TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

Linear Codes and Applications in Cryptography

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science (MSc)

in

Mathematics in Computer Science

by

Matthias Minihold, BSc
Registration Number: 0726352
Address: 3923 Jagenbach 140
eMail: matthias.minihold@gmx.at

to the Institute of Discrete Mathematics and Geometry
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr. Gerhard Dorfer

Vienna, 1.5.2013

(Signature of Author) (Signature of Advisor)

Contents

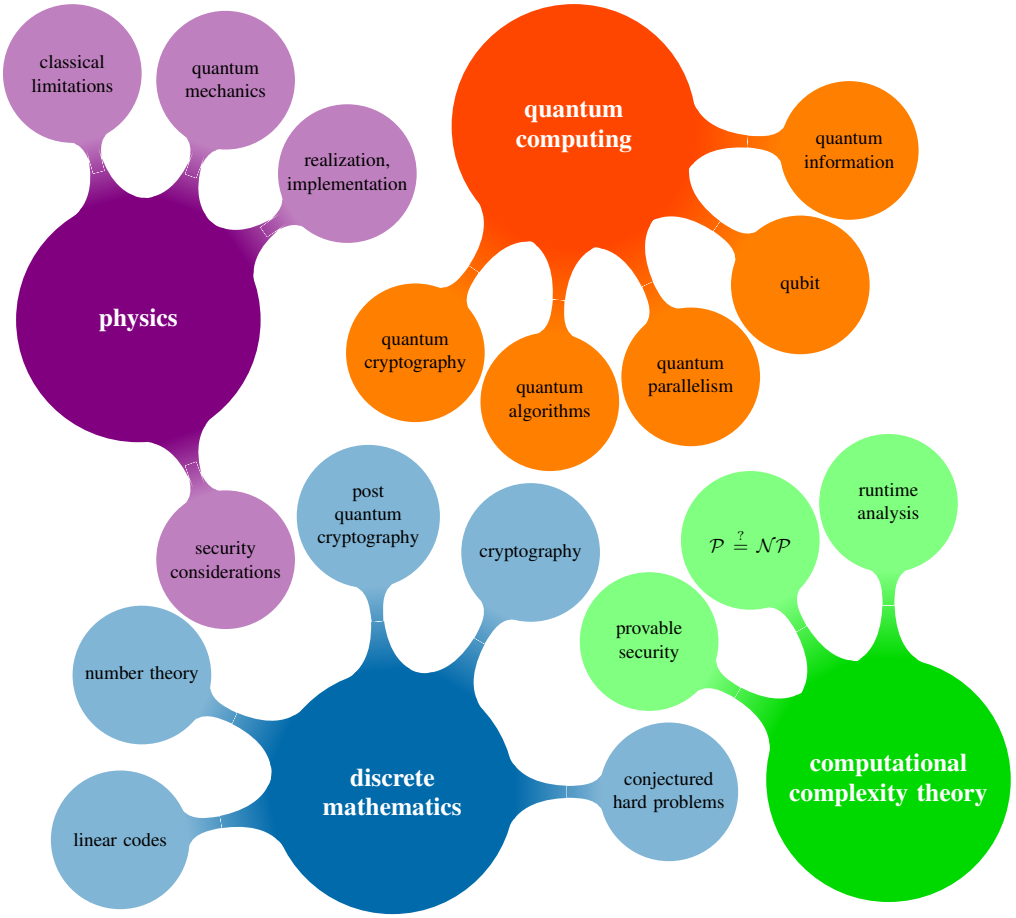

Figure 1: A mind map visualizing concepts and associations of topics in this thesis.

Abstract

In this master's thesis the focus is on bringing together interesting results of different areas — ranging from computational complexity theory to quantum physics — about the role of linear codes in modern cryptography from a mathematical point of view. See Figure 1 for an overview. We discuss algorithms that solve certain cryptographic tasks and thus we illuminate the application's side as well, which outlines the constructive manner of this field. On the one hand the interesting mathematical theory is presented and on the other hand we point out where the theory can be used directly for applications in nowadays information society.

Zusammenfassung

Diese Masterarbeit versucht die Ergebnisse unterschiedlicher Gebiete — angefangen von theoretischer Informatik bis hin zur Quantenphysik — miteinander zu kombinieren um die Rolle von linearen Codes in der modernen Kryptographie aus dem Standpunkt der diskreten Mathematik zu beleuchten. Siehe Abbildung 1 zur Übersicht. Es werden die wichtigsten Algorithmen präsentiert, welche gegebene kryptographische Problemstellungen lösen, um auch den Anforderungen der Anwendungsseite gerecht zu werden. Das verdeutlicht die konstruktive Natur dieses Gebiets; die interessante mathematische Theorie wird direkt verwendet um daraus praktischen Nutzen in unserer heutigen Informationsgesellschaft zu ziehen.

Danksagung, Acknowledgements, Tackar

- Ein großes Dankeschön an meine ganze Familie, meine Freunde und Studienkollegen für die Unterstützung und den Rückhalt während meiner gesamten Studienzeit. Ich möchte durchaus erwähnen, dass die beiden letzten Mengen nicht-leeren Schnitt haben!

- Thanks to my advisor Gerhard Dorfer for enabling me to write my master's thesis about this interesting topic in the first place and for the discussions and hints during the process of writing. Furthermore thank you for your flexibility which made it possible to finalize this thesis abroad — in Stockholm.

- Jag vill tacka Rikard Bøgvad och Hoshang Heydari så mycket för de hjälpsamma kommentarerna och för det varma välkomnandet i Stockholm.

- Additionally to all the personal thanking, I want to devote a small paragraph to the often overlooked and seldom thanked people: Thanks to the whole OpenSource community for providing software, templates and help in numerous forms such as helpful discussions in forums. A lot of work is done in the background by writing manuals for tools or a step by step how-to for a certain task, for example.

Introduction

Linear codes are an interesting area in discrete mathematics for two reasons. First of all they provide a mechanism to transmit data over a noisy channel while ensuring the data's integrity. On the other hand, as we will see, they can be used to protect data against unwanted readers, by using them for encryption.
Techniques of discrete mathematics are used to enhance the information transmission rate and different codes are being studied to provide solutions for various problems occurring in applications. Deep space communication for example has other needs than the telephone network for mobile phones, thus there is a variety of classes of channel codes. Each class meets other specific requirements and some classes, with importance for cryptography, are discussed in Chapter 1.
It is often said nowadays that we live in an information society, based upon information processing. Huge amounts of text and data is transmitted over digital networks today. Thus topics like protection of sensible information and the need for mechanisms to ensure integrity of data arise. Chapter 2 focuses on modern cryptography. Further motivation why data protection is necessary is given there. Then cryptographic primitives are discussed in general and the role of linear codes in this broad topic is especially illuminated. Possible applications and algorithms round this chapter off.
Before the general discussion is extended, we present examples of such public-key cryptosystems previously defined. Therefore we deal with the famous McEliece and Niederreiter cryptosystems in Chapter 3 and provide a scalable toy-example to illustrate the functioning using the computer algebra system Sage.
Then we assume a powerful, working quantum computer in Chapter 4 and regard the impact of such a device on current state-of-the-art cryptosystems. The question why and how certain systems based on linear codes withstand this threat so far is dealt with, too. The discussion continues with future needs and enhancements in the field to make cryptography ready in a post-quantum world, where a powerful quantum computer exists. Finally — in Section 4.6 — a relatively new approach, namely quantum cryptography, as well as the relations to and the implications for classical modern cryptography is addressed. We close the circle by giving an outlook where development might head to and see the important role of linear codes in modern cryptography once more.

