryptosystem $\Pi = (G, E, D)$ be defined in the usual way. Π will have plaintext space $\mathcal{M}$ and ciphertext space $\mathcal{C}$. Additionally the public key pk will define some randomness space $\mathcal{R} = \mathcal{R}(pk)$. The used randomness might be made explicit by writing $\mathbf{c} \leftarrow E_{pk}(\mathbf{m}; \mathbf{r})$ where $\mathbf{r} \in \mathcal{R}$ (such notation can also be used for other probabilistic algorithms other than just the encryption function).

In order to not repeat things, sometimes the games will have lines with comments, where the comments signify which game the commented line belongs to. This is because some games include many of the same lines.

Algorithm 5.7 is an example of a simple subprocedure of a code based game and is essentially a plaintext checking oracle (earlier this was denoted as O_{PCO}). By convention, if $\mathbf{m} \notin \mathcal{M}$, then it returns $\mathrm{PCO}(\mathbf{m} \notin \mathcal{M}, \mathbf{c}) = \perp$ – that is it has rejection of non-valid messages.

Algorithm 5.7 PCO $(\mathbf{m} \in \mathcal{M}, \mathbf{c})$

```
  Return $[\![D_{sk}(\mathbf{c}) = \mathbf{m}]\!]$
```

Let $\bullet \wedge \bullet$ denote the logical AND-operator and let $\neg\bullet$ denote the event that something does not happen. The following lemma is due to [40] and will be useful later.

Lemma 5.8 (Difference Lemma) *Let A, B, F be events defined in some probability distribution and suppose that $A \wedge \neg F \Leftrightarrow B \wedge \neg F$. Then*

$$|P(A) - P(B)| \leq P(F).$$

Proof Assume that $A \wedge \neg F \Leftrightarrow B \wedge \neg F$ and then $P(A \wedge \neg F) = P(B \wedge \neg F)$. Now since $P(A \wedge F)$ and $P(B \wedge F)$ are both numbers between 0 and $P(F)$, it follows that

$$\begin{aligned} |P(A) - P(B)| &= |P(A \wedge F) + P(A \wedge \neg F) - P(B \wedge F) - P(B \wedge \neg F)| \\ &= |P(A \wedge F) - P(B \wedge F)| \\ &\leq P(F). \end{aligned}$$

■

5.4 Correctness Of Public Key Cryptosystems

Let $\mathrm{E}(\bullet)$ denote some expected value. Now if for some public key cryptosystem $\Pi = (G, E, D)$, it is true that

$$\mathrm{E}\left(\max_{\mathbf{m}\in\mathcal{M}} P\left(D_{sk}(\mathbf{c}) \neq \mathbf{m} \mid \mathbf{c} \leftarrow E_{pk}(\mathbf{m})\right)\right) \leq \delta$$

then Π will be called *δ-correct.* The expectation is taken over $(pk, sk) = G(k)$.

Suppose some adversary A plays the following game on Π with an oracle O.

COR Input to both A and O the security parameter of Π, k.

1. O runs $G(k)$ to generate (pk, sk) and gives this tuple to A.
2. A chooses a message $\mathbf{m}$ and gives to O.
3. O encrypts $\mathbf{m}$, by computing $\mathbf{c} = E_{pk}(\mathbf{m})$.
4. O checks if $D_{sk}(\mathbf{c}) = \mathbf{m}$ and returns 1 if this is true and 0 if this is not true.

In this case, δ-correctness will mean that

$$P\left(\mathrm{COR}_{\Pi}^{A} \Rightarrow 1\right) \leq \delta.$$

If however $\Pi = \Pi^{O_1}$ is defined in relation to some random oracle O_1, then the correctness bound might depend on the amount of queries made to said oracle O_1. Denote this amount by q_{O_1}. In this case the game will be called $\mathrm{COR} - \mathrm{RO}_{\Pi}^{A}$ and if A makes at most q_{O_1} queries to the oracle O_1, then if

$$P\left(\mathrm{COR} - \mathrm{RO}_{\Pi}^{A} \Rightarrow 1\right) \leq \delta\left(q_{O_1}\right),$$

Π will be said to be *$\delta\left(q_{O_1}\right)$-correct.* The game $\mathrm{COR} - \mathrm{RO}$ in which A has access to oracle O_1 and plays on cryptosystem Π with oracle O can be seen below.

COR − RO Input to both A and O the security parameter of Pi, k.

1. O runs $G(k)$ to get (pk, sk) and gives this tuple to A.
2. A^{O_1} chooses a message $\mathbf{m}$ and gives to O.
3. O encrypts $\mathbf{m}$, by computing $\mathbf{c} = E_{pk}(\mathbf{m})$.
4. O checks if $D_{sk}(\mathbf{c}) = \mathbf{m}$ and returns 1 if this is true and 0 if this is not true.

Note that if the number of oracle queries is 0, then $\delta\left(q_{O_1}\right)$ is constant and thus the above correctness definitions are a special case of correctness in the random oracle model.

If however Π is being defined in relation to two different random oracles, then the correctness error, δ, will have to be stated in terms of the amount of queries to both of these oracles.

5.4.1 Rigidity Of Public Key Cryptosystems

Bernstein and Persichetti came up with the notion of a cryptosystem being rigid in 2018 in [8]. The idea is basically just that a rigid cryptosystem will always return the correct plaintext, when decryption is run on an encryption of said plaintext and reject otherwise.

Definition 5.9 (Rigid Cryptosystem) *Let $\Pi = (G, E, D)$ be a deterministic cryptosystem. If for all key pairs $(pk, sk) \leftarrow G$ and all ciphertexts $\mathbf{c}$ it holds that $D_{sk}(\mathbf{c}) = \perp$ or $E_{pk}(D_{sk}(\mathbf{c})) = \mathbf{c}$, then Π is said to be* rigid.

5.5 Key Encapsulation

Here I will introduce key encapsulation mechanisms. The idea behind them is that they are a way of exchanging keys between two parties, without actually sending the key itself or an encryption of said key. Instead an encapsulation of the key is sent and the hope is that the desired key is hard to find from the encapsulation. This can prove useful in scenarios, where a key has to be negotiated (for example in the case of negotiating a key to be used for symmetric cryptosystems – which reflects how public key cryptosystems typically are used in practice).

A *key encapsulation mechanism* (*KEM*) consists of three algorithms. The algorithms are as follows. Firstly there is a *key generation algorithm*, Gen, that takes a security parameter k and outputs a key pair (pk, sk), where pk will define a finite key space $\mathcal{K}$.

$$\mathrm{Gen}(k) = (pk, sk)\,.$$

The second is a *key encapsulation algorithm*, Encaps, that takes the public key, pk, as a parameter and outputs a tuple, $(K, \mathbf{c})$, where $\mathbf{c}$ is an encapsulation of the key K.

$$\mathrm{Encaps}(pk) = (K, \mathbf{c})\,.$$

The third and final algorithm is a deterministic *decapsulation algorithm*, Decaps, that takes the private key, sk, and an encapsulation, $\mathbf{c}$, as parameters and outputs either a key $K = \mathrm{Decaps}(sk, \mathbf{c}) \in \mathcal{K}$ or rejects if $\mathbf{c}$ is not a valid encapsulation.

$$\mathrm{Decaps}(sk, \mathbf{c}) = \begin{cases} K = \mathrm{Decaps}(sk, \mathbf{c}) \in \mathcal{K} & \text{if } \mathbf{c} \text{ is a valid encapsulation} \\ \text{reject} & \text{if } \mathbf{c} \text{ is not a valid encapsulation} \end{cases}.$$

Such a key encapsulation mechanism will be written as

$$\mathrm{KEM} = (\mathrm{Gen}, \mathrm{Encaps}, \mathrm{Decaps})\,.$$

Now let P be a probability and if

$$P\left(\mathrm{Decaps}(sk, \mathbf{c}) \neq K \mid (pk, sk) = \mathrm{Gen},\ (K, \mathbf{c}) = \mathrm{Encaps}(pk)\right) \leq \delta$$

then the used key encapsulation mechanism will be called *δ-correct.* Additionally it is noted that this definition also makes sense in the random oracle model.

5.5.1 CCA2 In Terms of Key Encapsulation Mechanisms

One thing that will be useful later is a reformulation of the game from section 4.2 and its adversarial advantage. This will be done in the way of a game based on key encapsulation mechanisms just as in [22] and is written below.

Let O_{Decaps} be a decapsulation oracle, that will perform the decapsulation algorithm as defined above. Suppose $A^{O_{\text{Decaps}}}$ has access to this specific oracle and plays the following game with the oracle O^{CCA2}.

CCA2 Input to both O^{CCA2} and $A^{O_{\text{Decaps}}}$ the security parameter k.

1. O^{CCA2} runs $\text{Gen}(k)$ to get (pk, sk) and gives pk to $A^{O_{\text{Decaps}}}$.
2. O^{CCA2} then computes the tuple $(K_0, \mathbf{c}) = \text{Encaps}(pk)$. O^{CCA2} also chooses a random key $K_1 \in \mathcal{K}$. O^{CCA2} then gives a tuple $(K_b, \mathbf{c})$ to $A^{O_{\text{Decaps}}}$, where K_b is chosen among K_0 and K_1 uniformly at random.
3. $A^{O_{\text{Decaps}}}$ can make as many queries as it wants to O_{Decaps}, but at the end of this step, it has to output a guess b', signifying its guess as to whether it holds $(K_0, \mathbf{c})$ or $(K_1, \mathbf{c})$.
4. If $A^{O_{\text{Decaps}}}$ guesses correctly, then O^{CCA2} will output 1 and if $A^{O_{\text{Decaps}}}$ does not guess correctly, then O^{CCA2} will output 0.

Let κ denote the key encapsulation mechanism that is used and then this game has the adversarial advantage

$$Adv_{\kappa}^{\text{CCA2}}\left(A^{O_{\text{Decaps}}}\right) = \left| P\left(\text{CCA2}_{\kappa}^{A^{O_{\text{Decaps}}}} \Rightarrow 1\right) - \frac{1}{2} \right|.$$

Again if this is negligible, then the key encapsulation mechanism will be indistinguishable against adaptive chosen ciphertext attacks (IND-CCA2).

5.6 The First Transformation

Let $\Pi = (G, E, D)$ be a public key cryptosystem with message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$. Let h be a hash function

$$h : \mathcal{M} \to \mathcal{R}.$$

The following transformation will be called T and is in fact the Encrypt-with-Hash construction from [3] originally proposed by Bellare et al. in 1998 [5].

The transformation transforms a cryptosystem Π such as to build a new cryptosystem

$$\Pi_1 = T(\Pi, h) = (G, E_1, D_1)$$

that is deterministic. Please note that the key generation algorithm is exactly the same as in Π.

Let $\mathbf{m}, \mathbf{m}' \in \mathcal{M}$ and $\mathbf{c} \in \mathcal{C}$. Now I can describe just exactly how this new Π_1 works.

Key Generation: The same as in Π.

Encryption: To encrypt a plaintext $\mathbf{m}$, compute

$$\mathbf{c} = E_1(pk, \mathbf{m}) = E_{pk}(\mathbf{m}; h(\mathbf{m})).$$

Decryption: To decrypt a ciphertext $\mathbf{c}$ start by computing

$$\mathbf{m}' = D_{sk}(\mathbf{c}).$$

If $\mathbf{m}'$ is not a valid plaintext or if $E_1(pk, \mathbf{m}') \neq \mathbf{c}$ then abort. Else return $\mathbf{m}'$.

Here it is presupposed that E uses a random vector when encrypting plaintext $\mathbf{m}$, but in this case it is replaced by the digest of h on the message that is to be encrypted. The decryption function just works as usual with the additional step of actually checking that encryption was done correctly.

5.6.1 Implying OW − PCA From OW − CPA With Correctness

Here I will follow [22] in showing how the T-transformation can be used to achieve an OW − PCA cryptosystem from an OW − CPA one. So let Π and Π_1 be as above and assume that Π is δ-correct.

Consider an adversary A that plays the game COR − RO on Π_1. This game will involve at most q_h queries to the random oracle h, $h(\mathbf{m}_0), \cdots, h(\mathbf{m}_{q_h-1})$. Now if such a query yields $h(\mathbf{m}_i) = D_1(sk, E_1(pk, \mathbf{m}_i; h(\mathbf{m}_i))) \neq \mathbf{m}_i$, then the query will be called *problematic*. Since it is assumed that Π is δ-correct however, then given that h returns independently random values and averaging over (pk, sk), every $h(\mathbf{m}_i)$ is problematic with probability at most δ. So the probability of a query being problematic is at most $q_h \cdot \delta$ meaning that

$$P\left(\mathrm{COR} - \mathrm{RO}^A_{\Pi_1} \Rightarrow 1\right) \leq q_h \cdot \delta.$$

Notice also that the resulting Π_1 is deterministic. This proves the following.

Theorem 5.10 *Let Π be a public key cryptosystem that is δ-correct, h be a hash function and let $\Pi_1 = T(\Pi, h)$. Suppose A is playing the game $\mathrm{COR} - \mathrm{RO}^A_{\Pi}$ and makes at most q_h queries to the random oracle h. Then Π_1 is δ_1-correct with $\delta_1(q_h) = q_h \cdot \delta$. Furthermore Π_1 is rigid.*

With this out of the way, on to security then. Consider the code based games in algorithms 5.11 to 5.13.

Algorithm 5.11 Games G_0-G_2

```
$(pk, sk) \leftarrow G$
$\mathbf{m} \xleftarrow{\$} \mathcal{M}$
$\mathbf{c} \leftarrow E_1(pk, \mathbf{m})$
$\mathbf{m}' \leftarrow B^{h(\cdot), \mathrm{PCO}(\cdot,\cdot)}(pk, \mathbf{c})$
**Return** $[\![\mathbf{m}' = \mathbf{m}]\!]$
```

Algorithm 5.12 PCO $\left(\mathbf{m}'' \in \mathcal{M}, \mathbf{c}'\right)$

```
$\mathbf{m}' = D_1\left(sk, \mathbf{c}'\right)$                                              $\triangleright G_0$
**Return** $[\![\mathbf{m}' = \mathbf{m}'']\!] \wedge [\![E_1\left(pk, \mathbf{m}'; h\left(\mathbf{m}'\right)\right) = \mathbf{c}']\!]$   $\triangleright G_0$
**Return** $[\![E_1\left(pk, \mathbf{m}''; h\left(\mathbf{m}''\right)\right) = \mathbf{c}']\!]$                    $\triangleright G_1$-$G_2$
```

Algorithm 5.13 $h\left(\mathbf{m}''\right)$

```
**if** $\exists r \ : \ \left(\mathbf{m}'', r\right) \in \mathcal{L}_h$ **then**
    **Return** $r$
**end if**
**if** $\mathbf{m}' = \mathbf{m}''$ **then**                          $\triangleright G_2$
    QUERY = true                              $\triangleright G_2$
    **Abort**                                 $\triangleright G_2$
**end if**                                    $\triangleright G_2$
$r \overset{\$}{\leftarrow} \mathcal{R}$
$\mathcal{L}_h = \mathcal{L}_h \cup \left\{\left(\mathbf{m}', r\right)\right\}$
**Return** $r$
```

The claim is that $\Pi_1 = T(\Pi)$ is a OW – PCA secure public key cryptosystem. To show this, suppose $B = B^{h(\bullet), \mathrm{PCO}(\bullet,\bullet)}$ is an adversary against the OW – PCA security of Π_1 playing the games G_0 to G_2 that has access to h as defined in algorithm 5.13 and PCO as defined in algorithm 5.12. Suppose further that B issues at most q_h issues to h and at most q_P queries to PCO.