1 Linear Codes

It is assumed that the reader is familiar with linear algebra and the basics about finite fields. We will now start with some basics in coding theory and then advance to the codes crucial for this thesis.
Here we are mostly interested in channel coding that means information needs to be transmitted over a noisy communication channel, where errors may happen. Additional information bits are appended before sending the message over the noisy channel. The reason of the redundancy is that it can be used to correct or at least detect errors that occurred during sending. How these redundancy comes into the data and how to efficiently encode and later decode the data are questions dealt with in the following sections.
Source coding is about compressing data or representing given source bits with a minimal amount of bits that therefore carry more information. Compression, contrary to channel coding, is a technique to avoid redundancy. Both are not further investigated in this thesis; only one last remark for readers familiar with telecommunication as sub-field of electric engineering. There, the term channel coding is used synonymously for forward error correction (FEC).
As can be seen in Chapter 2 about cryptography the codes originally constructed for error correction can also be used to deflate a strong cryptographic scheme for protecting information, sent over an insecure channel, against eavesdroppers.
The idea of using a technique that is good for one purpose exactly the other way round was striking for me. Hence my interest awoke and I began reading about related topics in more detail, which finally led to writing these lines and essentially this master's thesis.
Unless cited differently, the notations, definitions and results in this chapter are taken from my notes of the lecture "Fehlerkorrigierende Codes" (error correcting codes) [13] given by Prof. Gerhard Dorfer at Vienna University of Technology.

1.1 Definitions

The linear codes that are of interest in this work, are linear block codes over an alphabet $A = \mathbb{F}_q$, where $\mathbb{F}_q$ denotes the finite field with $q = p^\ell$ elements $\ell \in \mathbb{N}^\times, p$ prime. The alphabet is often assumed to be binary that is $p = 2, \ell = 1, q = 2, \mathbb{F}_2 = \{0, 1\}$. The encoding of the source bits is done in blocks of predefined length k, giving rise to the

name "block code".
Formally one can interpret encoding as applying an injective $\mathbb{F}_q-$linear function $f_C : \mathbb{F}_q^k \to \mathbb{F}_q^n$ on an input block of length k. We want the mapping f_C to be injective for the simple reason to be able to uniquely reconstruct the source block from the codeword. Thus the coordinates of f_C written as a $(k \times n)$ matrix G (called the generator matrix) describe the code C. The name generator matrix is used because every codeword can be generated by multiplying a source vector $x \in \mathbb{F}_q^k$ with G; $C = \left\{x \cdot G \mid x \in \mathbb{F}_q^k\right\} \leq \mathbb{F}_q^n$.
We see, a linear code C is a k-dimensional sub-vectorspace of $\mathbb{F}_q^n$, where n is called the length. The dimension $k \leq n$ corresponds to the number of information symbols of C, which for interesting codes is $< n$ due to the mentioned redundancy the code adds to the original information block. k is also referred to as dimension of the code. The sum $c := c_1 + c_2$ of any two codewords $c_1, c_2 \in C$ is again a codeword $c \in C \leq \mathbb{F}_q^n$. A codeword can be either seen as concatenated elements of the alphabet or as a vector, depending on which representation is more intuitive.

Definition 1.1. *Let $x = (x_1, x_2, \ldots, x_n)$, $y = (y_1, y_2, \ldots, y_n) \in \mathbb{F}_q^n$, then the Hamming distance of these two vectors is defined by $d(x, y) := |\{i : x_i \neq y_i\}|$.*
The minimal distance for a code C is $d := \min\{d(c, \bar{c})|c, \bar{c} \in C, c \neq \bar{c}\}$.
The Hamming weight of $x \in \mathbb{F}_q^n$ is defined by $w(x) := |\{i : x_i \neq 0\}|$. The weight $w(x) = d(x, 0)$ is the Hamming distance to the zero vector.

In case of a linear code it is easily obtained that $d = \min\{w(c) \mid c \in C, c \neq 0\}$. In general, the distance can be expressed in terms of the weight $d(x, y) = w(x - y)$.
Conformal with the standard literature about coding theory [22] an $[n, k, d]-$code C denotes a linear code. The codewords have length n, carry k bits non-redunant information and d is the minimal distance of C. On the other hand, a $(n, M, d)-$code is not necessarily linear; here M denotes a set of codewords. To get the more general parameter M in the linear case, let M be the set of all elements of the k-dimensional sub-vectorspace of $\mathbb{F}_q^n$ with cardinality $|M| = q^k$, where q is the number of the alphabet elements.

Example 1.2. *Consider the following binary $[5, 2, 3]-$code C with minimum distance $d = 3$. $C = \{00\ 000, 10\ 110, 01\ 101, 11\ 011\}$ would be a $(5, 2^2 = 4, 3)-$code since the encoding function $f_C : \mathbb{F}_2^2 \to \mathbb{F}_2^5$ generates a $2-$dimensional sub-vectorspace of $\mathbb{F}_2^5$.*

Definition 1.3. *Important relative parameters of an $[n, k, d]-$code are the relative minimal distance $D = \frac{d}{n}$ and the information rate $R = \frac{k}{n}$.*

Let's take a brief look on transferring data over a noisy communication channel, where errors are likely to occur, from a coding theoretic viewpoint:

Information $I \Rightarrow$ source (en)coding $\Rightarrow x = (x_1, x_2, \ldots, x_k) \in A^k$ message block $\Rightarrow$ channel (en)coding $\Rightarrow c = (c_1, c_2, \ldots, c_n) \in C \subseteq A^n$ codeword $\Rightarrow$ submission via noisy channel (this is where errors might alter the message) $\Rightarrow$ received message block $y = c + e \Rightarrow$ decoding $\Rightarrow \tilde{x} \approx x \Rightarrow$ inverse source coding $\Rightarrow$ (received information) $\tilde{I} \approx I$ (information before transmission).

Later we will see how this scheme can be extended when the information needs to be securely transmitted.

Theorem 1.4. *An $[n,k,d]-$code C can correct up to $\lfloor \frac{d-1}{2} \rfloor$ errors and detect $d-1$ errors (in the sense that the decoder knows some error occurred).*

Theorem 1.5 (Singleton Bound)**.** *Let C be a linear $[n,k,d]-$code then $k+d \leq n+1$.*

The rows of the $(k \times n)$ generator matrix G are a basis of the $\mathbb{F}_q$-vectorspace C. Encoding is a matrix multiplication $f_C(x) = xG$ that can be established with much less effort if G is given in a systematic block form $G = (I_k|M), I_k$ is the $(k \times k)$ identity matrix and M is a $(k \times (n-k))$ matrix over the alphabet A. Systematic encoding appends $(n-k)$ redundancy symbols, it maps $x = x_1 x_2 \dots x_k$ to $f_C(x) = x_1 x_2 \dots x_k c_1 \dots c_{n-k}$.
An important role in decoding a linear code C has the so-called parity check matrix H which is defined as $((n-k)\times n)$ matrix H with rank $n-k$ and the property that $H \cdot G^T = 0$. In particular, if the generator matrix $G = (I_k|M)$ has systematic form, then the parity check matrix $H = (-M^T|I_{n-k})$ has a very simple structure.

Definition 1.6. *The dual code of a linear code $C \leq \mathbb{F}_q^n$ is defined as*

$$C^{\perp} := \left\{ x = (x_1, x_2, \dots, x_n) \in \mathbb{F}_q^n \mid \sum_{i=1}^{n} x_i \cdot c_i = 0 \in \mathbb{F}_q, \forall c = (c_1, c_2, \dots, c_n) \in C \right\}.$$

1.2 General Decoding Schemes

Maximum likelihood decoding and minimum distance decoding are two general decoding schemes for linear as well as non-linear codes. Syndrome decoding on the other hand is an efficient scheme for linear codes.
Later we will see decoding methods best suited for particular classes of linear codes. Also their complexity will be regarded because efficient decoding algorithms are desirable.
Fast decoding is a requirement for cryptographic schemes as we will elaborate in Chapter 2. There the question of decoding arises, but in a different context. In other words we discuss how it can be seen as the inverse of a trapdoor function (see Definition (2.4)).

List Decoding

Definition 1.7. *Let C be a code and $x \in \mathbb{F}_q^n$ a received vector. An algorithm which, given C and x, outputs the list*

$$L_C(x,t) := \{c \in C \mid d(x,c) \leq t\}$$

of all codewords at distance at most t to the fixed vector x is called a list decoding algorithm with decoding radius t.

For a linear $[n,k,d]-$code C, a list decoding algorithm with decoding radius $t = \lfloor \frac{d-1}{2} \rfloor$ is sometimes called bounded distance decoding. In this case $|L_C(x,t)| \leq 1$ for all $x \in \mathbb{F}_q^n$.

Maximum Likelihood Decoding

Given a received codeword $x \in \mathbb{F}_q^n$ maximum likelihood decoding (MLD) tries to find the codeword $y \in C$ to maximize the probability that x was received given that y was sent. This probability is denoted by $\mathbb{P}(x \text{ received} \mid y \text{ sent})$.

Minimum Distance Decoding

Minimum distance decoding (MDD) is also known as nearest neighbour decoding and tries to minimize the Hamming distance $d(x, y)$ for all codewords $y \in C$ given a received word $x \in \mathbb{F}_q^n$.
If the channel is a discrete memoryless channel (for instance a binary symmetric channel, where no symbol to be sent is dependent on the previous ones sent) and the probability that errors occur $p < \frac{1}{2}$ then minimum distance decoding is equivalent to maximum likelihood decoding.
As always, this method has restrictions one has to be aware of. If burst errors are likely to occur the assumption on the channel to be memoryless can be unreasonable.

Syndrome Decoding

For an $[n, k, d]-$code C we can assume that the parity check matrix H is given. The code C can be described as the kernel of the surjective map s_H defined by

$$s_H := \begin{cases} \mathbb{F}_q^n \to \mathbb{F}_q^{n-k} \\ x \mapsto s_H(x) := x \cdot H^T. \end{cases}$$

The fact that

$$s_H(x) = 0 \Leftrightarrow x \in C$$

and therefore

$$s_H(x) = s_H(y) \Leftrightarrow x + C = y + C$$

can then be used for decoding purposes, namely for constructing a lookup table. This table can be of reduced size for storing all possible syndromes along with a minimal weight representative of the corresponding coset. With the syndrome calculations during the decoding process one only needs storage for $(q^{n-k} \times 2)$ words instead of the naive approach, where only a lookup in a table with $(q^n \times 2)$ entries has to be set up.
For codes of practical relevance it is desirable that the information rate $R = k/n$ is high. Roughly speaking if R is close to 1, syndrome decoding is an advantage since $n - k$ is comparable small. Using the terminology of the previous sections syndrome decoding is minimum distance decoding for a linear code over a noisy channel.

1.3 Important Classes of Codes

The classes of codes presented in this section are a selection of a variety of interesting codes that exist. The importance of these particular codes is due to their practical applications.

The well known class of Reed-Solomon codes for example has already a broad field of application in today's life. They can be combined with other codes to further increase their efficiency. Their error correcting capability is used for data, where burst errors are likely to occur. Audio and video data, for instance, is stored on CD and DVD. The different areas of deployment range from broadcasting media in DAB, DVB-T, DVB-S, DVB-C to internet connections via xDSL (containing the well known ADSL), just to name a few.
Next we present BCH codes as generalization of Reed-Solomon codes.
Finally the strength of Goppa codes is discussed in this section including an efficient decoding algorithm (see Patterson's algorithm in Section 1.4.2) which can also be used for cryptographic purposes.

1.3.1 Cyclic Codes

Cyclic codes are special linear codes that correspond to an ideal in the ring $\mathbb{F}_q[x]/(x^n-1)$. A word is in the code if and only if the cyclic shift of the word is in the code

$$c_0c_1\dots c_{n-1} \in C \Leftrightarrow c_{n-1}c_0c_1\dots c_{n-2} \in C.$$

One can change the notation for a codeword c to a polynomial $c(x) \in \mathbb{F}_q[x]$ for the simpler handling and the fact that multiplication by $x \mod (x^n-1)$ corresponds to a cyclic right shift of one position:

$$c_0c_1\dots c_{n-1} \leftrightarrow c(x) = \sum_{i=0}^{n-1} c_ix^i \in \mathbb{F}_q[x]/(x^n-1).$$

The generator polynomial $g(x)$ of a cyclic code is the monic polynomial of smallest degree in C. It follows that $g(x)|(x^n-1)$ and therefore the dimension k of the code is $k = n - \deg g(x)$. Obviously we have

$$c = c(x) \in C \Leftrightarrow g(x)|c(x).$$

In general the minimal distance d cannot be derived directly from the generator polynomial $g(x) = g_0 + g_1x + \dots + 1x^{n-k}$ but a $(k \times n)$ generator matrix G is easily obtained and is given by

$$G = \begin{pmatrix} g_0 & g_1 & g_2 & \dots & g_{n-k-1} & 1 & 0 & \dots & 0 \\ 0 & g_0 & g_1 & g_2 & \dots & g_{n-k-1} & 1 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & & & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & g_0 & g_1 & \dots & \dots & g_{n-k-1} & 1 \end{pmatrix}.$$

Observe that the leading coefficient of $g(x)$ is 1 and therefore another important polynomial is the uniquely determined parity check polynomial $h(x) = \frac{x^n-1}{g(x)}$ which can be used for decoding purposes.
The syndrome of cyclic codes can either be computed using $g(x)$ or $h(x)$

$$s_h(p(x)) := p(x)h(x) \mod (x^n-1), \tag{1.1}$$
$$s_g(p(x)) := p(x) \mod g(x). \tag{1.2}$$

Equation (1.1) can be used to calculate the syndrome because

$$c = c(x) \in C \Leftrightarrow g(x)|c(x) \Leftrightarrow (x^n - 1) = h(x)g(x)|h(x)c(x) \Leftrightarrow s_h(c(x)) = 0.$$

Alternatively s_g maps codewords to 0, because

$$c(x) \in C \Leftrightarrow g(x)|c(x) \Leftrightarrow c(x) \equiv 0 \mod g(x).$$

The dual $C^\perp$ (see Definition (1.6)) of a cyclic code C is again cyclic with generator polynomial $g^\perp(x) := x^k h(\frac{1}{x})$.

1.3.2 RS Codes

Reed-Solomon codes (RS) are defined over an alphabet $A := \mathbb{F}_q$. Along with the alphabet, the codeword length $n = q - 1$ is fixed too.
Additionally let β a primitive element in $\mathbb{F}_q$. This means the field can be written as $\mathbb{F}_q = \{0, 1 = \beta^0 = \beta^{q-1}, \beta^1, \beta^2, \ldots, \beta^{q-2}\}$ and each element in $\mathbb{F}_q^\times$ is a certain power of the primitive element β and therefore one gets

$$x^n - 1 = x^{q-1} - 1 = \prod_{j=0}^{q-2}(x - \beta^j) = \prod_{\gamma \in \mathbb{F}_q^\times}(x - \gamma).$$

Given fixed numbers b and k one defines the generator polynomial of the RS code as

$$g(x) := \prod_{j=0}^{n-k-1}(x - \beta^{b+j}),$$

a polynomial which has $n - k$ successive powers of β as roots.
The cyclic code one obtains is an $[n, k, d]-$code over $\mathbb{F}_q$ with dimension k and minimum distance $d = n - k + 1$; it is a so-called maximum distance separable (MDS) code.
A word p of length n, again interpreted as polynomial, is a codeword if the following condition holds:

$$p = p(x) \in C \Leftrightarrow p(\beta^b) = p(\beta^{b+1}) = \cdots = p(\beta^{b+d-2}) = 0.$$