As for game G_0, this is just the original OW – PCA game. The random oracle queries are stored in the set $\mathcal{L}_h$ and by convention, $h\left(\mathbf{m}'\right) = r \Leftrightarrow \left(\mathbf{m}', r\right) \in \mathcal{L}_h$. This gives

$$P\left(G_0^B \Rightarrow 1\right) = Adv_{\Pi_1}^{\mathrm{OW-PCA}}(B).$$

Now in games G_1 and G_2, the plaintext checking oracle PCO $\left(\mathbf{m}'', \mathbf{c}'\right)$ is replaced by a simulation that no longer does the check to see whether or not $\mathbf{m}'' = \mathbf{m}'$, with $\mathbf{m}' = D_1\left(sk, \mathbf{c}'\right)$. Observe that the whole game G_1 makes at most q_h queries to h. In the case that no queries $h\left(\mathbf{m}_i'\right)$ is problematic, then game G_1 proceeds and if B submits a PCO query $\left(\mathbf{m}'', \mathbf{c}'\right)$ together with an h query $h\left(\mathbf{m}'\right)$ that is problematic and $\mathbf{c}' = E_1\left(pk, \mathbf{m}''; h\left(\mathbf{m}''\right)\right)$, then G_1 returns 1. Together with theorem 5.10 this gives

$$\left|P\left(G_1^B \Rightarrow 1\right) - P\left(G_0^B \Rightarrow 1\right)\right| \leq \left(q_h + q_P\right) \cdot \delta.$$

G_2 and G_1 only differs if the flag QUERY is raised, in which case G_2 aborts. If this flag is raised, it means that B made a query to h on the target plaintext $\mathbf{m}$, since $(\mathbf{m}, \bullet) \in \mathcal{L}_h$. Now lemma 5.8 gives

$$\left|P\left(G_2^B \Rightarrow 1\right) - P\left(G_1^B \Rightarrow 1\right)\right| \leq P(\mathrm{QUERY}).$$

Consider the adversary C against the OW – CPA security of the original cryptosystem Π from algorithm 5.14.

Algorithm 5.14 $C\left(pk, \mathbf{c}\right)$

```
    $\mathbf{m}' \leftarrow B^{h(\bullet),\mathrm{PCO}(\bullet,\bullet)}\left(pk, \mathbf{c}\right)$
    Return $\mathbf{m}'$
```

C starts out by having input $\left(pk, \mathbf{c} \leftarrow E_{pk}\left(\mathbf{m}\right)\right)$, where $\mathbf{m}$ is random and unknown. Then C perfectly simulates the game G_2 for B and finally it outputs $\mathbf{m}' = \mathbf{m}$ if B wins the game G_2. This gives

$$P\left(G_2^B \Rightarrow 1\right) = Adv_{\Pi}^{\mathrm{OW-CPA}}\left(C\right).$$

Taking all of this so far together establishes the bound

$$Adv_{\Pi_1}^{\mathrm{OW-PCA}}\left(B\right) \le \left(q_h + q_P\right) \cdot \delta + P\left(\mathrm{QUERY}\right) + Adv_{\Pi}^{\mathrm{OW-CPA}}\left(C\right).$$

Algorithm 5.15 $\mathrm{D}\left(pk, \mathbf{c}\right)$

```
    $\mathbf{m}'' \leftarrow B^{h(\bullet),\mathrm{PCO}(\bullet,\bullet)}\left(pk, \mathbf{c}\right)$
    $\left(\mathbf{m}', \mathbf{r}'\right) \overset{\$}{\leftarrow} \mathcal{L}_h$
    Return $\mathbf{m}'$
```

Consider now the adversary D from algorithm 5.15 against the OW − CPA security of the not yet transformed cryptosystem Π. It starts out by having input $\left(pk, \mathbf{c} \leftarrow E_{pk}\left(\mathbf{m}\right)\right)$ and then perfectly simulates B in game G_2. If QUERY is set in G_2, then there must exist an entry $\left(\mathbf{m}, \bullet\right) \in \mathcal{L}_h$ and so D will return the correct $\mathbf{m}' = \mathbf{m}$ with probability at most $\frac{1}{q_h}$. This means that

$$P\left(\mathrm{QUERY}\right) \le \left(q_h + q_P\right) \cdot Adv_{\Pi}^{\mathrm{OW-CPA}}\left(\mathrm{D}\right).$$

Now let F be a single adversary to the OW − CPA security of Π and assume that the running time of B is about that of F. Folding C and D into one single adversary F and collecting the probabilities that were deduced from the games in this section then yields

$$Adv_{\Pi_1}^{\mathrm{OW-PCA}}\left(B\right) \le \left(q_h + q_P\right) \cdot \delta + \left(q_h + q_P + 1\right) \cdot Adv_{\Pi}^{\mathrm{OW-CPA}}\left(F\right) \tag{5.16}$$

proving the following theorem.

Theorem 5.17 *Let* Π *be a public key cryptosystem that is* OW − CPA, *h be a hash function and let* $\Pi_1 = T\left(\Pi, h\right)$. *Then* Π_1 *is* OW − PCA.

5.7 The Second Transformation

Let $\Pi_1 = \left(G_1, E_1, D_1\right)$ be a public key cryptosystem with message space $\mathcal{M}$, g be a hash function an let $\bullet^*$ denote the Kleene Star. g will then be defined by

$$g : \{0,1\}^* \to \mathcal{M}.$$

The following transformation will be called $U^{\not\perp}$, is due to [22] and yields a key encapsulation mechanism with implicit rejection (in the context of Fujisaki-Okamoto transformations this idea was originally proposed by Persichetti in [36, section 5.3])

$$\mathrm{KEM}^{\not\perp} = U^{\not\perp}\left(\Pi_1, g\right) = \left(\mathrm{Gen}^{\not\perp}, \mathrm{Encaps}, \mathrm{Decaps}^{\not\perp}\right).$$

The details of $\mathrm{KEM}^{\not\perp}$ are given here.

Gen$^{\perp}$: Use G_1 to generate a key pair (pk, sk). Choose a uniformly random string $\mathbf{s} \in \mathcal{M}$ and let $sk' = (sk, \mathbf{s})$.

Public Key: pk.

Private Key: sk'.

Encaps: To perform the key encapsulation, let $\mathbf{m} \in \mathcal{M}$ be any valid message and compute

$$\mathbf{c} = E_1(pk, \mathbf{m}).$$

Then compute

$$K = g(\mathbf{m}, \mathbf{c})$$

and return $(K, \mathbf{c})$.

Decaps$^{\perp}$: Notice that $sk' = (sk, \mathbf{s})$. When receiving $\mathbf{c}$, compute

$$\mathbf{m}' = D_1(sk, \mathbf{c}).$$

If decryption succeeds and $\mathbf{m}'$ is a valid message then return $K = g\left(\mathbf{m}', \mathbf{c}\right)$, otherwise return $K = g(\mathbf{s}, \mathbf{c})$.

5.7.1 Implying IND-CCA2 from OW – PCA With Correctness

The sole reason for using the $U^{\perp}$ transform and make the underlying cryptosystem, Π_1, into a KEM is to obtain a way to exchange keys that is secure in the CCA2 random oracle model. The fact that this is indeed possible will be shown here, following [22].

First a theorem about correctness being preserved.

Theorem 5.18 *Let Π_1 be a δ_1-correct public key cryptosystem and g be a hash function, then $\mathrm{KEM}^{\perp} = U^{\perp}(\Pi_1, g)$ is also δ_1-correct in the random oracle model.*

Proof Since g is a hash function and thus deterministic, the only randomness in $\mathrm{KEM}^{\perp}$ actually comes from Π_1. This in particular means that

$$P\left(\mathrm{Decaps}^{\perp}\left(sk', \mathbf{c}\right) \neq K \mid (pk, sk) = G_1,\ (K, \mathbf{c}) = \mathrm{Encaps}(pk)\right) \leq \delta_1. \qquad \blacksquare$$

Let $A = A^{\mathrm{Decaps}(\bullet), g(\bullet,\bullet)}$ be an adversary against the CCA2 security of $\mathrm{KEM}^{\perp}$, that issues at most q_D queries to $\mathrm{Decaps}^{\perp}$ and at most q_g queries to g. Now consider the games G_3 to G_6 from algorithms 5.19 to 5.22.

Algorithm 5.19 Games G_3-G_6

```
$(pk, sk) \leftarrow \mathrm{Gen}_1$
$\mathbf{s} \xleftarrow{\$} \mathcal{M}$
$sk' = (sk, \mathbf{s})$
$\mathbf{m} \xleftarrow{\$} \mathcal{M}$
$\mathbf{c} \leftarrow E_1(pk, \mathbf{m})$
$K_0 = g(\mathbf{m}, \mathbf{c})$
$K_1 \xleftarrow{\$} \{0,1\}^k$
$b \xleftarrow{\$} \{0,1\}$
$b' \leftarrow A^{\mathrm{Decaps}(\bullet), g(\bullet,\bullet)}(pk, \mathbf{c}, K_b)$
**Return** $[\![b' = b]\!]$
```

Algorithm 5.20 $g\left(\mathbf{m}'', \mathbf{c}'\right)$

```
if $\exists K \quad \left(\mathbf{m}'', \mathbf{c}', K\right) \in \mathcal{L}_g$ then
    Return $K$
end if
$K \xleftarrow{\$} \mathcal{K}$
if $\mathbf{m}'' = \mathbf{s}$ then                          $\triangleright G_4$-$G_6$
    QUERY = true                               $\triangleright G_4$-$G_6$
    Abort                                      $\triangleright G_4$-$G_6$
end if                                         $\triangleright G_4$-$G_6$
if $D_1\left(sk, \mathbf{c}'\right) = \mathbf{m}''$ then               $\triangleright G_5$-$G_6$
    if $\mathbf{c}' = \mathbf{c}$ then                          $\triangleright G_6$
        CHAL = true                            $\triangleright G_6$
        Abort                                  $\triangleright G_6$
    end if                                     $\triangleright G_6$
    if $\exists K' \; : \; \left(\mathbf{c}', K'\right) \in \mathcal{L}_D$ then          $\triangleright G_5$-$G_6$
        $K = K'$                                 $\triangleright G_5$-$G_6$
    else                                       $\triangleright G_5$-$G_6$
        $\mathcal{L}_D = \mathcal{L}_D \cup \left\{\left(\mathbf{c}', K\right)\right\}$           $\triangleright G_5$-$G_6$
    end if                                     $\triangleright G_5$-$G_6$
end if                                         $\triangleright G_5$-$G_6$
$\mathcal{L}_g = \mathcal{L}_g \cup \left\{\left(\mathbf{m}'', \mathbf{c}', K\right)\right\}$
Return $K$
```

Algorithm 5.21 $\text{Decaps}^{\perp}\left(\mathbf{c}' \neq \mathbf{c}\right)$ for G_3-G_4

```
$\mathbf{m}' = D_1\left(sk, \mathbf{c}'\right)$
if $\mathbf{m}' = \perp$ then                           $\triangleright G_3$
    Return $K = g\left(\mathbf{s}, \mathbf{c}'\right)$                 $\triangleright G_3$
end if                                         $\triangleright G_3$
if $\mathbf{m}' = \perp$ then                           $\triangleright G_4$
    Return $K = g'\left(\mathbf{c}'\right)$                    $\triangleright G_4$
end if                                         $\triangleright G_4$
if $\mathbf{m}' = \mathbf{s}$ then                           $\triangleright G_4$
    Return $K = g'\left(\mathbf{c}'\right)$                    $\triangleright G_4$
end if                                         $\triangleright G_4$
Return $K = g\left(\mathbf{m}', \mathbf{c}'\right)$
```

Algorithm 5.22 $\text{Decaps}^{\perp}\left(\mathbf{c}' \neq \mathbf{c}\right)$ for G_5-G_6

```
if $\exists K \; : \; \left(\mathbf{c}', K\right) \in \mathcal{L}_D$ then
    Return $K$
else
    $K \xleftarrow{\$} \mathcal{K}$
    $\mathcal{L}_D = \mathcal{L}_D \cup \left\{\left(\mathbf{c}', K\right)\right\}$
    Return $K$
end if
```

Game G_3 is just the original CCA2 game and thus gives

$$\left|P\left(G_3^A \Rightarrow 1\right) - \frac{1}{2}\right| = Adv_{\mathrm{KEM}^{\perp}}^{\mathrm{CCA2}}(A).$$

Game G_4 changes two things. Firstly if $g(\mathbf{s}, \bullet)$ is queried, then the boolean flag QUERY is raised. Secondly, $\mathrm{Decaps}^{\perp}\left(\mathbf{c}'\right)$ is made perfectly random. If $\mathbf{m}' = D_1\left(sk, \mathbf{c}'\right) = \perp$ or $\mathbf{m}' = D_1\left(sk, \mathbf{c}'\right) = \mathbf{s}$ then $K = g\left(\mathbf{s}, \mathbf{c}'\right)$ is replaced by $K = g'\left(\mathbf{c}'\right)$ $\left(g'\right.$ is to be regarded as an internal random oracle, that cannot be accessed by $\left.A\right)$. If $\mathbf{m}' = D_1\left(sk, \mathbf{c}'\right) = \mathbf{s}$ then A will only notice in the case that A queries $g(\mathbf{s}, \bullet)$. In that case, G_4 will abort. Unless G_4 aborts, then A's view is independent of the secret $\mathbf{s}$. This leads to

$$\left|P\left(G_4^A \Rightarrow 1\right) - P\left(G_3^A \Rightarrow 1\right)\right| \leq \frac{q_g}{|\mathcal{M}|}.$$

In game G_5 the $\mathrm{Decaps}^{\perp}$ is modified in a way so as to not use the secret key any more. Two lists, $\mathcal{L}_g$ and $\mathcal{L}_D$, are now used. If $\left(\mathbf{m}'', \mathbf{c}', K\right) \in \mathcal{L}_g$ then g must have been queried on $\left(\mathbf{m}'', \mathbf{c}'\right)$. This also means that $g\left(\mathbf{m}'', \mathbf{c}'\right) = K$. If $\left(\mathbf{c}', K\right) \in \mathcal{L}_D$ then $\mathrm{Decaps}^{\perp}\left(\mathbf{c}'\right) = K$ and either g was queried on $\mathbf{m}'' = D_1(sk, \mathbf{c})$ or $\mathrm{Decaps}^{\perp}$ was queried on $\mathbf{c}'$.