With the observation that evaluating the polynomial $p(x) = p_0 + p_1 x + \cdots + p_{n-1}x^{n-1}$ at β^j for $j = b, \ldots, b+d-2$, and interpreting the result as a vector $(p(\beta^b), p(\beta^{b+1}), \ldots, p(\beta^{b+d-2}))$, is exactly the same as multiplying the vector $p = (p_0, \ldots, p_{n-1})$ with the matrix

$$H = \begin{pmatrix} 1 & \beta^b & \cdots & \beta^{(n-1)b} \\ 1 & \beta^{b+1} & \cdots & \beta^{(n-1)(b+1)} \\ \vdots & \vdots & & \vdots \\ 1 & \beta^{b+d-2} & \cdots & \beta^{(n-1)(b+d-2)} \end{pmatrix}^T,$$

one can see that this matrix H is a parity check matrix of the RS code C.
RS codes can be defined in an alternative way using the evaluation function

$$f_C : \begin{cases} \mathbb{F}_{q,k}[x] \to \mathbb{F}_q^n \\ p(x) \mapsto (p(1), p(\beta), \ldots, p(\beta^{q-2})) \end{cases}$$

where $\mathbb{F}_{q,k}[x]$ denotes the set of polynomials over $\mathbb{F}_q$ of degree less than k.
A drawback of the RS codes often criticized is that the code length depends on the alphabet. This can be avoided if another class of codes, a first generalization, is introduced — the BCH codes.

1.3.3 BCH Codes

BCH codes, named after Bose, Chaudhuri, Hocquenghem, do not have the limitation that the code length is fixed by the alphabet. Instead of taking a primitive element of the alphabet $A := \mathbb{F}_q$, choose a non-negative integer n, $\gcd(n, q) = 1$ and let β be a primitive $n-$th root of unity in a certain field extension $\mathbb{F}_{q^m}$ of the alphabet $\mathbb{F}_q$ of degree $m \geq 1$. Here the minimal $m \geq 1$ is $ord(q) \mod n$, the multiplicative order of q in $\mathbb{Z}_n^\times$. Then we have $q^m \equiv 1 \mod n$, or equivalently $n | q^m - 1$. The polynomial

$$x^n - 1 = \prod_{i=1}^{n} (x - \beta^i) \tag{1.3}$$

can be written as a product of linear factors in $\mathbb{F}_{q^m}$.
We mention that the choice $n := q - 1, m := 1$ leads to RS codes. If $n := q^m - 1$ the BCH code is called primitive.
To construct a BCH code, we need to fix a non-negative integer b and the designed distance δ to obtain the largest possible cyclic code C having zeros at $\beta^b, \beta^{b+1}, \ldots, \beta^{b+\delta-2}$.

Definition 1.8. *Let $\mu^{(i)}(x)$ denote the minimal polynomial of β^i over $\mathbb{F}_q$ and b, δ as before. The generator polynomial*

$$g(x) := lcm(\mu^{(b)}(x), \mu^{(b+1)}(x), \ldots, \mu^{(b+\delta-2)}(x))$$

defines a cyclic code — a BCH code.

Because of Equation (1.3), $g(x) | x^n - 1$ holds.
In the case when $b = 1$, C is called narrow-sense BCH code.

Theorem 1.9. *The dimension of the code BCH C is $k = n - \deg g(x) \geq n - (\delta - 1)m$ and its minimum distance d is at least as big as the defined minimum distance δ.*

A proof of this theorem, along with more interesting facts and examples about BCH codes can be found in the lecture notes [13, Ch. 1.11].

Definition 1.10. *A sequence of codes $(C_i)_{i \in \mathbb{N}}$ over $\mathbb{F}_q$ with parameters $[n_i, k_i, d_i], i \in \mathbb{N}$ is called good if*

$$\lim_{i \to \infty} n_i = \infty, \ \lim_{i \to \infty} R_i = \lim_{i \to \infty} k_i / n_i > 0, \ \lim_{i \to \infty} d_i / n_i > 0,$$

in words, if both — the information rate and the relative minimum distance — approach a non-zero limit as $i \to \infty$.
A family of codes is called bad if it is not good.

This definition, along with a proof of the theorem that any sequence of BCH codes is bad, is given in [22, (Ch. 9 § 5)]. Soon we will encounter a class of good codes.

1.3.4 GRS Codes

Following [22, (Ch. 10 § 8)] let $v := (v_1, v_2, \ldots, v_n)$ be non-zero elements of $\mathbb{F}_q$ and let $\alpha := (\alpha_1, \alpha_2, \ldots, \alpha_n)$ contain pairwise distinct elements of $\mathbb{F}_{q^m}$.

Definition 1.11. *The generalized RS code (GRS) is given by*

$$GRS_k(\alpha, v) := \{(v_1 F(\alpha_1), v_2 F(\alpha_2), \ldots, v_n F(\alpha_n)) \in \mathbb{F}_{q^m}^n \mid F(x) \in \mathbb{F}_{q^m,k}[x]\},$$

weighted evaluations of polynomials $F(x) \in \mathbb{F}_{q^m}[x]$ of degree $\deg F < k$.

This construction yields an $[n, k, d]-$MDS code. A parity check matrix is given by

$$H = \begin{pmatrix} y_1 & y_2 & \cdots & y_n \\ y_1\alpha_1 & y_2\alpha_2 & \cdots & y_n\alpha_n \\ y_1\alpha_1^2 & y_2\alpha_2^2 & \cdots & y_n\alpha_n^2 \\ \vdots & \vdots & \vdots & \vdots \\ y_1\alpha_1^{n-k-1} & y_2\alpha_2^{n-k-1} & \cdots & y_n\alpha_n^{n-k-1} \end{pmatrix} = \tag{1.4}$$

$$= \begin{pmatrix} 1 & 1 & \ldots & 1 \\ \alpha_1 & \alpha_2 & \ldots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \ldots & \alpha_n^2 \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_1^{n-k-1} & \alpha_2^{n-k-1} & \ldots & \alpha_n^{n-k-1} \end{pmatrix} \cdot \begin{pmatrix} y_1 & 0 & 0 & \ldots & 0 \\ 0 & y_2 & 0 & \ldots & 0 \\ 0 & 0 & y_3 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & y_n \end{pmatrix} \tag{1.5}$$

where $(y_1, \ldots, y_n)$ is a vector depending on $(v_1, \ldots, v_n)$,see Theorem (1.12).
We remark that for the choices $n = q - 1, k \leq n$, $v_1 = v_2 = \cdots = v_n = 1$ and the parameter b to define $\alpha_i = \beta^{b+i}, i = 1, 2, \ldots, n$ where β is primitive in $\mathbb{F}_q$, one gets the classical RS codes (see Section 1.3.2) over $\mathbb{F}_q$.

Theorem 1.12. *The dual of a generalized Reed-Solomon code of length n is*

$$GRS_k(\alpha, v)^\perp = GRS_{n-k}(\alpha, y)$$

for a suitable vector y.

A proof of how to construct the vector y can also be found in [22, (Ch. 10 § 8)]. This vector plays an important role in the definition of alternant codes in the next section.

1.3.5 Alternant Codes

Before giving a description of alternant codes and their properties and capabilities we have to discuss the subfield subcode construction.

Subfield Subcodes

Following MacWilliams and Sloane [22, (Ch. 7 § 7)] given non-negative integers $r \leq n$, and a prime power q, define $m \in \mathbb{N}$ such that $rm \leq n$ holds.
With this notation we want to look at parity check matrices and thus codes defined over the (big) field $\mathbb{F}_{q^m}$ and study codes that are restricted to the (smaller) subfield $\mathbb{F}_q \leq \mathbb{F}_{q^m}$.
Let H be a parity check matrix of a code $\hat{C}_H$. This means $H = (H_{ij})$, with elements $H_{ij} \in \mathbb{F}_{q^m}$, for $i = 1, \ldots, r; j = 1, \ldots, n$, is an $(r \times n)$ matrix over $\mathbb{F}_{q^m}$ with full rank r.
We are now interested in two codes with respect to H:

$$\hat{C}_H := \left\{ b = (b_1, b_2, \ldots, b_n) \in \mathbb{F}_{q^m}^n \mid Hb^T = 0 \right\} \leq \mathbb{F}_{q^m}^n, \tag{1.6}$$

$$C_H := \left\{ a = (a_1, a_2, \ldots, a_n) \in \mathbb{F}_q^n \mid Ha^T = 0 \right\} \leq \mathbb{F}_q^n. \tag{1.7}$$

This means $\hat{C}_H$ consists of all vectors $b = (b_1, b_2, \ldots, b_n), b_i \in \mathbb{F}_{q^m}$ with $Hb^T = 0$ and is thus a $[n, \hat{k}, \hat{d}]-$code over $\mathbb{F}_{q^m}$ with $\hat{k} = n - r$ ($\hat{d}$ cannot be determined in this general setting), whereas $C_H = \hat{C}_H \cap \mathbb{F}_q^n$ consists of all vectors $a = (a_1, a_2, \ldots, a_n), a_i \in \mathbb{F}_q$ with $Ha^T = 0$. An alternative way to obtain C_H as well as the code parameters is to define a matrix $\bar{H}$. Upon fixing basis elements $\alpha_1, \ldots, \alpha_m$ of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$, one gets the $(rm \times n)$ matrix $\bar{H}$ by replacing each entry H_{ij} by the column vector $\vec{h}_{ij}$, so that

$$H_{ij} = \sum_{l=1}^{m} \alpha_l h_{ijl}, \quad \vec{h}_{ij} = \begin{pmatrix} h_{ij1} \\ h_{ij2} \\ \vdots \\ h_{ijm} \end{pmatrix} \in \mathbb{F}_q^m$$

holds. With this, we can characterize the codewords in C_H. For a word $a = (a_1, a_2, \ldots, a_n) \in \mathbb{F}_q^n$ we have the following:

$$\begin{aligned} a \in C_H &\Leftrightarrow Ha^T = 0 \\ &\Leftrightarrow \sum_{j=1}^{n} H_{ij} a_j = 0 \quad i = 1, \ldots, r \\ &\Leftrightarrow \sum_{j=1}^{n} h_{ijl} a_j = 0 \quad i = 1, \ldots, r;\ l = 1, \ldots, m \\ &\Leftrightarrow \bar{H} a^T = 0. \end{aligned}$$

Thus H and $\bar{H}$ define the same code C_H. Since the rank of $\bar{H}$ over $\mathbb{F}_q$ is at most rm, we identify C_H as dimension $k \geq n - rm$ code. By definition the $(r \times n)$ matrix H has rank r. The null-space of the matrix, which is the code C_H with parity check matrix H, has dimension $k \leq n - r$.
To see $C_H \subseteq \hat{C}_H$ is easy, in fact by definition C_H contains only those codewords of $\hat{C}_H$ where each entry is in $\mathbb{F}_q$.

Definition 1.13. *We write*

$$C_H = \hat{C}_H | \mathbb{F}_q$$

and call C_H a subfield subcode of $\hat{C}_H$ or the restriction of $\hat{C}_H$ to $\mathbb{F}_q$.

To summarize the results, in general the $[n, k, d]-$subfield subcode C_H of the $[n, \hat{k}, \hat{d}]-$code $\hat{C}_H$ has the parameter $d \geq \hat{d}$ and the dimension $n - (n - \hat{k})m \leq k \leq \hat{k}$.

Definition of Alternant Codes

For two non-negative integers $K, n,\ K \leq n$ an alternant code is the subfield subcode of $GRS_K(\alpha, v)$, with α, v as in Section 1.3.4. More specific the alternant code

$$\mathcal{A}(\alpha, y) := \{c = (c_1, c_2, \ldots, c_n) \in GRS_K(\alpha, v) \mid c_i \in \mathbb{F}_q, i = 1, \ldots n\} = GRS_K(\alpha, v)|\mathbb{F}_q,$$

where y is the vector from Theorem 1.12 such that

$$GRS_K(\alpha, v)^{\perp} = GRS_{n-K}(\alpha, y).$$

Another characterization is that $\mathcal{A}(\alpha, y)$ consists of all vectors $c \in \mathbb{F}_q^n$ such that $Hc^T = 0$, where H is a parity check matrix of $GRS_K(\alpha, v)$, also compare to Equation (1.4):

$$H = \begin{pmatrix} y_1 & y_2 & \ldots & y_n \\ y_1\alpha_1 & y_2\alpha_2 & \ldots & y_n\alpha_n \\ y_1\alpha_1^2 & y_2\alpha_2^2 & \ldots & y_n\alpha_n^2 \\ \vdots & \vdots & \vdots & \vdots \\ y_1\alpha_1^{n-k-1} & y_2\alpha_2^{n-k-1} & \ldots & y_n\alpha_n^{n-k-1} \end{pmatrix}. \tag{1.8}$$

This identifies $\mathcal{A}(\alpha, y)$ as an $[n, k, d]-$code over $\mathbb{F}_q$ with $k \geq n - mr, r := n - K$ and $d \geq r + 1$. We remark that the class of alternant codes, and therefore especially the class of Goppa codes, is good as in Definition (1.10).

1.4 Goppa Codes

In this section we deal with the class of classical Goppa codes. Sometimes in older literature the name Goppa codes is used for codes that nowadays are usually called algebraic-geometric (AG) codes. Although they are interesting and experts believe they will be deployed in practice soon, we will not discuss AG codes in detail.
Nevertheless, Goppa codes can be seen as subfield subcodes from the modern, more general definition of AG codes as is elaborated in [16] by Høholdt and Pellikaan. There, the authors further state that, in order to fully understand this algebraic-geometric view on codes, it takes "5 years to grasp for an outsider".
The same authors also think that the more general class of AG codes will be implemented for practical purposes only when decoding algorithms as fast as Euclid's algorithm with a time complexity of $\mathcal{O}(n^2)$, where n is the codeword length, will be found.
To date, with exception for some special cases, most algorithms for decoding AG codes have a complexity of $\mathcal{O}(n^3)$, which is considered impractical.
However, here we present the classical approach to define Goppa codes and have a look at a suitable decoding algorithm for binary codes.

According to the seminal work of McEliece [24] classical Goppa codes — especially those defined over the binary alphabet $\mathbb{F}_2$ — seem to be best suited for applications in cryptography and will therefore be discussed in further detail in Section 1.4.2.

1.4.1 Classical Goppa Codes

Now we consider the classical Goppa code with Goppa polynomial $G(z) \in \mathbb{F}_{q^m}[z]$ of degree $r := \deg G(z)$ and locator set $L = \{\alpha_1, \alpha_2, \ldots, \alpha_n\} \subseteq \mathbb{F}_{q^m}$, where $m \geq 1$ is a fixed non-negative integer. The so-called locator set (or support) L can be as big as $\mathbb{F}_{q^m}$, which is a common choice, as long as $G(\alpha_i) \neq 0$, for all $\alpha_i \in L$.

Definition 1.14. *The Goppa code $\Gamma := \Gamma(L, G)$ with support L and Goppa polynomial $G(z)$ is defined by*

$$\Gamma := \left\{ c = (c_1, c_2, \ldots, c_n) \in \mathbb{F}_q^n \mid R_c(z) := \sum_{i=1}^{n} \frac{c_i}{z - \alpha_i} \equiv 0 \mod G(z) \right\}.$$

It can easily be seen that Γ is a linear $[n, k, d]-$code over the field $\mathbb{F}_q$ since the sum of two codewords $c + \bar{c}$ and scalar multiples $a \cdot c, a \in \mathbb{F}_q$ fulfill $R_{c+\bar{c}} \equiv 0$ respectively $R_{ac} \equiv 0$, when we define $R_c(z) := \sum_{i=1}^{n} \frac{c_i}{z-\alpha_i}$ as before.
Upon receiving the word $x = c + e = (x_1, x_2, \ldots, x_n)$, which can be seen as a codeword and errors added at unknown positions, the syndrome polynomial is defined by

$$S(z) := R_x(z) = \sum_{i=1}^{n} \frac{x_i}{z - \alpha_i}. \tag{1.9}$$

From this definition follows that $x \in \Gamma \Leftrightarrow S(z) = R_x(z) = R_c(z) + R_e(z) \equiv 0 \mod G(z)$. Using Patterson's algorithm (from Section 1.4.2) the decoding $x \mapsto c$ can be done efficiently, if $w(e) \leq t$ errors occurred.