Now let $\mathbf{c}'$ be a fixed ciphertext and let $\mathbf{m}' = D_1\left(sk, \mathbf{c}'\right)$. Since $g(\perp, \bullet)$ is not allowed and $g\left(\mathbf{s}, \mathbf{c}'\right)$ results in abort, then if $\mathbf{m}' \in \{\perp, \mathbf{s}\}$, g can never add the tuple $\left(\mathbf{c}', K\right)$ to $\mathcal{L}_D$. So in this case, both G_4 and G_5 will have their $\mathrm{Decaps}^{\perp}$ return a uniformly random key.

So assume that $\mathbf{m}' \notin \{\perp, \mathbf{s}\}$ and that $D_1\left(sk, \mathbf{c}'\right) \neq \mathbf{s}$. In this case, if g is queried on $\left(\mathbf{m}', \mathbf{c}'\right)$ before $\mathrm{Decaps}^{\perp}$, then $D_1\left(sk, \mathbf{c}'\right) = \mathbf{m}'$ is recognised and $\left(\mathbf{c}', K''\right) \notin \mathcal{L}_D$ for any K''. So g adds $\left(\mathbf{m}', \mathbf{c}', K \xleftarrow{\$} \mathcal{K}\right)$ to $\mathcal{L}_g$ and adds $\left(\mathbf{c}', K\right)$ to $\mathcal{L}_D$, giving $\mathrm{Decaps}^{\perp}\left(\mathbf{c}'\right) = K = g\left(\mathbf{m}', \mathbf{c}'\right)$.

If however $\mathrm{Decaps}^{\perp}$ is queried on $\mathbf{c}'$ first, then $\left(\mathbf{c}', K''\right) \notin \mathcal{L}_D$ for any K''. So $\mathrm{Decaps}^{\perp}$ adds $\left(\mathbf{c}', K \xleftarrow{\$} \mathcal{K}\right)$ to $\mathcal{L}_D$ and thus $\mathrm{Decaps}^{\perp}\left(\mathbf{c}'\right) = K$. If g is then subsequently queried on $\left(\mathbf{m}', \mathbf{c}'\right)$, then $D_1\left(sk, \mathbf{c}'\right) = \mathbf{m}'$ is recognised and the entry $\left(\mathbf{c}', K\right) \in \mathcal{L}_D$ is also recognised, so g adds the tuple $\left(\mathbf{m}', \mathbf{c}', K\right)$ to $\mathcal{L}_g$ and returns K. This ensures that $g\left(\mathbf{m}', \mathbf{c}'\right) = K$.

All of this just goes to show that the view of A is the same in both G_4 and G_5. So this leads to

$$P\left(G_5^A \Rightarrow 1\right) = P\left(G_4^A \Rightarrow 1\right).$$

Now in game G_6, if $\mathbf{m}$ is the challenge message and A queries g on $(\mathbf{m}, \mathbf{c})$, then the boolean flag CHAL is raised and G_6 is aborted. Using lemma 5.8 then gives

$$\left|P\left(G_6^A \Rightarrow 1\right) - P\left(G_5^A \Rightarrow 1\right)\right| \leq P(\mathrm{CHAL}).$$

Since $g(\mathbf{m}, \mathbf{c})$ will never be given to A in G_6, A has to guess the bit b in order to win the game giving

$$P\left(G_6^A\right) = \frac{1}{2}.$$

Algorithm 5.23 $B^{\mathrm{PCO}(\cdot,\cdot)}\left(pk', \mathbf{c}\right)$

```
$K^{\dagger} \xleftarrow{\$} \mathcal{K}$
$\mathbf{s} \xleftarrow{\$} \mathcal{M}$
$b' \leftarrow A^{\mathrm{Decaps}^{\perp}(\cdot), g(\cdot,\cdot)}\left(pk', \mathbf{c}, K^{\dagger}\right)$
**if** $\exists\left(\mathbf{m}', \mathbf{c}'' K'\right) \in \mathcal{L}_g \ : \ \mathrm{PCO}\left(\mathbf{m}', \mathbf{c}\right) = 1$ **then**
    **Return** $\mathbf{m}'$
**else**
    **Abort**
**end if**
```

Algorithm 5.24 $g\left(\mathbf{m}'', \mathbf{c}'\right)$

```
**if** $\exists K \ : \ \left(\mathbf{m}'', \mathbf{c}', K\right) \in \mathcal{L}_g$ **then**
    **Return** $K$
**end if**
$K \xleftarrow{\$} \mathcal{K}$
**if** $\mathbf{m}'' = \mathbf{s}$ **then**
    **Abort**
**end if**
**if** $\mathrm{PCO}\left(\mathbf{m}'', \mathbf{c}'\right) = 1$ **then**
    **if** $\exists K' \ : \ \left(\mathbf{c}', K'\right) \in \mathcal{L}_D$ **then**
        $K = K'$
    **else**
        $\mathcal{L}_D = \mathcal{L}_D \cup \left\{\left(\mathbf{c}', K\right)\right\}$
    **end if**
**end if**
$\mathcal{L}_g = \mathcal{L}_g \cup \left\{\left(\mathbf{m}'', \mathbf{c}', K\right)\right\}$
**Return** $K$
```

Suppose now that the OW − PCA security adversary B for Π_1 is built as in algorithms 5.23 and 5.24, with B having access to a plaintext checking oracle, $\mathrm{PCO}(\cdot,\cdot)$ and $\mathrm{Decaps}^{\perp}(\cdot)$ defined as in G_6. Suppose furthermore that the running time of B is about that of A. This adversary B then perfectly simulates G_6 for A. If the flag CHAL is raised, then A must have queried g on $(\mathbf{m}, \mathbf{c})$ and $\left(\mathbf{m}, \mathbf{c}, K'\right) \in \mathcal{L}_g$ for some K'. A will then return $\mathbf{m}' = \mathbf{m}$ and win the OW − PCA game. So

$$P(\mathrm{CHAL}) = Adv_{\Pi_1}^{\mathrm{OW-PCA}}(B).$$

Now collecting the probabilities deduced from the games of this section yields

$$Adv_{\mathrm{KEM}^{\perp}}^{\mathrm{CCA2}}(A) \leq \frac{q_g}{|\mathcal{M}|} + Adv_{\Pi_1}^{\mathrm{OW-PCA}}(B), \tag{5.25}$$

proving the following theorem.

Theorem 5.26 *Let* Π_1 *be a public key cryptosystem that is* OW − PCA, g *be a hash function and let* $\mathrm{KEM}^{\perp} = U^{\perp}(\Pi_1, g)$ *be a key encapsulation mechanism, then* $\mathrm{KEM}^{\perp}$ *is* IND-CCA2.

5.8 Transforming The Original McEliece Public Key Cryptosystem

The goal of this section is to apply the T- and then the $U^{\perp}$-transformation to the McEliece PKC in order to achieve IND-CCA2 security in the random oracle model.

Before doing so however, some things has to be set straight. Firstly, the security reduction from section 5.6.1 only works if the McEliece PKC is OW – CPA. Note that if some probabilistic polynomial time adversary A were to attempt to win the game OW – CPA played on the McEliece PKC, then it would have to solve problem 3.10, but doing so and winning with non-negligible probability would contradict conjecture 3.11.

Secondly there's the question of the hash function to be used. Remember that the underlying Goppa code of the McEliece PKC can only correct up to $\lfloor \frac{n-1}{2} \rfloor$ errors. So a hash function that takes a message as input and outputs a pseudorandom vector of weight at most t is needed (it needs to be pseudorandom, so as to make sure that the vector $\mathbf{z}$ used for encryption is not easily guessed). The hash function also needs to be defined for all possible input messages and preferably also be easy to sample. Are such hash functions even possible? Do they even exist?

5.8.1 The Hash Functions

As for the hash function used in the T-transformation, [1] provides a hash function that is exactly what is needed. The authors claim that they built it so that it takes time linear in the length of the input message to compute the digest, it is defined on all possible messages, it is injective and it provides digests of a constant weight and the digest has an entropy higher than or equal to that of the input message. If this last claim is true, then if any adversary was to try to guess the error vector $\mathbf{z}$, then they might as well just try to guess the message itself. The only downside to using this function is that the authors seem to be requiring that $2^k < \binom{n}{t}$ (they do themselves point out that there are other such functions that do not have this potential problem).

As for the transformation $U^{\perp}$ all that is needed is a pseudorandom collision intractable hash function that outputs messages of length k. It needs to be pseudorandom, because otherwise the key that is to be exchanged might easily be guessed or at least have a lower entropy than what could have been achieved, it needs to be collision intractable, because this makes sure, that any adversary playing the CCA2 game for KEMs will have a hard time winning said game and it needs to output messages of length k so as to make sure that the output is within the message space of the McEliece PKC (with k being as in the original McEliece PKC). Fortunately such hash functions are well-known and have been so for quite some time now.

5.8.2 Applying The Transformations

Having solved the pertaining issues to the application of the transformations T and $U^{\perp}$ to the McEliece PKC, I will now move on to show how the CCA2 secure converted McEliece PKC looks like. Notice that after applying firstly the T- and then the $U^{\perp}$ transformation, this is actually a key encapsulation mechanism now.

System Parameters: $n, t \in \mathbb{N}$ where $t \leq \left\lfloor \frac{n-1}{2} \right\rfloor$.

Key Generation: Given the parameters n, t generate the following matrices:

G: $k \times n$ generator matrix of an irreducible binary $[n, k]$ Goppa code $\mathcal{G}$ which can correct up to t errors.
S: $k \times k$ random binary non-singular matrix.
P: $n \times n$ random permutation matrix.

Then compute the $k \times n$ matrix $G' = \mathrm{S}G\mathrm{P}$,
build a hashing algorithm h that outputs bit strings of length n and (Hamming) weight t (see e.g. [1]),
choose a collision intractable hash function g that takes input of arbitrary length and gives an output of length k and
choose a uniformly random message $\mathbf{s}$ of length k.

Public Key: $\left(G', t, h, g\right)$.

Private Key: $\left(S, D_{\mathcal{G}}, P, \mathbf{s}\right)$ where $D_{\mathcal{G}}$ is an efficient decoding algorithm for $\mathcal{G}$ (see e.g. algorithm 2.22).

Encapsulation: Start by encrypting the message $\mathbf{m} \in \{0, 1\}^k$. This is done by computing $h(\mathbf{m}) = \mathbf{z} \in \{0, 1\}^n$ and then computing the ciphertext

$$\mathbf{c} = \mathbf{m}G' \oplus \mathbf{z}.$$

Then compute the key

$$K = g(\mathbf{m}, \mathbf{c}).$$

Decapsulation: Upon receiving $\mathbf{c}$, start off by decrypting $\mathbf{c}$. This is done by calculating

$$\mathbf{c}\mathrm{P}^{-1} = \mathbf{m}'\mathrm{S}G \oplus \mathbf{z}\mathrm{P}^{-1}.$$

Then apply the decoding algorithm $D_{\mathcal{G}}$ to this. Because $\mathbf{c}\mathrm{P}^{-1}$ should have a hamming distance of t to the Goppa code, the codeword obtained should be

$$\mathbf{m}'\mathrm{S}G = D_{\mathcal{G}}\left(\mathbf{c}\mathrm{P}^{-1}\right).$$

Now one can now compute the plaintext $\mathbf{m}'$ as

$$\mathbf{m}' = \left(\mathbf{m}'\mathrm{S}G\right) G^{-1}\mathrm{S}^{-1}.$$

Then compute

$$\mathbf{c}' = \mathbf{m}'G' \oplus h\left(\mathbf{m}'\right).$$

If $\mathbf{m}'$ is not a valid plaintext or if $\mathbf{c}' \neq \mathbf{c}$, then decryption has failed.
If decryption succeeds however, then compute

$$K' = g\left(\mathbf{m}', \mathbf{c}\right).$$

Otherwise compute

$$K' = g(\mathbf{s}, \mathbf{c}).$$

Notice here that it does not make sense to talk of a non-valid plaintext, unless it is not of length k, which it will be, if protocol is followed. The check to see whether or not $\mathbf{m}'$ is a valid plaintext is then simply included here for the case in which two parties agree to some arbitrary structure on $\mathbf{m}$. In the real world, one would just choose $\mathbf{m}$ to be uniformly random.

5.8.2.1 Security After Applying Both Transforms In Succession

In [22] there also is a discussion of what the security of the transformed cryptosystem is like. Here I will rehash this in terms of applying the transformations to the McEliece PKC.

Denote the original McEliece PKC as MCE. Now that both transforms have been applied, one ends up with the key encapsulation mechanism

$$\mathrm{KEM}^{\perp} = \mathrm{FO}^{\perp}(\mathrm{MCE}, h, g) = U^{\perp}(T(\mathrm{MCE}, h), g) = \left(\mathrm{Gen}^{\perp}, \mathrm{Encaps}, \mathrm{Decaps}^{\perp}\right)$$

as described above.

Let B be a CCA2 adversary against $\mathrm{KEM}^{\perp}$, A be an adversary against the OW − CPA security of $T(\mathrm{MCE}, h)$ and let $q_{\mathrm{RO}} = q_h + q_g$ denote the total number of queries that B sends to the random oracles h and g. Combining eqs. (5.16) and (5.25) then gives

$$Adv_{\mathrm{MCE}}^{\mathrm{CCA2}}(B) \leq q_{\mathrm{RO}} \cdot \delta + \frac{2q_{\mathrm{RO}}}{|\mathcal{M}|} + 2q_{\mathrm{RO}} \cdot Adv_{\mathrm{MCE}}^{\mathrm{OW-CPA}}(A). \tag{5.27}$$

Remember from the discussion in section 3.3.2, that security can be stated in terms of only k. One could choose a number $\kappa \in \mathbb{N}$ and let that be the security parameter. This κ could be many things, but for now let it denote the bit length of k. The phrase "κ bits of security" then actually makes sense. Suppose adversary B runs in time $t(B)$ and has advantage $Adv(B)$. "κ bits of security" would then mean that

$$\frac{t(B))}{Adv(B)} \geq 2^{\kappa}.$$

Some recommendations on the bounds of δ and $|\mathcal{M}|$ can then be made for the security bound in eq. (5.27). These can be seen in table 5.28.

Term in concrete bound	Minimal requirement for κ bits of security
$q_{\mathrm{RO}} \cdot \delta$	$\delta \leq 2^{-\kappa}$
$\frac{q_{\mathrm{RO}}}{\lvert\mathcal{M}\rvert}$	$\lvert\mathcal{M}\rvert \geq 2^{\kappa}$

TABLE 5.28: Recommended bounds for security of $\mathrm{KEM}^{\perp}$.

If the error vector used for encryption in the original McEliece PKC is not of a too high weight (such that error correction is actually possible) as above, then δ could be chosen to be 0, because correct decryption would always be possible. If not 0, then at least a number $\epsilon \in \mathbb{R}$ that is arbitrarily close to it, $0 < \epsilon \ll 1$. In this case, for the first row in table 5.28 one has

$$\frac{t(B)}{Adv(B)} \geq \frac{q_{\mathrm{RO}}}{q_{\mathrm{RO}} \cdot \delta} = \frac{1}{\delta} \geq 2^{\kappa}.$$

This is indeed true, since δ can just be chosen as $2^{-\kappa}$ (avoiding $\delta = 0$ — even though it is entirely possible — so as to make sure that no expression is undefined. Notice however that as δ approaches 0, $\frac{1}{\delta}$ approaches infinity, making completely sure that is is above the bound of 2^{κ}).

As for the the other bound in table 5.28, all that is needed is a large enough plaintext space. This can safely be assumed, since eq. (3.12) implies that k can be chosen to have some arbitrarily large lower bound. In particular κ bits of security can be achieved as $\kappa = k$, since $|\mathcal{M}| = 2^k$ and thus it is possible to achieve at least $n - mt$ bits of security.

All of this was actually just to show that an arbitrary level of security can still be reached, after the application of the transforms as above and if t is chosen to be maximal, then for some fixed m, n can still be used as the sole security parameter – just as in the original McEliece PKC.

Chapter 6

How To Be Safe In The Quantum Random Oracle Model

In chapter 5 I laid out how one can transform the original McEliece PKC into a key encapsulation mechanism that is IND-CCA2. Due to recent developments in the research field of quantum computers, it has become more and more interesting to see, if a public key cryptosystem can be used against adversaries that has access to a quantum computer (so-called *quantum adversaries*). This chapter will then be all about how one might transform the original McEliece PKC into something that is secure against such adversaries. I note here, that if this is indeed possible, then the transformed cryptosystem will especially also be safe against attacks from any classical adversaries, since it is safe to assume that such an adversary will also have access to a classical computer and that a quantum computer will also be able to do all the calculations that a classical computer can do and then some. If this was not the case, then said quantum computer would not be of much use.

Here I will as usual give some introduction to the topics that I will be talking about in section 6.1. In section 6.2 I will follow their analysis of the T-tranformation from section 5.6 in the case of a quantum adversary. As for section 6.3 I will show how to build a new key encapsulation mechanism that is to replace the one from section 5.7 in order to achieve CCA2 security in the face of a quantum adversary. Finally I will show how the new transformed McEliece PKC is in section 6.4.

6.1 A Brief Introduction To Quantum Computation

The purpose of this section is to develop the framework about which some later proofs will be constructed. Therefore I will follow [22] in giving a brief introduction to the necessary ideas and concepts, but not give a full introduction to the field of quantum computation. For a full introduction, I will suggest that the reader reads a proper textbook on the subject.

6.1.1 Qubits

Just as normal computers, quantum computers have bits. They do differ however and therefore a specific name for these are needed. In the case of a quantum computer, a bit will be called a *qubit*. The standard notation for such a qubit is $|b\rangle$. This is the *ket*-notation for a quantum superposition b:

$$|b\rangle = \begin{pmatrix} b_0 \\ \vdots \\ b_{n-1} \end{pmatrix} \in \mathbb{C}^n.$$

For the purposes of this chapter, all that is needed however are bits, so the vector $|b\rangle \in \mathbb{C}^2$ will only have two entries and itself be a linear combination $|b\rangle = \alpha|0\rangle + \beta|1\rangle$ of the two basis vectors $|0\rangle$ and $|1\rangle$ with probability amplitudes $\alpha, \beta \in \mathbb{C}$ such that $|\alpha|^2 + |\beta|^2 = 1$. When this is the case, the quantum particle that gives rise to $|b\rangle$ is bound to be found in some state and thus it is said that the *superposition* that it is in is *normalised.*

Notice that classical computers only deal with bits, that are either 0 or 1 whilst in the case of a quantum computer, a qubit can be some part of both. This is the essence of the superposition notion of the vector $|b\rangle$. Since 0 and 1 form the basis of vectors made with binary numbers, $\{|0\rangle, |1\rangle\}$ is the basis of qubits. This basis is called the *standard orthonormal computational basis.*

Notice also that classical bits can be mapped to be qubits:

$$b \to 1 \cdot |b\rangle + 0 \cdot |1 - b\rangle,$$

where the subtraction is done modulo 2.

6.1.2 Quantum Registers

Suppose you have a collection of qubits $\{|b_i\rangle\}$. This collection will constitute a linear combination of all of them:

$$\left\{ \sum_{(b_0,\cdots,b_{n-1})\in\{0,1\}^n} \alpha_{b_0\cdots b_{n-1}} \cdot |b_0 \cdots b_{n-1}\rangle \,\middle|\, \alpha_{b_0,\cdots,b_{n-1}} \in \mathbb{C}^n. \right.$$

This is a superposition and so it must still be normalised:

$$\sum_{(b_0,\cdots,b_{n-1})\in\{0,1\}^n} \left|\alpha_{b_0\cdots b_{n-1}}\right|^2 = 1.$$

Just as in the one-dimensional case, the basis $\{|b_0 \cdots b_{n-1}\rangle\} \mid (b_0 \cdots b_{n-1}) \in \{0,1\}^n$ will be called the *standard orthonormal computational basis.*

6.1.3 Measurements

Just as any quantum mechanical particle, a qubit can be measured. For qubits, the measurement will be done in the standard orthonormal computational basis. When measuring, the superposition will *collapse* into some specific state. This means that when measuring a qubit $|b\rangle = \alpha \cdot |0\rangle + \beta \cdot |1\rangle$, the result of the measurement will be $|0\rangle$ with probability $|\alpha|^2$ and $|1\rangle$ with probability $|\beta|^2$. Measuring a quantum register $\sum_{(b_0,\cdots,b_{n-1})\in\{0,1\}^n} \alpha_{b_0\cdots b_{n-1}} \cdot |b_0 \cdots b_{n-1}\rangle$ however will provide the measurement $|b_0 \cdots b_{n-1}\rangle$ with probability $\left|\alpha_{b_0\cdots b_{n-1}}\right|^2$.

The term “collapse” is used, because during measurement, the amplitudes of the superposition of a single qubit collapses into one of the combinations $\{\pm(1,0), \pm(0,1)\}$. In the n-dimensional case, this means that all of the amplitudes are switched into 0, except for the qubit that belongs to the exact state that is returned by the measurement, which will be switched to 1.

In code based games, performing such a measurement will be denoted by the function Measure $(\bullet)$.

6.1.4 Quantum Oracles And Quantum Adversaries

As for the standard random oracle model, some quantum oracles are needed. Let

$$O : \{0,1\}^n \to \{0,1\}^m$$

and $\mathbf{x} \in \{0,1\}^n$, $\mathbf{y} \in \{0,1\}^m$. In [2, 10] quantum oracles are then viewed as a mapping:

$$|\mathbf{x}\rangle|\mathbf{y}\rangle \mapsto |\mathbf{x}\rangle|\mathbf{y} \oplus O(\mathbf{x})\rangle$$

and so they will be here.

Let U be a unitary operation and when dealing with functions, let $\bullet \circ \bullet$ also denote the operation of function composition. Consider the quantum adversary A with access to the quantum random oracle O. This is done using the sequence $U \circ O$ and writing $A^{|O\rangle}$ denotes that the oracles are *quantum-accessible* by A (meaning that A can send qubits to an oracle). Note that this differs from normal random oracles, who can only process classical bits.

6.1.5 Quantum Random Oracle Model

Security games will be considered in the *quantum random oracle model* (*QROM*) almost in the same way as in the random oracle model. The difference is that the adversaries are considered to have quantum access to the random oracles that are involved. At the same time the adversaries will still have classical access to all other oracles, that are not quantum oracles.

In 2012 Zhandry proved that if $A^{|f\rangle}$ issues at most q quantum queries to $|f\rangle$, then it cannot distinguish between a random function

$$f : \{0,1\}^m \to \{0,1\}^n$$

and a $2q$-wise independent function [47]. This means that quantum random oracles can be viewed as polynomials of sufficiently large degree. So the quantum random oracle $|h\rangle$ can be viewed as an oracle that evaluates a random polynomial of degree $2q$ over the finite field $\mathbb{F}_{2^n}$.

6.1.6 Correctness Of Public Key Cryptosystems In The QROM

Like in the classical case, a notion of correctness of public key cryptosystems will be needed in the quantum random oracle model. Let $\Pi = \Pi_1^O$ be a public key cryptosystem that is defined in relation to some random oracle $|O_1\rangle$. The correctness bound will again depend on the number of queries q_{O_1} to $|O_1\rangle$.

Suppose some quantum adversary A that has access to the quantum random oracle $|O_1\rangle$ plays the following game on $\Pi = (G, E, D)$ with an oracle O.

COR − QRO Input to both A and O the security parameter k of Π.

1. O runs $G(k)$ to generate (pk, sk) and gives this tuple to A.
2. $A^{|O_1\rangle}$ chooses a message $\mathbf{m}$ and gives to O.
3. O encrypts $\mathbf{m}$, by computing $\mathbf{c} = E_{pk}(\mathbf{m})$.
4. O checks if $D_{sk}(\mathbf{c}) = \mathbf{m}$ and returns 1 if this is not true and 0 if this is true.

Notice first that this differs from the game COR − RO in the last step, whereas in the classical case, 1 is returned if the check passes and 0 otherwise, whilst it is turned around in the quantum random oracle case. Now assume that A makes at most q_{O_1} quantum queries to the quantum random oracle $|O_1\rangle$. If

$$P\left(\mathrm{COR} - \mathrm{QRO}_{\Pi}^{A} \Rightarrow 1\right) \leq \delta\left(q_{O_1}\right)$$

then Π is said to be $\delta\left(q_{O_1}\right)$*-correct.*

6.1.7 Algorithmic One-Way To Hiding

Let $[\bullet]$ denote the closed interval from 0 to some number. That is $[\lambda] = [0, \lambda]$.

Suppose you have an algorithm A that has access to the quantum random oracle $|O\rangle$ and possibly other oracles as well. For such an algorithm and such an oracle, an extractor algorithm that returns a measurement x' of a randomly chosen query to $|O\rangle$ will be defined as in algorithm 6.1.

Algorithm 6.1 $EXT\,[A, |O\rangle]\,(inp)$

```
$i \xleftarrow{\$} [q_O]$
Run $A^{|O\rangle}(inp)$ until the $i$'th query $|\hat{\mathbf{x}}\rangle$ to $|O\rangle$
if $i >$ number of queries to $|O\rangle$ then
    Return $\perp$
else
    $\mathbf{x}' \leftarrow \mathrm{Measure}(|\hat{\mathbf{x}}\rangle)$
    Return $\mathbf{x}'$
end if
```

The following lemma about this extractor algorithm is made in [22] and is an algorithmic adaptation of Lemma 5 in [44].

Lemma 6.2 (Algorithmic One-way To Hiding) *Let* $|O\rangle : \{0,1\}^n \to \{0,1\}^m$ *be a quantum random oracle and let* A *be a quantum algorithm issuing at most* q_O *queries to* $|O\rangle$ *such that it on input* $\mathbf{x} \in \{0,1\}^n$, $\mathbf{y} \in \{0,1\}^m$ *outputs either* 0 *or* 1. *Let* F *be any probabilistic algorithm that does not make any queries to* $|O\rangle$ *and input bit strings in* $\{0,1\}^{n+m}$. *Then*

$$\left|P\left(1 = A^{|O\rangle}(inp) \,\middle|\, \mathbf{x} \xleftarrow{\$} \{0,1\}^n,\ inp = F(\mathbf{x}, O(\mathbf{x}))\right) - P\left(1 = A^{|O\rangle}(inp) \,\middle|\, (\mathbf{x},\mathbf{y}) \xleftarrow{\$} \{0,1\}^{n+m},\ inp = F(\mathbf{x},\mathbf{y})\right)\right|$$

$$\leq 2q_O \cdot \sqrt{P\left(\mathbf{x} = EXT\,[A, |O\rangle]\,(inp) \,\middle|\, (\mathbf{x},\mathbf{y}) \xleftarrow{\$} \{0,1\}^{n+m},\ inp = F(\mathbf{x},\mathbf{y})\right)}.$$

6.1.8 Generic Quantum Search

A *Bernoulli distribution* is the discrete probability distribution that describes how a random variable can take the value 1 with probability λ or the value 0 with probability $1 - \lambda$, for $\lambda \in [0, 1]$. For such a value $\lambda \in [0, 1]$, denote by B_λ such a Bernoulli distribution.

If the value of a bit b can be said to be coming from a Bernoulli distribution B_λ, then the above means that $P(b = 1) = \lambda$.

Below is seen a problem from [23, 46], that will lead on to what is needed in this section.

Problem 6.3 (Generic Quantum Search Problem (GSP)) *Let B_λ be a Bernoulli distribution and let X be some finite set with elements $x \in X$. Now define*

$$\mathrm{F} \ : \ X \to \{0, 1\}$$

and suppose you are given quantum access to an oracle that upon input x returns $\mathrm{F}(x)$*. Find some x such that* $\mathrm{F}(x) = 1$ *and for each x,* $\mathrm{F}(x)$ *is distributed according to B_λ.*

Here a slight variation of problem 6.3 is needed. This version is given in [25] and is essentially the same, except for the fact that the Bernoulli parameter $\lambda(x)$ may depend on the input x, but is upper bounded by some global $\lambda \in [0, 1]$. This is called the *generic quantum search problem with bounded probabilities* (*GSPB*) and is modelled as a code based game in algorithm 6.4.

Algorithm 6.4 GSPB_λ

```
$(\lambda(x)) \mid x \in X \leftarrow A_1$
**if** $\exists x \in X \ : \ \lambda(x) > \lambda$ **then**
    **Return** 0
**end if**
**for all** $x \in X$ **do**
    $\mathrm{F}(x) \leftarrow B_{\lambda(x)}$
**end for**
$x \leftarrow A_2^{|F(\cdot)\rangle}$
**Return** $\mathrm{F}(x)$
```

The convention here will be that this game is denoted by GSPB_λ, and GSPB_λ^A whenever A plays the game. Just as for any of the other games seen so far, if $\mathrm{GSPB}_\lambda^A \Rightarrow 1$ then A is said to win the game (is able to output a $\mathrm{F}(x) = 1$ and not $\mathrm{F}(x) = 0$). The following lemma about this game comes from [25].

Lemma 6.5 *Let $\lambda \in \{0, 1\}$ and let A denote any quantum adversary with quantum access to the oracle* $|\mathrm{F}\rangle$*. Suppose A plays the game* GSPB_λ *(from algorithm 6.4) and issues at most q quantum queries to* $|\mathrm{F}\rangle$*, then*

$$P\left(\mathrm{GSPB}_\lambda^A \Rightarrow 1\right) \leq 8 \cdot \lambda \cdot (q + 1)^2 .$$

6.2 Removing The Randomness In The QROM

Recall the T-transformation from section 5.6. In section 5.6.1 it was shown that the correctness of the underlying cryptosystem that the transformation is done upon is changed by at most a factor of q_h (equalling the total number of queries that an adversary at most poses to the random oracle h in the game $\mathrm{COR}-\mathrm{RO}$). This was because the randomness of the underlying cryptosystem was removed and replaced by a hash function. In the QROM however, one does not deal with the game $\mathrm{COR}-\mathrm{RO}$, but instead the game $\mathrm{COR}-\mathrm{QRO}$. This means that the bound on correctness is a bit different than in the classical case. Following [22], I will lay out how to arrive at a lemma about exactly that.