Goppa Codes are Alternant Codes

Goppa codes are an important subclass of alternant codes, which can be seen after some calculations (see [22, (Ch. 12 § 3)]).
For fixed i, α_i is no zero of the Goppa polynomial $G(z) = \sum_{i=0}^{r} g_i z^i$ with $g_i \in \mathbb{F}_{q^m}, g_r \neq 0$, therefore $z - \alpha_i$ has an inverse $\mod G(z)$. Since $z = \alpha_i$ is a zero of $G(\alpha_i) - G(z)$

$$(z - \alpha_i)^{-1} := \frac{G(\alpha_i) - G(z)}{z - \alpha_i} G(\alpha_i)^{-1} \in \mathbb{F}_{q^m}[z],$$

is a polynomial because the nominator can be divided in this domain. Finally

$$(z - \alpha_i) \cdot (z - \alpha_i)^{-1} = (z - \alpha_i) \cdot \frac{G(\alpha_i) - G(z)}{z - \alpha_i} G(\alpha_i)^{-1} \equiv 1 \mod G(z)$$

holds. The expression $\frac{G(\alpha_i)-G(z)}{z-\alpha_i} G(\alpha_i)^{-1} = -\frac{G(z)-G(\alpha_i)}{z-\alpha_i} G(\alpha_i)^{-1} \in \mathbb{F}_{q^m}[z]$ is a polynomial of degree $\deg G(z) - 1 = r - 1$ for all $i = 1, \ldots, n$ since a degree one polynomial has been

divided from a degree r polynomial. We have:

$$\begin{aligned} c = (c_1, c_2, \ldots, c_n) \in \Gamma &\Leftrightarrow \sum_{i=1}^{n} \frac{c_i}{z - \alpha_i} = 0 \mod G(z) \\ &\Leftrightarrow G(z) \sum_{i=1}^{n} \frac{c_i G(\alpha_i)^{-1}}{z - \alpha_i} - \sum_{i=1}^{n} \frac{c_i}{z - \alpha_i} G(\alpha_i)^{-1} G(\alpha_i) = 0 \mod G(z) \\ &\Leftrightarrow \sum_{i=1}^{n} c_i \frac{G(z) - G(\alpha_i)}{z - \alpha_i} G(\alpha_i)^{-1} = 0 \in \mathbb{F}_{q^m}[z]. \end{aligned}$$

Since the degree of the left hand side is $\leq r - 1$ the condition to be zero as a polynomial is equivalent to be zero $\mod G(z)$.
Pellikaan et al. [30, Proposition 8.3.9] show the following theorem:

Theorem 1.15. *Let $G(z) = \sum_{i=0}^{r} g_i z^i$ with $g_i \in \mathbb{F}_{q^m}, g_r \neq 0$ be the Goppa polynomial to construct $\Gamma := \Gamma(L, G)$ with $L = \{\alpha_1, \alpha_2, \ldots, \alpha_n\} \subseteq \mathbb{F}_{q^m}$.*
Then the resulting Goppa code $\Gamma(L, G) = \mathcal{A}(\alpha, y)$ is an alternant code with parameters $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ of fixed order and $y = (G(\alpha_1)^{-1}, G(\alpha_2)^{-1}, \ldots, G(\alpha_n)^{-1})$.

Proof. For all $i = 1, 2, \ldots, n$ the polynomial can be written as

$$\begin{aligned} \frac{G(z) - G(\alpha_i)}{z - \alpha_i} &= \sum_{l=0}^{r} \frac{g_l (z^l - \alpha_i^l)}{z - \alpha_i} = \sum_{l=0}^{r} g_l \sum_{j=0}^{l-1} z^j \alpha_i^{l-1-j} \\ &= \sum_{j=0}^{r-1} \left(\sum_{l=j+1}^{r} g_l \alpha_i^{l-1-j} \right) z^j. \end{aligned}$$

Hence equating the coefficients of z^j to zero and using the remarks from above, one sees

$$\begin{aligned} c = (c_1, c_2, \ldots, c_n) \in \Gamma &\Leftrightarrow \sum_{i=1}^{n} c_i \left(\sum_{l=j+1}^{r} g_l \alpha_i^{l-1-j} \right) G(\alpha_i)^{-1} = 0, \quad j = 0, 1, \ldots, r-1 \\ &\Leftrightarrow \bar{H} \cdot c^T = \big(h_1, h_2, \ldots, h_i, \ldots, h_n\big) \cdot c^T = 0, \end{aligned}$$

$$h_i := G(\alpha_i)^{-1} \begin{pmatrix} g_r & 0 & 0 & \ldots & 0 \\ g_{r-1} & g_r & 0 & \ldots & 0 \\ g_{r-2} & g_{r-1} & g_r & & 0 \\ \vdots & & & \ddots & \vdots \\ g_1 & g_2 & g_3 & \ldots & g_r \end{pmatrix} \cdot \begin{pmatrix} 1 \\ a_i^1 \\ \vdots \\ a_i^{r-2} \\ a_i^{r-1} \end{pmatrix}, \quad i = 1, 2, \ldots, n.$$

$\bar{H}$ is a $r \times n$ parity check matrix of the form $\bar{H} = CXY$, with the invertible, lower triangle shaped matrix C, the Vandermonde matrix X and the diagonal matrix Y:

$$\bar{H} = \begin{pmatrix} g_r & 0 & \ldots & 0 \\ g_{r-1} & g_r & \ldots & 0 \\ \vdots & & \ddots & \vdots \\ g_1 & g_2 & \ldots & g_r \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & \ldots & 1 \\ \alpha_1 & \alpha_2 & \ldots & \alpha_n \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_1^{r-1} & \alpha_2^{r-1} & \ldots & \alpha_n^{r-1} \end{pmatrix} \cdot \begin{pmatrix} G(\alpha_1)^{-1} & 0 & \ldots & 0 \\ 0 & G(\alpha_2)^{-1} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & G(\alpha_n)^{-1} \end{pmatrix}.$$

To see that the Goppa code with this parity check matrix is an alternant code, we have to multiply $\bar{H}$ with the inverse of the invertible matrix C to get another parity check matrix for the code Γ. Comparing the parity check matrices $H := C^{-1} \cdot \bar{H}$ in (1.10) with the structure of (1.8) we see that $\Gamma(L, G) = \mathcal{A}(\alpha, y)$ with $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ of fixed order and $y = (G(\alpha_1)^{-1}, G(\alpha_2)^{-1}, \ldots, G(\alpha_n)^{-1})$. □

Either the matrix $\bar{H}$ from the proof of Theorem (1.15) can be used as a parity check matrix for the Goppa code or $H := C^{-1} \cdot \bar{H}$, which we write down explicitly once more:

$$H = \begin{pmatrix} G(\alpha_1)^{-1} & G(\alpha_2)^{-1} & \ldots & G(\alpha_n)^{-1} \\ \alpha_1 G(\alpha_1)^{-1} & \alpha_2 G(\alpha_2)^{-1} & \ldots & \alpha_n G(\alpha_n)^{-1} \\ \alpha_1^2 G(\alpha_1)^{-1} & \alpha_2^2 G(\alpha_2)^{-1} & \ldots & \alpha_n^2 G(\alpha_n)^{-1} \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_1^{r-1} G(\alpha_1)^{-1} & \alpha_2^{r-1} G(\alpha_2)^{-1} & \ldots & \alpha_n^{r-1} G(\alpha_n)^{-1} \end{pmatrix}. \tag{1.10}$$

As discussed in Section 1.3.5, Γ is an $[n, k, d]-$code with length $n = |L|$, dimension $k \geq n - mr$ and minimal distance $d \geq r + 1$. To explicitly put this in the $[n, k, d]-$form, $\Gamma := \Gamma(L, G)$ is a linear $[n = |L|, k \geq n - m \cdot \deg G, d \geq \deg G + 1]-$code.
As Berlekamp summarized Goppa's work in [2], the q-ary Goppa code Γ can decode t errors if $\deg G(z) = 2t$. If Γ is a binary Goppa code this result can be improved. In this case the Goppa polynomial $G(z)$ need only have degree t and no repeated irreducible factors to be capable of decoding up to t errors. We will present the result now and give detailed coverage about the decoding of binary Goppa codes in Section 1.4.2.