Let $\Pi = (G, E, D)$ and $\Pi_1 = T(\Pi)$ be as in section 5.6 with Π being δ-correct, A be a quantum adversary playing the game $\mathrm{COR}-\mathrm{QRO}$ with (pk, sk) being some fixed key pair of Π_1. Furthermore suppose that A is given quantum access to some oracle $h(\bullet)$ to which it sends at most q_h queries. If $\mathcal{M}$ is the message space of Π_1, then $\mathbf{m} \in \mathcal{M}$ is a valid message of Π_1. Let also $\mathcal{R}$ be the randomness space being defined by pk. Denote by $\mathcal{R}_{\mathrm{bad}}$ the set of randomness that leads to an encryption of $\mathbf{m}$ using Π that when decrypted does not give back the message $\mathbf{m}$. That is

$$\mathcal{R}_{\mathrm{bad}}(pk, sk, \mathbf{m}) = \left\{\mathbf{r} \in \mathcal{R} \,\middle|\, D_{sk}\left(E_{pk}(\mathbf{m}; \mathbf{r})\right) \neq \mathbf{m}\right\}.$$

Now define the fraction of such bad randomness, $\delta(pk, sk, \mathbf{m})$, as

$$\delta(pk, sk, \mathbf{m}) = \frac{|\mathcal{R}_{bad}(pk, sk, \mathbf{m})|}{|\mathcal{R}|}$$

along with

$$\delta(pk, sk) = \max_{\mathbf{m} \in \mathcal{M}} \delta(pk, sk, \mathbf{m}).$$

Note that this means that if one takes the expectation over the key pairs that G can output, then $\delta = \mathrm{E}(\delta(pk, sk))$.

What is needed is an upper bound on $P\left(\mathrm{COR}-\mathrm{QRO}^A \Rightarrow 1\right)$. So let $F(\bullet)$ be as in problem 6.3, $f(\bullet)$ be some $2q_h$-wise independent hash function that A cannot access, but the random oracle $h(\bullet)$ has internally and lastly let also $\mathrm{Sample}(\bullet)$ be a probabilistic algorithm that on input a set Y returns an element $y \in Y$ uniformly at random. If however the function $\mathrm{Sample}(\bullet; \bullet)$ is used, then the second argument will be the explicit randomness that is to be used. That is $\mathrm{Sample}(Y; f(\mathbf{m}))$ denotes the deterministic execution of $\mathrm{Sample}(Y)$, where $f(\mathbf{m})$ is the randomness to be used. Now define $h(\bullet)$ as in algorithm 6.8 and yet another quantum adversary $B = (B_1, B_2)$ as in algorithms 6.6 and 6.7. Suppose further that B has a running time about that of A.

Algorithm 6.6 B_1

```
$(pk, sk) \leftarrow G$
**for all** $\mathbf{m} \in \mathcal{M}$ **do**
    $\lambda(\mathbf{m}) = \delta(pk, sk, \mathbf{m})$
**end for**
**Return** $\lambda(\mathbf{m} \in \mathcal{M})$
```

Algorithm 6.7 $B_2^{|F(\bullet)\rangle}$

```
Pick a $2q_O$-wise hash $f$
$\mathbf{m}'' \leftarrow A^{|h(\bullet)\rangle}(pk, sk)$
**Return** $\mathbf{m}''$
```

Algorithm 6.8 $h(\mathbf{m})$

```
**if** $F(\mathbf{m}) = 0$ **then**
    $h(\mathbf{m}) = \text{Sample}(\mathcal{R} \setminus \mathcal{R}_{\text{bad}}(pk, sk, \mathbf{m}); f(\mathbf{m}))$
**else**
    $h(\mathbf{m}) = \text{Sample}(\mathcal{R}_{\text{bad}}(pk, sk, \mathbf{m}); f(\mathbf{m}))$
**end if**
**Return** $h(\mathbf{m})$
```

B is an adversary against the game GSPB_λ. It starts off by using G to generate a key pair (pk, sk) and then goes on to find the Bernoulli parameters $\lambda(\mathbf{m}) = \delta(pk, sk, \mathbf{m})$ for each $\mathbf{m} \in \mathcal{M}$. Notice that these are bounded by

$$\lambda = \delta(pk, sk) = \max_{\mathbf{m}\in\mathcal{M}} P\left(D_{sk}\left(E_{pk}(\mathbf{m})\right) \neq \mathbf{m}\right).$$

Supposing (pk, sk) is kept fixed and $\mathbf{m} \in \mathcal{M}$ then, in the game GSPB_λ, the random variable $F(\mathbf{m})$ is distributed according to $B_{\lambda(\mathbf{m})} = B_{\delta(pk,sk,\mathbf{m})}$ for each such $\mathbf{m}$. Notice that the returned value from $h(\bullet)$ is distributed uniformly at random in $\mathcal{R}$ and so h is a quantum random oracle.

Now A will win the game $\text{COR} - \text{QRO}_{\Pi_1}$ if and only if it returns the message $\mathbf{m}$ such that $h(\mathbf{m}) \in \mathcal{R}_{\text{bad}}(pk, sk, \mathbf{m})$. This is equivalent to $F(\mathbf{m}) = 1$. If problem 6.3 is interpreted as a game (call it GSP_λ), then this is equivalent to the case in which B wins the game GSP_λ. So by using lemma 6.5, it can be seen that

$$P\left(\text{COR} - \text{QRO}_{\Pi_1}^A \Rightarrow 1 \,\middle|\, (pk, sk)\right) \leq P\left(\text{GSP}_{\delta(pk,sk)}^B \Rightarrow 1\right) \leq 8 \cdot \delta(pk, sk) \cdot (q_h + 1)^2.$$

Averaging over all the possible key pairs (pk, sk) that G can output, this gives

$$\delta_1(q_h) = P\left(\text{COR} - \text{QRO}^A \Rightarrow 1\right) \leq 8 \cdot \delta \cdot (q_h + 1)^2$$

proving the following.

Lemma 6.9 *Let* Π *be a* δ*-correct public key cryptosystem,* h *be a hash function and let* $\Pi_1 = T(\Pi, h)$*. Then* Π_1 *is* δ_1*-correct where* $\delta_1 = \delta_1(q_h) \leq 8 \cdot (q_h + 1)^2 \cdot \delta$*.*

Now that correctness has been taken care of, it is time to move on to the security reduction. The thing is that the security reduction from OW − CPA to OW − PCA also works in the quantum random oracle model, which I will write about here (again following [22] – loosely being based upon [43]).

So let A, B and h be as just before. A is an OW − CPA quantum adversary and B is an OW − PCA quantum adversary that runs in time about that of A. Assume that B has (classical) access to a plaintext checking oracle $\text{PCO}(\bullet, \bullet)$ and given that it simulates A also has quantum access to $|h(\bullet)\rangle$ (being modelled here as a random $2q_h$-wise independent hash function). Assume further that B makes at most q_h queries to $|h\rangle$ and q_P (classical) queries to PCO.

Algorithm 6.10 Games G_7-G_9 and O

```
$(pk, sk) \leftarrow G$
$\mathbf{m} \xleftarrow{\$} \mathcal{M}$
$\mathbf{r} = h(\mathbf{m})$                                          $\triangleright G_7$-$G_8$
$\mathbf{r} \xleftarrow{\$} \mathcal{R}$                                            $\triangleright G_9, O$
$\mathbf{c} = E_{pk}(\mathbf{m}; \mathbf{r})$
$\mathbf{m}' \leftarrow B^{|h(\cdot)\rangle, \mathrm{PCO}(\cdot,\cdot)}(pk, \mathbf{c})$                  $\triangleright G_8$-$G_9$
$\mathbf{m}' \leftarrow EXT\left[B^{\mathrm{PCO}(\cdot,\cdot)}, |h(\cdot)\rangle\right](pk, \mathbf{c})$         $\triangleright O$
**Return** $[\![\mathbf{m}' = \mathbf{m}]\!]$
```

Algorithm 6.11 PCO $\left(\mathbf{m}'' \in \mathcal{M}, \mathbf{c}'\right)$

```
$\mathbf{m}' = D_1\left(sk, \mathbf{c}'\right)$                                   $\triangleright G_7$
**Return** $[\![\mathbf{m}' = \mathbf{m}'']\!] \wedge [\![E_{pk}\left(\mathbf{m}'; h\left(\mathbf{m}'\right)\right) = \mathbf{c}']\!]$     $\triangleright G_7$
**Return** $[\![E_{pk}\left(\mathbf{m}''; h\left(\mathbf{m}''\right)\right) = \mathbf{c}']\!]$              $\triangleright G_8, G_9, O$
```

The PCO-oracle is defined as in algorithm 6.11 and the oracle h is again defined as in algorithm 6.8. Consider then the games G_7 to G_9 and O from algorithm 6.10.

The game G_7 is just the original OW − PCA game and so

$$P\left(G_7^B \Rightarrow 1\right) = Adv_{\Pi_1}^{\mathrm{OW-PCA}}(B).$$

In game G_8 the plaintext checking oracle is replaced by a simulation, that no longer makes use of the secret key. Note that G_8 proceeds just as G_7, up until the time, that B submits a problematic query to PCO. Call this event BADR. Now since if BADR does not occur, G_7 and G_8 proceed identically, this means that

$$\left|P\left(G_8^B \Rightarrow 1\right) - P\left(G_7^B \Rightarrow 1\right)\right| \leq P(\mathrm{BADR}).$$

If however B does make a problematic query, then there must exist some adversary F that can perfectly simulate G_7 and G_8 and win the game $\mathrm{COR} - \mathrm{QRO}_{\Pi_1}$. From lemma 6.9 it then follows that

$$P(\mathrm{BADR}) \leq P\left(\mathrm{COR} - \mathrm{QRO}_{\Pi_1}^F \Rightarrow 1\right) \leq 8 \cdot (q_h + q_P + 1)^2 \cdot \delta.$$

In the game G_9 $\mathbf{r} = h(\mathbf{m})$ is replaced by a uniformly random $\mathbf{r}$.

Algorithm 6.12 F $(\mathbf{m}, \mathbf{r})$

```
$(pk, sk) \leftarrow G$
$\mathbf{c} = E_{pk}(\mathbf{m}; \mathbf{r})$
$inp = (pk, \mathbf{c})$
**Return** $inp$
```

Let the extractor algorithm of game O be defined as in algorithm 6.1 and let $x = \mathbf{m}$ and $y = \mathbf{r}$. Considering algorithm 6.12 and applying lemma 6.2 it follows that

$$\left|P\left(G_9^B \Rightarrow 1\right) - P\left(G_8^B \Rightarrow 1\right)\right| \leq 2 \cdot q_h \cdot \sqrt{P\left(O^B \Rightarrow 1\right)}.$$

Algorithm 6.13 $C(pk, \mathbf{c})$

```
$\mathbf{m}' \leftarrow B^{|h(\cdot)\rangle, \mathrm{PCO}(\cdot,\cdot)}(pk, \mathbf{c})$
**Return** $\mathbf{m}'$
```

Now let C be an adversary against the OW − CPA security of Π. It is rather trivially constructed in algorithm 6.13. It simulates game G_9 for B and outputs $\mathbf{m}' = \mathbf{m}$ if B wins in game G_9. This means that

$$P\left(G_9^B \Rightarrow 1\right) = Adv_{\Pi}^{\mathrm{OW-CPA}}(C) \leq \sqrt{Adv_{\Pi}^{\mathrm{OW-CPA}}(C)}.$$

Algorithm 6.14 $\mathrm{D}(pk, \mathbf{c})$

```
$\mathbf{m}' \leftarrow EXT\left[B^{\mathrm{PCO}(\cdot,\cdot)}, |h(\cdot)\rangle\right](pk, \mathbf{c})$
**Return** $\mathbf{m}'$
```

In algorithm 6.14 is constructed another trivial adversary against the OW − CPA security of Π. It is called D and Simulates game O for B, thus obtaining

$$P\left(O^B \Rightarrow 1\right) = Adv_{\Pi}^{\mathrm{OW-CPA}}(\mathrm{D}).$$

Now collecting all the probabilities deduced from the games in this section and combining C and D into a single OW − CPA adversary A yields

$$Adv_{\Pi_1}^{\mathrm{OW-PCA}}(B) \leq 8 \cdot (q_h + q_P + 1)^2 \cdot \delta + (1 + q_h) \cdot \sqrt{Adv_{\Pi}^{\mathrm{OW-CPA}}(A)}. \qquad (6.15)$$

This proves the following theorem.

Theorem 6.16 *Let* Π *be a public key that is* OW − CPA *in the random oracle model, h be a hash function and let* $\Pi_1 = T(\Pi, h)$*. Then* Π_1 *is* OW − PCA *in the quantum random oracle model.*

6.3 A New Second Transformation

In the last section, it was shown that the T-transformation from section 5.6 also can provide OW − PCA security in the quantum random oracle model. Just as in the previous chapter, what is actually wanted is IND-CCA2 security but this time in the quantum random oracle model. Just like in section 5.7 this section will introduce a second transformation, that provides a security reduction from IND-CCA2 to OW − PCA. There are some differences here though, that will make sure, that the reduction also works in the QROM. Unlike before this transformation has explicit rejection and requires that the cryptosystem that it is applied upon is deterministic.

Let $\Pi_1 = (G_1, E_1, D_1)$ be a deterministic and rigid public key cryptosystem with message space $\mathcal{M}$ (where all valid messages have length k). Let also g and g' be hash functions

$$\begin{aligned} g &: \{0,1\}^* \to \{0,1\}^k \\ g' &: \{0,1\}^k \to \{0,1\}^k. \end{aligned}$$

The following transformation will be called $QU_m^{\perp}$, is due to [22] and yields a key encapsulation mechanism with explicit rejection

$$\mathrm{QKEM}_m^{\perp} = QU_m^{\perp}\left(\Pi_1, g, g'\right) = \left(\mathrm{QGen} = G_1, \mathrm{QEncaps}_m, \mathrm{QDecaps}_m^{\perp}\right).$$

The details of $\mathrm{QKEM}_m^{\perp}$ are given here.

QGen: Use G_1 to generate a key pair (pk, sk).

Public Key: pk.

Private Key: sk.

QEncaps$_m$: To perform the key encapsulation, let $\mathbf{m} \in \mathcal{M}$ be any valid message and compute

$$\mathbf{c} = E_1\left(pk, \mathbf{m}\right).$$

Then compute

$$\mathbf{d} = g'\left(\mathbf{m}\right)$$

and

$$K = g\left(\mathbf{m}\right).$$

Finally return $(K, \mathbf{c}, \mathbf{d})$.

QDecaps$_m^{\perp}$: Upon receiving $(\mathbf{c}, \mathbf{d})$, compute

$$\mathbf{m}' = D_1\left(sk, \mathbf{c}\right).$$

If $\mathbf{m}'$ is not a valid message or $g'\left(\mathbf{m}'\right) \neq \mathbf{d}$ then abort. Otherwise return $K = g\left(\mathbf{m}'\right)$.

6.3.1 Implying IND-CCA2 from OW − PCA With Correctness In The QROM

The discussion of security and correctness needs to be changed, when compared to the discussion from section 5.7.1. This is partly because the transformation is a bit different but also because it is no longer the random oracle model that is dealt with, but instead the quantum random oracle model. For this I follow [22] again (and again it is loosely based upon [43]).

Algorithm 6.17 Games G_{10}-G_{13}

```
$b \xleftarrow{\$} \{0,1\}$                                   $\triangleright G_{10}$-$G_{12}$
$(pk, sk) \leftarrow G_1$
$\mathbf{m} \xleftarrow{\$} \{0,1\}^k$
$\mathbf{c} = E_1(pk, \mathbf{m})$
$K_0 = g(\mathbf{m})$
$K_1 \xleftarrow{\$} \{0,1\}$
$\mathbf{d} = g'(\mathbf{m})$                                  $\triangleright G_{10}$-$G_{12}$
$K = K_b$                                          $\triangleright G_{10}$-$G_{12}$
$\mathbf{d} \xleftarrow{\$} \{0,1\}^k$                              $\triangleright G_{13}$
$K \xleftarrow{\$} \{0,1\}^k$                              $\triangleright G_{13}$
$b' \leftarrow B^{\text{QDecaps}_m^{\perp}, |g\rangle, |g'\rangle}(pk, (\mathbf{c}, \mathbf{d}), K)$
Return $[\![b' = b]\!]$                             $\triangleright G_{10}$-$G_{12}$
Return $b'$                                        $\triangleright G_{13}$
```

Algorithm 6.18 QDecaps$_m^{\perp}$ $\left((\mathbf{c}', \mathbf{d}') \neq (\mathbf{c}, \mathbf{d})\right)$ for G_{10}-G_{13}

```
if $\mathbf{c}' = \mathbf{c}$ then                                $\triangleright G_{11}$-$G_{13}$
    Return $\perp$                                  $\triangleright G_{11}$-$G_{13}$
end if                                             $\triangleright G_{11}$-$G_{13}$
$\mathbf{m}' = D_1(sk, \mathbf{c}')$
if $\mathbf{m}' = \mathbf{m}$ then                                $\triangleright G_{12}$-$G_{13}$
    Abort                                          $\triangleright G_{12}$-$G_{13}$
end if                                             $\triangleright G_{12}$-$G_{13}$
if $\mathbf{m} \neq \perp \wedge g'(\mathbf{m}') = \mathbf{d}'$ then
    Return $K' = g(\mathbf{m}')$
else
    Return $\perp$
end if
```

Start off by letting B be an adverary against the CCA2 security of QKEM$_m^{\perp}$. Assume that B issues at most q_D classical queries to QDecaps$_m^{\perp}$, at most q_g queries to $|g\rangle$ and at most $q_{g'}$ queries to $|g'\rangle$. Now consider the games G_{10} to G_{13} given in algorithms 6.17 and 6.18.

G_{10} is just the original CCA2 game, so

$$Adv^{\text{CCA2}}_{\text{QKEM}_m^{\perp}}(B) = \left| P\left(G_{10}^B \Rightarrow 1\right) - \frac{1}{2} \right|.$$

Now, in game G_{11}, QDecaps$_m^{\perp}$ is changed so that it always returns $\perp$ if it is queried on some ciphertext that has the form $(\mathbf{c}, \mathbf{d})$. Such a query is not allowed in G_{10} or G_{11}, so the focus can be limited to the case where $\mathbf{d}' \neq \mathbf{d}$. Let $\mathbf{m}' = D_1(sk, \mathbf{c})$ and if $\mathbf{m}' = \mathbf{m}$ and $\mathbf{d}' \neq \mathbf{d}$, then $g'(\mathbf{m}') = g'(\mathbf{m}) = \mathbf{d} \neq \mathbf{d}'$, so G_{10} will also return $\perp$. Now, G_{11} is only different from the game G_{10} if $\mathbf{m}' \neq \mathbf{m}$ and so this means that the change from G_{10} to G_{11} is only conceptual, unless $\mathbf{m}$ will result in a correctness error. This gives

$$\left|P\left(G_{11}^{B} \Rightarrow 1\right) - P\left(G_{10}^{B} \Rightarrow 1\right)\right| \leq \delta_1$$

giving rise to the following result.

Theorem 6.19 *Let* $\Pi_1 = (G_1, E_1, D_1)$ *be a deterministic and rigid public key cryptosystem and let* $\mathrm{QKEM}_m^{\perp} = QU_m^{\perp}\left(\Pi_1, g, g'\right)$*. If* Π_1 *is* δ_1*-correct, then so is* $\mathrm{QKEM}_m^{\perp}$*.*

Having taken care of correctness, do not yet forget the results of B playing games G_{10} and G_{11} as they will still be used going forward. Have a look at G_{12} for instance. Here $\mathrm{QDecaps}_m^{\perp}$ is changed slightly again, so as to abort, if it is queried on a ciphertext $(\mathbf{c}, \mathbf{d})$ with $\mathbf{c}' \neq \mathbf{c}$, whilst $D_1\left(sk, \mathbf{c}'\right) = \mathbf{m}$. Since Π_1 is rigid and $D_1\left(sk, \mathbf{c}'\right) \neq \perp$, it must be true that $\mathbf{c}' = E_1\left(pk, D_1\left(sk, \mathbf{c}'\right)\right) = E_1(pk, \mathbf{m}) = \mathbf{c}$. Now the probability that an abort will occur is 0 and so

$$P\left(G_{12}^{B} \Rightarrow 1\right) = P\left(G_{11}^{B} \Rightarrow 1\right).$$

Notice here that the line that aborts in G_{12} and G_{13} is only triggered if $\mathbf{m}' = \mathbf{m}$, so $\mathrm{QDecaps}_m^{\perp}$ cannot trigger a query to g or g' on $\mathbf{m}$.

As for game G_{13}, the tuple $(\mathbf{d}, K)$ is picked uniformly at random and there is no longer a check to see if B guessed the right bit at the end of the game. Given that

$$\begin{aligned} P\left(G_{12}^{B} \Rightarrow 1\right) &= \frac{1}{2}\left(P\left(G_{12}^{B} \Rightarrow 1 \mid b = 1\right) + P\left(G_{12}^{B} \Rightarrow 1 \mid b = 0\right)\right) \\ &= \frac{1}{2}\left(P\left(G_{12}^{B} \Rightarrow 1 \mid b = 1\right) - P\left(G_{12}^{B} \Rightarrow 0 \mid b = 0\right)\right) + \frac{1}{2}, \end{aligned}$$

then

$$\begin{aligned} \left|P\left(G_{12}^{B} \Rightarrow 1\right) - \frac{1}{2}\right| &= \frac{1}{2}\left|P\left(G_{12}^{B} \Rightarrow 1 \mid b = 1\right) - P\left(G_{12}^{B} \Rightarrow 0 \mid b = 0\right)\right| \\ &\leq \frac{1}{2}\left(\left|P\left(G_{12}^{B} \Rightarrow 1 \mid b = 1\right) - P\left(G_{13}^{B} \Rightarrow 1\right)\right| + \right. \\ &\qquad \left.\left|P\left(G_{12}^{B} \Rightarrow 0 \mid b = 0\right) - P\left(G_{13}^{B} \Rightarrow 1\right)\right|\right). \end{aligned}$$

Letting $F(\mathbf{m}, \mathbf{d}) = (\mathbf{m}, \mathbf{d})$ for $b = 1$ and $F(\mathbf{m}, \mathbf{d}, K) = (\mathbf{m}, \mathbf{d}, K)$ for $b = 0$, lemma 6.2 can be used to make an upper bound upon the last two terms in the above, so use the notation $\bullet \times \bullet$ to denote the combined oracle, that is a combination of two different oracles.

Algorithm 6.20 $\mathrm{D}_0^{|g\rangle, |g'\rangle}$

```
$(pk, sk) \leftarrow G_1$
$\mathbf{c} = E_1(pk, \mathbf{m})$
$b' \leftarrow B^{\mathrm{QDecaps}_m^{\perp}, |g\rangle, |g'\rangle}(pk, (\mathbf{c}, \mathbf{d}), K)$
**Return** $b'$
```

Algorithm 6.21 $\mathrm{D}_1^{|g\rangle,|g'\rangle}(\mathbf{m},\mathbf{d})$

```
$(pk, sk) \leftarrow G_1$
$\mathbf{c} = E_1(pk, \mathbf{m})$
$K \xleftarrow{\$} \{0,1\}^k$
$b' \leftarrow B^{\mathrm{QDecaps}_m^{\perp},|g\rangle,|g'\rangle}(pk, (\mathbf{c}, \mathbf{d}), K)$
**Return** $b'$
```

Algorithm 6.22 Game g_0

```
$(\mathbf{m}, \mathbf{d}, K) \xleftarrow{\$} \{0,1\}^{2k+m}$
$i \xleftarrow{\$} [q_{g'} + q_g + q_D]$
Run $\mathrm{D}_0^{|g\times g'\rangle}(\mathbf{m}, \mathbf{d}, K)$ until the $i$'th query $|\hat{m}\rangle$ to $|g \times g'\rangle$
**if** $i >$ number of queries to $|g \times g'\rangle$ **then**
    **Return** 0
**else**
    $\mathbf{m}' \leftarrow$ Measure $(|\hat{m}\rangle)$
    **Return** $[\![\mathbf{m}' = \mathbf{m}]\!]$
**end if**
```

Algorithm 6.23 Game g_1

```
$(\mathbf{m}, \mathbf{d}) \xleftarrow{\$} \{0,1\}^{k+m}$
$i \xleftarrow{\$} [q_{g'} + q_D]$
Run $\mathrm{D}_1^{|g\rangle,|g'\rangle}(\mathbf{m}, \mathbf{d})$ until the $i$'th query $|\hat{\mathbf{m}}\rangle$ to $|g'\rangle$
**if** $i >$ number of queries to $|g'\rangle$ **then**
    **Return** 0
**else**
    $\mathbf{m}' \leftarrow$ Measure $(|\hat{\mathbf{m}}\rangle)$
    **Return** $[\![\mathbf{m}' = \mathbf{m}]\!]$
**end if**
```

Consider the adversaries D_0 and D_1 from algorithms 6.20 and 6.21 playing the games g_0 and g_1 from algorithms 6.22 and 6.23.

First notice that D_1 will issue at most $q_{g'} + q_D$ queries to g'. If run on input $(\mathbf{m}, \mathbf{d} = g'(\mathbf{m}))$ then D_1 will perfectly simulate game G_{12} for the bit $b = 1$. If however D_1 is run on a uniformly random input $(\mathbf{m}, \mathbf{d}) \xleftarrow{\$} \{0,1\}^{k+m}$, then D_1 perfectly simulates the game G_{13} for bit $b = 1$. This means that

$$\left|P\left(G_{12}^B \Rightarrow 1 \mid b = 1\right) - P\left(G_{13}^B \Rightarrow 1\right)\right| =$$

$$\left|P\left(\mathbf{m} \xleftarrow{\$} \{0,1\}^k,\ \mathbf{d} = g'(\mathbf{m}),\ b \leftarrow \mathrm{D}_1(\mathbf{m}, \mathbf{d})\ :\ b = 1\right) - \right.$$

$$\left. P\left((\mathbf{m}, \mathbf{d}) \xleftarrow{\$} \{0,1\}^{k+m},\ b \leftarrow \mathrm{D}_1(\mathbf{m}, \mathbf{d})\ :\ b = 1\right)\right|$$

$$\leq 2\left(q_{g'} + q_D\right) \cdot \sqrt{P\left(g_1^{\mathrm{D}_1} \Rightarrow 1\right)}.$$

Notice here that D_0 will make $q_{g'} + q_g + q_D$ queries to $g' \times g$. If run on input $\left(\mathbf{m}, (\mathbf{d}, K) = g' \times g\left(\mathbf{m}\right)\right)$ it then perfectly simulates the game G_{12} for bit $b = 0$. If it is run on the uniformly random input $(\mathbf{m}, \mathbf{d}, K) \xleftarrow{\$} \{0,1\}^{2k+m}$ instead, it then perfectly simulates the game G_{13}. So

$$\left|P\left(G_{12}^{B} \Rightarrow 0 \,\middle|\, b = 0\right) - P\left(G_{13}^{B} \Rightarrow 1\right)\right| \leq 2\left(q_{g'} + q_g + q_D\right) \cdot \sqrt{P\left(g_0^{\mathrm{D}_0} \Rightarrow 1\right)}.$$

This just established the bound

$$\left|P\left(G_{12}^{B} \Rightarrow 1\right) - \frac{1}{2}\right| \leq \left(q_{g'} + q_D\right) \cdot \sqrt{P\left(g_1^{\mathrm{D}_1} \Rightarrow 1\right)} + \left(q_{g'} + q_g + q_D\right) \cdot \sqrt{P\left(g_0^{\mathrm{D}_0} \Rightarrow 1\right)}.$$

Knowing that $b \in \{0,1\}$, one can then say that an upper bound on $P\left(g_b^{\mathrm{D}_b} \Rightarrow 1\right)$ is needed.

Now let Roots $(\bullet)$ be the function that returns the roots of a provided polynomial.

Algorithm 6.24 Games $G_{14,0}$ and $G_{15,0}$

```
$(\mathbf{m}, \mathbf{d}, K) \xleftarrow{\$} \{0,1\}^{2k+m}$
$i \xleftarrow{\$} [q_{g'} + q_g]$
$(pk, sk) \leftarrow G_1$
$\mathbf{c} = E_1(pk, \mathbf{m})$
Run $B^{\mathrm{QDecaps}_m^{\perp}, |g\rangle, |g'\rangle}(pk, (\mathbf{c}, \mathbf{d}), K)$ until the $i$'th query $|\hat{\mathbf{m}}\rangle$ to $|g \times g'\rangle$
**if** $i >$ number of queries to $|g \times g'\rangle$ **then**
    **Return** 0
**else**
    $\mathbf{m}' \leftarrow$ Measure $(|\hat{\mathbf{m}}\rangle)$
    **Return** $[\![\mathbf{m}' = \mathbf{m}]\!]$
**end if**
```

Algorithm 6.25 Games $G_{14,1}$ and $G_{15,1}$

```
$(\mathbf{m}, \mathbf{d}, K) \xleftarrow{\$} \{0,1\}^{2k+m}$
$i \xleftarrow{\$} [q_{g'}]$
$(pk, sk) \leftarrow G_1$
$\mathbf{c} = E_1(pk, \mathbf{m})$
Run $B^{\mathrm{QDecaps}_m^{\perp}, |g\rangle, |g'\rangle}(pk, (\mathbf{c}, \mathbf{d}), K)$ until the $i$'th query $|\hat{\mathbf{m}}\rangle$ to $|g'\rangle$
**if** $i >$ number of queries to $|g'\rangle$ **then**
    **Return** 0
**else**
    $\mathbf{m}' \leftarrow$ Measure $(|\hat{\mathbf{m}}\rangle)$
    **Return** $[\![\mathbf{m}' = \mathbf{m}]\!]$
**end if**
```

Algorithm 6.26 $\text{QDecaps}_m^{\perp}\left(\left(\mathbf{c}',\mathbf{d}'\right)\neq(\mathbf{c},\mathbf{d})\right)$ for $G_{14,b}$

```
if $\mathbf{c}' = \mathbf{c}$ then
    Return $\perp$
end if
$\mathbf{m}' = D_1\left(sk, \mathbf{c}'\right)$
if $\mathbf{m}' = \mathbf{m}$ then
    Abort
end if
if $\mathbf{m}' \neq \perp \wedge g'\left(\mathbf{m}'\right) = \mathbf{d}'$ then
    Return $K' = g\left(\mathbf{m}'\right)$
else
    Return $\perp$
end if
```

Algorithm 6.27 $\text{QDecaps}_m^{\perp}\left(\left(\mathbf{c}',\mathbf{d}'\right)\neq(\mathbf{c},\mathbf{d})\right)$ for $G_{15,b}$

```
if $\mathbf{c}' = \mathbf{c}$ then
    Return $\perp$
end if
$\mathbf{m}' = D_1\left(sk, \mathbf{c}'\right)$
if $\mathbf{m}' = \mathbf{m}$ then
    Abort
end if
if $\exists \mathbf{m}'' \in \text{Roots}\left(g'(X) - \mathbf{d}'\right) \;:\; D_1(sk, \mathbf{c}) = \mathbf{m}''$ then
    Return $K' = g\left(\mathbf{m}''\right)$
else
    Return $\perp$
end if
```

Consider games $G_{14,b}$ and $G_{15,b}$ from algorithms 6.24 to 6.27. These will be used to establish the desired bound upon $P\left(g_b^{\mathrm{D}_b} \Rightarrow 1\right)$.

The games $G_{14,b}$ are just reformulations of the games g_b that no longer uses the helper adversaries D_b. In g_b a query to $g' \times g$ $\left(\text{or } g'\right)$ is picked uniformly at random amongst the queries that D_b issues – including all of the queries that arise implicitly from B when querying $\text{QDecaps}_m^{\perp}$. Instead $G_{14,b}$ excludes the implicit queries and only picks an explicit query that B issued directly to $g' \times g$ $\left(\text{or } g'\right)$ uniformly at random.

Notice that D_b will reject queries to $\text{QDecaps}_m^{\perp}$ of the form $(\mathbf{c},\mathbf{d})$ and aborts if $\text{QDecaps}_m^{\perp}$ is queried on a ciphertext $\left(\mathbf{c}',\mathbf{d}'\right)$ such that $D_1\left(sk,\mathbf{c}'\right) = \mathbf{m}$. This means that no query to $\text{QDecaps}_m^{\perp}$ can trigger a query to $g' \times g$ $\left(\text{or } g'\right)$ on $\mathbf{m}$, so D_b will lose the games g_b if the games randomly pick one of the queries that were triggered by B's queries to $\text{QDecaps}_m^{\perp}$. This gives

$$P\left(g_b^{\mathrm{D}_b} \Rightarrow 1\right) \leq P\left(G_{14,b}^{B} \Rightarrow 1\right).$$

Since $g' = g(X)$ is a random polynomial of degree $2q_{g'}$ over $\mathbb{F}_{2^k}$, then if $\left(\mathbf{c}',\mathbf{d}'\right)$ is a valid encapsulation, $\mathbf{m}'$ will be a root of $g'(X) - \mathbf{d}'$. $G_{14,b}$ and $G_{15,b}$ should return the same outputs for every query $\left(\mathbf{c}', \mathbf{d}'\right) \neq (\mathbf{c}, \mathbf{d})$. Clearly the case where $\mathbf{c}' = \mathbf{c}$ will result in $\perp$ in all four games. All of the games also abort if $\mathbf{m}' = D_1\left(sk, \mathbf{c}'\right) = \mathbf{m}$. So the focus can be limited to the cases where $\mathbf{c}' \neq \mathbf{c}$ and $\mathbf{m}' = D_1\left(sk, \mathbf{c}'\right) \neq \mathbf{m}$. This leaves two cases.

1. In the first case $\text{QDecaps}_m^{\perp}\left(\mathbf{c}', \mathbf{d}'\right)$ will return $\perp$ in $G_{15,b}$. This means that $\mathbf{m}'$ is not a root of $g'(X) - \mathbf{d}'$. Now this case can only occur if and only if $g'\left(\mathbf{m}'\right) \neq \mathbf{d}'$ or $\mathbf{m}' = \perp$. In this case the condition in the last `if`-statement of $\text{QDecaps}_m^{\perp}\left(\mathbf{c}', \mathbf{d}'\right)$ in $G_{14,b}$ will return `false` and $G_{14,b}$ will return $\perp$ just as in $G_{15,b}$.
2. In the second case, $\text{QDecaps}_m^{\perp}\left(\mathbf{c}', \mathbf{d}'\right)$ will not return $\perp$ in $G_{15,b}$, so $\mathbf{m}'$ is a root of $g'(X) - \mathbf{d}'$ and $D_1\left(sk, \mathbf{c}'\right) = \mathbf{m}'$. This also means that $g'\left(\mathbf{m}'\right) = \mathbf{d}'$ and so $\text{QDecaps}_m^{\perp}\left(\mathbf{c}', \mathbf{d}'\right)$ will return $K' = g\left(\mathbf{m}'\right)$ in $G_{15,b}$. In this case the condition in the last `if`-statement in $G_{14,b}$ will return `true` and so $\text{QDecaps}_m^{\perp}\left(\mathbf{c}', \mathbf{d}'\right)$ also return $K' = g\left(\mathbf{m}'\right)$ in $G_{14,b}$.

Combining the above two cases should make it clear that $\text{QDecaps}_m^{\perp}$ is equivalent in the two games. This means that

$$P\left(G_{15,b}^B \Rightarrow 1\right) = P\left(G_{14,b}^B\right).$$

Let PCO be a plaintext checking oracle as has been seen before in the last chapter.

Algorithm 6.28 $A_0^{\text{PCO}}(pk, \mathbf{c})$

```
$(\mathbf{d}, K) \xleftarrow{\$} \{0,1\}^{k+m}$
$i \xleftarrow{\$} \left[q_g + q_{g'}\right]$
Run $B^{\text{QDecaps}_m^{\perp}, |g\rangle, |g'\rangle}(pk, (\mathbf{c}, \mathbf{d}), K)$ until the $i$'th query $|\hat{\mathbf{m}}\rangle$ to $|g \times g'\rangle$
**if** $i >$ number of queries to $|g \times g'\rangle$ **then**
    **Return** 0
**else**
    $\mathbf{m}' \leftarrow \text{Measure}(|\hat{\mathbf{m}}\rangle)$
    **Return** $\mathbf{m}'$
**end if**
```

Algorithm 6.29 $A_1^{\text{PCO}}(pk, \mathbf{c})$

```
$(\mathbf{d}, K) \xleftarrow{\$} \{0,1\}^{k+m}$
$i \xleftarrow{\$} \left[q_{g'}\right]$
Run $B^{\text{QDecaps}_m^{\perp}, |g\rangle, |g'\rangle}(pk, (\mathbf{c}, \mathbf{d}), K)$ until the $i$'th query $|\hat{\mathbf{m}}\rangle$ to $|g'\rangle$.
**if** $i >$ number of queries to $|g'\rangle$ **then**
    **Return** $\perp$
**else**
    $\mathbf{m}' \leftarrow \text{Measure}(|\hat{\mathbf{m}}\rangle)$
    **Return** $\mathbf{m}'$
**end if**
```

Algorithm 6.30 $\mathrm{QDecaps}_m^{\perp}\left(\left(\mathbf{c}', \mathbf{d}'\right) \neq (\mathbf{c}, \mathbf{d})\right)$ for A_0 and A_1

```
if $\mathbf{c}' = \mathbf{c}$ then
    Return $\perp$
end if
if $\exists \mathbf{m}' \in \mathrm{Roots}\left(g'(X) - \mathbf{d}'\right) \ : \ \mathrm{PCO}\left(\mathbf{m}', \mathbf{c}'\right) = 1$ then
    Return $K' = g\left(\mathbf{m}'\right)$
else
    Return $\perp$
end if
```

Again let $b \in \{0, 1\}$ and consider adversaries A_b from algorithms 6.28 to 6.30. A_b is an adversary against the $\mathrm{OW}-\mathrm{PCA}$ security of Π_1 that simulates $G_{15,b}$ for B.

Now $G_{15,b}^B \Rightarrow 1$ implies that no query $D_1\left(sk, \mathbf{c}'\right) = \mathbf{m}$ is ever done in $G_{15,b}^B$ and so even though A_b does not check whether or not $D_1\left(sk, \mathbf{c}'\right) = \mathbf{m}$ in their simulation of $\mathrm{QDecaps}_m^{\perp}\left(\mathbf{c}', \mathbf{d}'\right)$, B's view is not changed. This also means that

$$P\left(G_{15,b}^B \Rightarrow 1\right) = Adv_{\Pi_1}^{\mathrm{OW-PCA}}\left(A_b\right).$$

Notice that each time B makes a query to $\mathrm{QDecaps}_m^{\perp}$ on $\left(\mathbf{c}' \neq \mathbf{c}, \mathbf{d}'\right)$, A_b computes the set $\mathrm{Roots}\left(g'(X) - \mathbf{d}'\right)$ of complex roots. Using the fundamental theorem of algebra it can be seen that this set has $2q_{g'} - 1$ elements because $g'(X) - \mathbf{d}'$ is a polynomial of degree $2q_{g'} - 1$. In the worst case scenario they will need to check every element $\mathbf{m}' \in \mathrm{Roots}\left(g'(X) - \mathbf{d}'\right)$ to see if $\mathrm{PCO}\left(\mathbf{m}', \mathbf{c}'\right) = 1$.

Now folding A_0 and A_1 into a single $\mathrm{OW}-\mathrm{PCA}$ adversary A and collecting all the probabilities deduced from the games of this sections together yields

$$Adv_{\mathrm{QKEM}_m^{\perp}}^{\mathrm{CCA2}}(B) \leq \left(2q_{g'} + q_g + 2q_D\right) \sqrt{Adv_{\Pi_1}^{\mathrm{OW-PCA}}(A)} + \delta_1. \tag{6.31}$$

Provided that the running time of A is about that of B, this proves the following.

Theorem 6.32 *Let* Π_1 *be a public key cryptosystem that is* $\mathrm{OW}-\mathrm{PCA}$, *g and g' be hash functions and let* $\mathrm{QKEM}_m^{\perp} = QU_m^{\perp}\left(\Pi_1, g, g'\right)$ *be a key encapsulation mechanism. Then* $\mathrm{QKEM}_m^{\perp}$ *is* IND-CCA2 *in the quantum random oracle model.*

6.3.2 Implicit Rejection In The QROM

One can also do implicit rejection in the QROM just as in section 5.7. The key generation algorithm again appends a uniformly random string $\mathbf{s}$ to the secret key and when decapsulation is performed, this is again used to make the alternative key that signifies a rejection:

$$K = g(\mathbf{s}, \mathbf{c}, \mathbf{d}).$$

Other than that the rest is just as in section 6.3.

This is also a way to achieve a key exchange mechanism that is IND-CCA2. The security reduction follows the same discussion that is in section 6.3.1 with some minor differences. In this case $\mathrm{QDecaps}_m^{\perp}$ will be changed to a $\mathrm{QDecaps}_m^{\perp}$ that always knows if $(\mathbf{c}, \mathbf{d})$ is valid or not and the rejection $\perp$ will be changed to $g(\mathbf{s}, \mathbf{c}, \mathbf{d})$. The rest of the proof is the same [22].

6.4 The Final CCA2 Secure Key Encapsulation Mechanism In The QROM

This section really emphasises why the McEliece PKC is so interesting. Remember how in section 5.8 I argued that the McEliece PKC is OW − CPA. Well again letting the original McEliece PKC be denoted MCE, then theorems 6.16 and 6.32 taken together shows that

$$\begin{aligned}\mathrm{QKEM}_m^{\perp} = \mathrm{QFO}_m^{\perp} = QU_m^{\perp}\left(T\left(\mathrm{MCE}, h\right), g, g'\right)\\ = \left(\mathrm{QGEN} = G_1, \mathrm{QEncaps}_m, \mathrm{QDecaps}_m^{\perp}\right)\end{aligned}$$

is a key encapsulation mechanism that is IND-CCA2. Well almost. The real world differs from the world of random oracles, so this is technically just a conjecture. But it is a well-founded one. Let me discuss this a bit further before showing how to build the converted MCE.

6.4.1 The Hash Functions

As for the T-transformation, the hash function from [1] can still be used for the same reasons as in section 5.8.1. What is then needed is a discussion of the original McEliece PKC in the quantum random oracle mode. I will come back to this in a bit.

As for $QU_m^{\perp}$ the hash functions used need to be pseudorandom and collision intractable due to the same reasons given in section 5.8.1. One of them needs to take input of an arbitrary length and the other needs to take input of length k (with k being defined as in the original McEliece PKC). There are luckily some well-known cryptographic hash functions that can be used for the arbitrary case that have not yet been broken by quantum computers. As for the case, where the input needs to be of length k, one could still use a hash function that takes an arbitrary input, since it will then also take input of length k (being consistent with definition 5.3). This solves all pertaining issues as to which hash functions are to be used.

6.4.2 Applying The Transformations

Having taken care of the hash functions that are to be used, I will lay out here, how the converted McEliece PKC looks after applying the T-transformation and then the $QU_m^{\perp}$-transformation.

System Parameters: $n, t \in \mathbb{N}$ where $t \leq \left\lfloor \frac{n-1}{2} \right\rfloor$.

Key Generation: Given the parameters n, t generate the following matrices:

G: $k \times n$ generator matrix of an irreducible binary $[n, k]$ Goppa code $\mathcal{G}$ which can correct up to t errors.
S: $k \times k$ random binary non-singular matrix.
P: $n \times n$ random permutation matrix.

Then compute the $k \times n$ matrix $G' = \mathrm{SGP}$,
build a hashing algorithm h that outputs bit strings of length n and (Hamming)

weight t (see e.g. [1]),
choose a collision intractable hash function g that takes input of arbitrary length and gives an output of length k and
choose a collision intractable hash function g' that takes input of length k and gives output of length k.

Public Key: $\left(G', t, h, g, g'\right)$.

Private Key: $\left(S, D_{\mathcal{G}}, P\right)$ where $D_{\mathcal{G}}$ is an efficient decoding algorithm for $\mathcal{G}$ (see e.g. algorithm 2.22).

Encapsulation: Start by encrypting the message $\mathbf{m} \in \{0,1\}^k$. This is done by computing $h(\mathbf{m}) = \mathbf{z} \in \{0,1\}^n$ and then computing the ciphertext

$$\mathbf{c} = \mathbf{m}G' \oplus \mathbf{z}.$$

Then compute

$$\mathbf{d} = g'(\mathbf{m})$$

and the key

$$K = g(\mathbf{m}).$$

Decapsulation: Upon receiving $(\mathbf{c}, \mathbf{d})$, start off by decrypting $\mathbf{c}$. This is done by calculating

$$\mathbf{c}\mathrm{P}^{-1} = \mathbf{m}'\mathrm{S}G \oplus \mathbf{z}\mathrm{P}^{-1}.$$

Then apply the decoding algorithm $D_{\mathcal{G}}$ to this. Because $\mathbf{c}\mathrm{P}^{-1}$ should have a hamming distance of t to the Goppa code, the codeword obtained should be

$$\mathbf{m}'\mathrm{S}G = D_{\mathcal{G}}\left(\mathbf{c}\mathrm{P}^{-1}\right).$$

Now one can now compute the plaintext $\mathbf{m}'$ as

$$\mathbf{m}' = \left(\mathbf{m}'\mathrm{S}G\right) G^{-1}\mathrm{S}^{-1}.$$

Then compute

$$\mathbf{c}' = \mathbf{m}'G' \oplus h\left(\mathbf{m}'\right).$$

If $\mathbf{m}'$ is not a valid plaintext or if $\mathbf{c}' \neq \mathbf{c}$, then decryption has failed and the procedure should abort.
If decryption succeeds however, then compute

$$\mathbf{d}' = g'\left(\mathbf{m}'\right)$$

and see if $\mathbf{d}' = \mathbf{d}$. If this is not the case, then abort. Otherwise return $K = g\left(\mathbf{m}'\right)$.

Again it makes no sense to talk of a non-valid plaintext, unless it is not of length k, which can safely be assumed if protocol is followed correctly. The check to see whether or not $\mathbf{m}'$ is a valid plaintext is then included, such as to make sure that if two parties agree on some structure on $\mathbf{m}$, then such a case is still covered.

6.4.3 Security After Applying The Transformations

Suppose the original McEliece PKC could be broken. Then an adversary could find **m** from **c** and imitate a legitimate party in a key exchange protocol or get to know the key that two legitimate parties wish to use for their symmetric encryption scheme. Obviously this would pose a problem. However, there are as of yet not any published classical- or quantum attacks on the original McEliece PKC that breaks the one-wayness of the original proposal of McEliece.

Now let $q_{\mathrm{RO}} = q_h + q_g + q_{g'}$ denote the total number of queries that the CCA2 adversary B implicitly or explicitly sends to the quantum random oracles h, g and g'. Combining eq. (6.15) and eq. (6.31) then gives

$$Adv^{\mathrm{CCA2}}_{\mathrm{QKEM}^{\perp}_{m}}(B) \leq 8q_{\mathrm{RO}} \left(\sqrt{q^2_{\mathrm{RO}} \cdot \delta + q_{\mathrm{RO}} \cdot \sqrt{Adv^{\mathrm{OW-CPA}}_{\mathrm{MCE}}(A)}} + q_{\mathrm{RO}} \cdot \delta \right)$$

and just like in section 5.8.2.1 it is then entirely possible to achieve an arbitrary "κ bits of security", with the security only relying on the $\mathrm{OW} - \mathrm{CPA}$ security of the original McEliece PKC (that is, it is relying on k, n and t from the original McEliece PKC).

Chapter 7

Conclusions And Future Work

The goal of this thesis was to describe the original McEliece PKC and outline a theoretical framework for using it in the face of an attacker – supposing that the attacker could be either a classical- or a quantum adversary. I then went thoguh how this can be done using the transformations laid out in chapters 5 and 6.

As public key cryptosystems are primarily used for key exchange protocols the transformations that I have laid out are used to build a key encapsulation mechanism. Now, CCA2 security is generally considered the standard level of security for public key cryptosystems in modern times and the aforementioned transformations achieves exactly this level of security when applied to the original McEliece PKC as shown in their respective chapters – unlike the original McEliece PKC, as was shown in chapter 4. Thus a way of exchanging keys in a safe way is achieved even under the threat of a quantum adversary.

One of the big takeaways from this thesis is then that the threat of general quantum computers becoming a real concern in the near future is eliminated, unless someone manages to prove that P = NP or someone comes up with a new quantum algorithm that can exploit the nature of the McEliece PKC or cryptographic hash functions.

7.1 Potential Future Work

The work has been done using Fujisaki-Okamoto transformations, but there are other ways to achieve CCA2 security that I have not touched upon. These other transformations of the original McEliece PKC could also be tested out in future work. In particular I would like to note that there is a way to transform the cryptosystem so as to not use Goppa codes but another type of linear codes whilst still achieving the desired security in the random oracle model whilst not needing the proposed hash algorithm from [1] [14]. Such methods could (and should) be tested in the future for security in the quantum random oracle model.

Additionally I would like to note that there are even more Fujisaki-Okamoto transformations that I have not yet introduced, but could be worked into a further study of this field. These are for example the original one that Fujisaki and Okamoto suggested in 1999. There are also other transformations from [22] that achieves tight security reductions in the classical random oracle model. This is not something that I have gone through, since it would have required me to introduce yet another transformation from OW − CPA to yet another security definition before applying the T-transformation from section 5.6 and it is not necessary in order to achieve tightness in the quantum random oracle model, which actually was achieved.

Yet another next logical step for future work would be to do an implementation in software (or hardware – or both). Doing it in hardware is not something that I dare undertake, but doing so in software would be reasonable.

Bibliography

[1] Alessandro Barenghi and Gerardo Pelosi. "Constant Weight Strings in Constant Time: A Building Block for Code-Based Post-Quantum Cryptosystems". In: *Proceedings of the 17th ACM International Conference on Computing Frontiers.* CF '20. Catania, Sicily, Italy: Association for Computing Machinery, 2020, pp. 132–141.

[2] R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf. "Quantum lower bounds by polynomials". In: *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No.98CB36280).* 1998, pp. 352–361.

[3] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. "Deterministic and Efficiently Searchable Encryption". In: *Advances in Cryptology - CRYPTO 2007.* Ed. by Alfred Menezes. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 535–552.

[4] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. "Relations among notions of security for public-key encryption schemes". In: *Advances in Cryptology — CRYPTO '98.* Ed. by Hugo Krawczyk. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 26–45.

[5] Mihir Bellare, Shai Halevi, Amit Sahai, and Salil Vadhan. "Many-to-one trapdoor functions and their relation to public-key cryptosystems". In: *Advances in Cryptology — CRYPTO '98.* Ed. by Hugo Krawczyk. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 283–298.

[6] Mihir Bellare and Phillip Rogaway. "The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs". In: *Advances in Cryptology - EUROCRYPT 2006.* Ed. by Serge Vaudenay. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 409–426.

[7] E. Berlekamp, R. McEliece, and H. van Tilborg. "On the Inherent Intractability of Certain Coding Problems". In: *IEEE Transactions on Information Theory* 24.3 (1978), pp. 384–386.

[8] Daniel J. Bernstein and Edoardo Persichetti. *Towards KEM Unification.* Cryptology ePrint Archive, Paper 2018/526. `https://eprint.iacr.org/2018/526`. 2018.

[9] Thomas A. Berson. "Failure of the McEliece public-key cryptosystem under message-resend and related-message attack". In: *Advances in Cryptology — CRYPTO '97.* Ed. by Burton S. Kaliski. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 213–220.

[10] Dan Boneh et al. “Random Oracles in a Quantum World”. In: *Advances in Cryptology – ASIACRYPT 2011*. Ed. by Dong Hoon Lee and Xiaoyun Wang. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 41–69.

[11] A. Canteaut and F. Chabaud. “A new algorithm for finding minimum-weight words in a linear code: application to McEliece’s cryptosystem and to narrow-sense BCH codes of length 511”. In: *IEEE Transactions on Information Theory* 44.1 (1998), pp. 367–378.

[12] Anne Canteaut and Florent Chabaud. “Improvements of the Attacks on Cryptosystems Based on Error-Correcting Codes”. In: *Rapport interne du Departement Mathematiques et Informatique* 95 (1995), p. 21.

[13] Anne Canteaut and Nicolas Sendrier. “Cryptanalysis of the Original McEliece Cryptosystem”. In: *Advances in Cryptology — ASIACRYPT’98*. Ed. by Kazuo Ohta and Dingyi Pei. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 187–199.

[14] Pierre-Louis Cayrel, Gerhard Hoffmann, and Edoardo Persichetti. “Efficient Implementation of a CCA2-Secure Variant of McEliece Using Generalized Srivastava Codes”. In: *Public Key Cryptography – PKC 2012*. Ed. by Marc Fischlin, Johannes Buchmann, and Mark Manulis. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 138–155.

[15] Florent Chabaud. “On the security of some cryptosystems based on error-correcting codes”. In: *Advances in Cryptology — EUROCRYPT’94*. Ed. by Alfredo De Santis. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 131–139.

[16] Danny Dolev, Cynthia Dwork, and Moni Naor. “Non-Malleable Cryptography”. In: *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*. STOC ’91. New Orleans, Louisiana, USA: Association for Computing Machinery, 1991, pp. 542–552.

[17] D. Engelbert, R. Overbeck, and A. Schmidt. “A Summary of McEliece-Type Cryptosystems and their Security”. In: *Journal of Mathematical Cryptology* 1.2 (2007), pp. 151–199.

[18] Marc P. C. Fossorier, Kazukuni Kobara, and Hideki Imai. “Modeling Bit Flipping Decoding Based on Nonorthogonal Check Sums With Application to Iterative Decoding Attack of McEliece Cryptosystem”. In: *IEEE Transactions on Information Theory* 53.1 (2007), pp. 402–411.

[19] Eiichiro Fujisaki and Tatsuaki Okamoto. “Secure Integration of Asymmetric and Symmetric Encryption Schemes”. In: *Advances in Cryptology - CRYPTO ’99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*. Vol. 1666. Lecture Notes in Computer Science. Springer, 1999, pp. 537–554.

[20] Valerii Denisovich Goppa. “A new class of linear correcting codes”. In: *Problems of Information Transmission* 6.3 (1970), pp. 207–212.

[21] Chris Hall, Ian Goldberg, and Bruce Schneier. “Reaction Attacks against Several Public-Key Cryptosystem”. In: *Information and Communication Security*. Ed. by Vijay Varadharajan and Yi Mu. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 2–12.

[22] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. *A Modular Analysis of the Fujisaki-Okamoto Transformation*. Cryptology ePrint Archive, Paper 2017/604. `https://eprint.iacr.org/2017/604`. 2017.

[23] Andreas Hülsing, Joost Rijneveld, and Fang Song. *Mitigating Multi-Target Attacks in Hash-based Signatures*. Cryptology ePrint Archive, Paper 2015/1256. `https://eprint.iacr.org/2015/1256`. 2015.

[24] A. Al Jabri. “A Statistical Decoding Algorithm for General Linear Block Codes”. In: *Cryptography and Coding*. Ed. by Bahram Honary. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 1–8.

[25] Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. “A Concrete Treatment of Fiat-Shamir Signatures in the Quantum Random-Oracle Model”. In: *Advances in Cryptology – EUROCRYPT 2018*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Cham: Springer International Publishing, 2018, pp. 552–586.

[26] Kazukuni Kobara and Hideki Imai. “Semantically Secure McEliece Public-Key Cryptosystems -Conversions for McEliece PKC -”. In: *Public Key Cryptography*. Ed. by Kwangjo Kim. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 19–35.

[27] Niels Lauritzen. *Concrete Abstract Algebra: From Numbers to Gröbner Bases*. Cambridge University Press, 2003.

[28] P. J. Lee and E. F. Brickell. “An Observation on the Security of McEliece’s Public-Key Cryptosystem”. In: *Advances in Cryptology — EUROCRYPT ’88*. Ed. by D. Barstow et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 275–280.

[29] J.S. Leon. “A probabilistic algorithm for computing minimum weights of large error-correcting codes”. In: *IEEE Transactions on Information Theory* 34.5 (1988), pp. 1354–1359.

[30] F. Levy-dit-Vehel and S. Litsyn. “Parameters of Goppa codes revisited”. In: *IEEE Transactions on Information Theory* 43.6 (1997), pp. 1811–1819.

[31] J.H. van Lint. *Introduction to Coding Theory*. Third Edition. Springer Berlin, Heidelberg, 1999.

[32] P. Loidrean and N. Sendrier. “Some weak keys in McEliece public-key cryptosystem”. In: *Proceedings. 1998 IEEE International Symposium on Information Theory (Cat. No.98CH36252)*. 1998, p. 382.

[33] Robert J McEliece. “A Public-Key Cryptosystem Based On Algebraic Coding Theory”. In: *The Deep Space Network Progress Report* 42-44 (1978), pp. 114–116.

[34] Robert Niebuhr, Mohammed Meziani, Stanislav Bulygin, and Johannes Buchmann. “Selecting parameters for secure McEliece-based cryptosystems”. In: *International Journal of Information Security* 11 (2012), pp. 137–147.

[35] R. Overbeck. “Statistical Decoding Revisited”. In: *Information Security and Privacy*. Ed. by Lynn Margaret Batten and Reihaneh Safavi-Naini. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 283–294.

[36] Edoardo Persichetti. “Improving the efficiency of code-based cryptography”. PhD thesis. Department Of Mathematics, University Of Auckland, 2012.

[37] Charles Rackoff and Daniel R. Simon. “Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack”. In: *Advances in Cryptology — CRYPTO ’91*. Ed. by Joan Feigenbaum. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 433–444.

[38] N. Sendrier. “Finding the permutation between equivalent linear codes: the support splitting algorithm”. In: *IEEE Transactions on Information Theory* 46.4 (2000), pp. 1193–1203.

[39] C. E. Shannon. “A mathematical theory of communication”. In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423.

[40] Victor Shoup. *Sequences of games: a tool for taming complexity in security proofs*. Cryptology ePrint Archive, Paper 2004/332. `https://eprint.iacr.org/2004/332`. 2004.

[41] Jacques Stern. “A method for finding codewords of small weight”. In: *Coding Theory and Applications*. Ed. by Gérard Cohen and Jacques Wolfmann. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989, pp. 106–113.

[42] Hung-Min Sun. “Further cryptanalysis of the McEliece public-key cryptosystem”. In: *IEEE Communications Letters* 4.1 (2000), pp. 18–19.

[43] Ehsan Ebrahimi Targhi and Dominique Unruh. “Post-Quantum Security of the Fujisaki-Okamoto and OAEP Transforms”. In: *Theory of Cryptography*. Ed. by Martin Hirt and Adam Smith. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 192–216.

[44] Dominique Unruh. *Revocable quantum timed-release encryption*. Cryptology ePrint Archive, Paper 2013/606. `https://eprint.iacr.org/2013/606`. 2013.

[45] Alexander Vardy. “The intractability of computing the minimum distance of a code”. In: *IEEE Transactions on Information Theory* 43.6 (1997), pp. 1757–1766.

[46] Mark Zhandry. “How to Construct Quantum Random Functions”. In: *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*. 2012, pp. 679–687.

[47] Mark Zhandry. “Secure Identity-Based Encryption in the Quantum Random Oracle Model”. In: *Advances in Cryptology – CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 758–775.