Binary Goppa Codes

In [22, (Ch. 12 § 3) Theorem 6] MacWilliams and Sloane note the following result.

Theorem 1.16. *Given a Goppa polynomial $G(z)$ of degree $t := \deg G(z)$ that has no multiple zeros, so that the lowest degree perfect square $\bar{G}(z)$ that is divisible by $G(z)$ is $\bar{G}(z) = G(z)^2$, then the Goppa code $\Gamma(L, G)$ is a $[n, k, d \geq 2t + 1]-$code.*

We will use this result and give an efficient decoding algorithm for a Goppa polynomial $G(z)$ of degree $t := \deg G(z)$ that is able to correct up to t errors in Section 1.4.2. In Chapter 3 a binary Goppa code is used to construct a cryptosystem using the computer algebra system Sage. We demonstrate the error correcting capability of the code there.

1.4.2 Decoding Algorithms

Decoding of alternant codes, and therefore especially decoding of classical Goppa codes, consists of three stages (see [22, (Ch. 12 § 9)]).
Let $\mathcal{A}(\alpha, v)$ be an alternant code over $\mathbb{F}_q$ with parity check matrix H of the form (1.4) and with minimum distance d. As usual, the vector $v := (v_1, v_2, \ldots, v_n)$ consists of non-zero elements of $\mathbb{F}_q$ and $\alpha := (\alpha_1, \alpha_2, \ldots, \alpha_n)$ contains pairwise distinct elements.
Next we suppose that $t \leq \lfloor \frac{d-1}{2} \rfloor$ errors have occurred in the coordinates $1 \leq j_1 < j_2 < \cdots < j_t \leq n$ of a received word x the with error values $Y_1, \ldots, Y_t \in \mathbb{F}_q$, respectively.
The definitions $X_i := \alpha_{j_i}, \quad i = 1, \ldots, t$ and $r := n - k$ abbreviate the following formulas.

Assume, as usual, the word $x = c + e$ with a weight t error vector e is received and we are interested in reconstructing the codeword c.

Stage 1) The syndrome is computed using the parity check Matrix H from Equation (1.8), which is dependent on both vectors α and $y = (y_1, y_2, \ldots, y_n)$. Recall the connection of v and y from the construction of the dual GRS code.

$$(S_0, S_1, \ldots, S_{r-1}) = x \cdot H^T = e \cdot H^T = \overbrace{(0, \ldots, 0, \underbrace{Y_1}_{\text{pos. } j_1}, \ldots)}^{\text{error vector}} \cdot H^T$$

It is useful to define the polynomial $S(z) := \sum_{j=0}^{r-1} S_j z^j$.

Stage 2) Find the coefficients of both, the error locator polynomial σ

$$\sigma(z) := \prod_{i=1}^{t} (1 - X_i z)$$

and the error evaluator polynomial ω

$$\omega(z) := \sum_{k=1}^{t} Y_k y_{j_k} \prod_{i \neq k} (1 - X_i z)$$

so that they satisfy

$$\frac{\omega(z)}{\sigma(z)} \equiv S(z) \mod z^r \tag{1.11}$$

with the syndrome polynomial $S(z)$ computed in Stage 1.

Equation (1.11) is called "key equation" of the decoding process of alternant codes.

The key equation can be solved uniquely using a modified Euclidean algorithm. Repeated polynomial divisions are carried out until certain degree conditions for the remainders are met (see [22, (Ch. 12 § 9) Theorem 16]).

Stage 3) Because $\sigma(X_i^{-1}) = 0$ for all $i = 1, 2, \ldots, t$, we see that the reciprocals of the roots of $\sigma(z)$ are, by definition, the error positions $X_i, i = 1, 2, \ldots, t$. The error locations can be found by performing a search that checks for all the field elements $\gamma \in \mathbb{F}_{q^m}$, whether $\sigma(\gamma) = 0$. We remark that more sophisticated algorithms, like Chien search, exist for this task.

The error values can be computed, which is only necessary for $q \neq 2$, because for $q = 2$ an error is easily corrected by flipping the erroneous bit.

In the general case the error values $Y_l, l = 1, \ldots, t$, are given by the following formula, sometimes referred to as Forney's Formula:

$$Y_l = \frac{\omega(X_l^{-1})}{\prod_{i \neq l}(1 - X_i X_l^{-1})} = -X_l \frac{\omega(X_l^{-1})}{y_{j_l} \sigma'(X_l^{-1})}.$$

To obtain this result, we have to express the formal derivative σ' of $\sigma(z) = \prod_{i=1}^{t}(1 - X_i z)$. It is computed as

$$\sigma'(z) = \sum_{k=1}^{t} -X_k \prod_{i \neq k}(1 - X_i z).$$

Evaluating $\omega(z) = \sum_{k=1}^{t} Y_k y_{j_k} \prod_{i \neq k}(1 - X_i z)$ at $z = X_l^{-1}$ yields

$$\omega(X_l^{-1}) = Y_l y_{j_l} \prod_{i \neq l}(1 - X_i X_l^{-1})$$
$$\Leftrightarrow Y_l = \frac{\omega(X_l^{-1})}{y_{j_l} \prod_{i \neq l}(1 - X_i X_l^{-1})} = -X_l \frac{\omega(X_l^{-1})}{y_{j_l} \sigma'(X_l^{-1})}.$$

Since in this thesis we focus on the binary case for various reasons, the decoding becomes easier as we will see in the next section.

Patterson's Algorithm

Patterson was the first to show that for a general Goppa polynomial $G(z)$, where no further assumptions need to be made about $G(z)$, there exists an efficient algebraic decoding scheme for the corresponding Goppa code Γ. He proposed a decoding algorithm with a good running time, as we will present later in this section (see 1.4.2).
If the alphabet is binary, the description of Goppa codes gets easier and also the decoding method presented here has an advantage over the generic case.
Following [7, (§ 1.2.4)] after fixing n, m and $t < \frac{n}{m} \Leftrightarrow tm < n$, choose a monic irreducible polynomial $G(z) \in \mathbb{F}_{2^m}[z], \deg G = t$. Using this generator polynomial $G(z)$, which is assumed to have no multiple zeros, and defining the Goppa code Γ as usual yields a binary code with the parameters $[n, k \geq n - tm, d \geq 2t + 1]$. This is a direct application of Theorem 1.16.
Now for a received word $x = c + e, w(e) \leq t$, we want to compute the syndrome polynomial $S(z)$. We could either use the generic key equation (1.11) for alternant codes, or Equation (1.12), since we have seen that they correspond to each other in Theorem 1.15. The latter equation is more practical for the special case of Goppa codes and uses the definition of the syndrome as in (1.9) to compute a solution to the key equation:

$$\frac{\omega(z)}{\sigma(z)} \equiv S(z) \mod G(z). \tag{1.12}$$

It consists of three stages as discussed in the generic case for alternant codes, but becomes easier since binary Goppa codes are used. More specific it is best to take the syndrome polynomial $S(z)$ as defined in Equation (1.9) and as Barreto, Lindner and Misoczki [29] pointed out, use what they call "Patterson locator polynomial" — a slightly altered definition of the error locator polynomial

$$\sigma(z) := \prod_{i=1}^{t}(z - X_i).$$

This polynomial has roots at $X_i, i = 1, 2, \ldots, t$ instead of their reciprocals. Here, using the Patterson locator polynomial, the error evaluator polynomial

$$\begin{aligned}\omega(z) \equiv S(z)\sigma(z) \mod G(z) &= \sum_{j=1}^{t} \frac{Y_j}{z - X_j} \prod_{i=1}^{t} (z - X_i) \\ &= \sum_{j=1}^{t} \prod_{i \neq j} (z - X_i) = \sigma'(z)\end{aligned}$$

is the derivative of $\sigma(z) = \prod_{i=1}^{t}(z - X_i)$, since every error value has to be 1. The task now is to solve the equation $\sigma'(z) \equiv S(z)\sigma(z) \mod G(z)$ for unknown $\sigma(z)$.
By decomposing $\sigma(z) = u(z)^2 + zv(z)^2$ in "even" and "odd" parts, it is obvious that the polynomials $u(z), v(z)$ satisfy $\deg u(z) \leq \lfloor \frac{t}{2} \rfloor$ and $\deg v(z) \leq \lfloor \frac{t-1}{2} \rfloor$ since $\deg \sigma(z) = t$. The derivative of the error locator polynomial simplifies to $\sigma'(z) = 2u(z)u'(z) + v(z)^2 + 2zv(z)v'(z) = v(z)^2$ here, because the characteristic of $\mathbb{F}_{2^m}$ is 2.
Thus one can write

$$\omega(z) = S(z) \underbrace{(u(z)^2 + zv(z)^2)}_{\sigma(z)} \equiv \sigma'(z) \mod G(z) \tag{1.13}$$

$$\equiv v(z)^2 \mod G(z). \tag{1.14}$$

To solve Equation (1.14) for the unknown polynomial $\sigma(z)$, first compute the inverse of the syndrome $T(z) \equiv S(z)^{-1} \mod G(z)$ using the extended Euclidean algorithm (EEA) for polynomials. If we compute the greatest common divisor of $S(z)$ and $G(z)$ it is clear that the answer will be $\gcd(S(z), G(z)) = 1$, since $\deg S(z) < \deg G(z) = t$ and the Goppa polynomial is assumed to be irreducible. Because this is already known in advance, the EEA is only used to compute polynomials $x(z), y(z)$, such that $x(z)S(z) + y(z)G(z) = \gcd(S(z), G(z)) = 1$. Regarding this equation $\mod G(z)$ shows $T(z) := x(z)$ is the inverse of $S(z) \mod G(z)$. With this polynomial we can write

$$u(z)^2 \equiv (T(z) + z)v(z)^2 \mod G(z).$$

Next we need to compute a square root of $T(z) + z$. This is a polynomial $r(z)$ with the property that $r(z)^2 = T(z) + z \mod G(z)$. Algebraically the square root can be computed by decomposing $T(z) + z = T_0(z)^2 + zT_1(z)^2$. Given a fixed Goppa polynomial $G(z)$ it is sufficient to once and for all compute $w(z)^2 \equiv z \mod G(z)$. To obtain the square root, the next step is to compute $r(z) := T_0(z) + w(z)T_1(z)$. Since

$$r(z)^2 = (T_0(z) + w(z)T_1(z))^2 = T_0(z)^2 + w(z)^2 T_1(z)^2 \equiv T(z) + z \mod G(z)$$

holds here over $\mathbb{F}_{2^m}$, this is a solution.
Finally, a modified version of EEA that stops the computation when the polynomials for expressing the gcd reach a certain degree helps to obtain two polynomials $u(z), v(z)$ with

$$u(z) \equiv r(z)v(z) \mod G(z), \tag{1.15}$$

$$\deg u(z) \leq \left\lfloor \frac{t}{2} \right\rfloor, \deg v(z) \leq \left\lfloor \frac{t-1}{2} \right\rfloor. \tag{1.16}$$

We also refer to the example in Chapter 3, where the function called `modifiedEEA` provides both the functionality of the common extended Euclidean algorithm as well as the possibility to stop the calculation and thus solve Equation (1.15).
At this point — using a similar argument as above — we are again aware that the output of the common EEA that computes the greatest common divisor of $r(z)$ and $G(z)$ is $\gcd(r(z), G(z)) = 1$. We discuss solving (1.15) in greater detail now and follow the notation of [20]: The EEA computes a sequence of polynomials fulfilling the recursion

$$r_{h-1} = q_{h+1}r_h + r_{h+1}, \quad h = 0, 1, \ldots, s-1,$$

and stops for some s to return the sought after $\gcd(r_{-1}, r_0) = r_s$, where $r_0 := r(z)$ and $r_{-1} := G(z)$ in our case. The degrees of the residue polynomials satisfy

$$\deg r_{h+1} < \deg r_h, \quad h = 0, 1, \ldots, s.$$

Observe that the degrees in the sequence drop from $\deg G(z) = t$ at the beginning to 0 in the last step, where $r_{s-1} = q_{s+1}r_s + r_{s+1} = q_{s+1}r_s$ holds. Therefore there exists a smallest integer $j, 0 \leq j \leq s$ such that $\deg r_j \leq \frac{t}{2}$. Apart from the necessary computation of the $r_h, h = 0, 1, \ldots, s-1$ in order to obtain the gcd, it is clever to keep track of a second sequence during the Euclidean algorithm as well.
To compute solutions to Equation (1.15) while respecting the degree restrictions it is sufficient to calculate only the first j elements in the sequence

$$z_h = z_{h-2} - q_h z_{h-1}, \quad h = 1, 2, \ldots, j,$$

where $z_0 := 1$ and $z_{-1} := 0$.
These sequences relate in the following way (see [20, Chapter 8, Ex. 8.43]):

$$\begin{aligned} r_h &\equiv z_h r_0 \mod r_{-1}, \quad h = -1, 0, \ldots, s, \\ \deg z_h &= \deg r_{-1} - \deg r_{h-1}, \quad h = 0, 1, \ldots, s. \end{aligned}$$

The mentioned modification to the common EEA to perform this task, is to stop the algorithm after the minimal index j is reached. Then the previous element in the sequence satisfies $\deg r_{j-1} > \frac{t}{2}$ and therefore we have

$$\deg z_j = \deg G - \deg r_{j-1} = t - \deg r_{j-1} < \frac{t}{2}.$$

Because j was the minimal index with the property that the degree of $u(z) := r_j(z)$ drops below $\frac{t}{2}$ and $\deg z_j \leq \left\lfloor \frac{t-1}{2} \right\rfloor < \frac{t}{2}$ holds, we set $v(z) := z_j(z)$ to satisfy both equations, Equation (1.15) and Equation (1.16).
Hence the `modifiedEEA` that checks these degree conditions yields the desired solution.
The aim of this section is achieved since $\sigma(z) = u(z)^2 + zv(z)^2$ is fully determined, which enables us to present the following listing based on the computation steps discussed above and known as Patterson's algorithm.
Computing the error positions e is sufficient, since flipping bits corrects the errors and yields $c = x + e$. As remarked earlier, the computationally hardest part of Algorithm 1 is the root finding, which thus is a topic in current research.

Algorithm 1: The decoding process of binary Goppa codes

Input : Received vector $x = c + e$, the binary Goppa code $\Gamma(L, G)$.
Output: Error positions e.

Compute syndrome $S(z)$ for the word x.
$T(z) \leftarrow S(z)^{-1} \mod G(z)$
if *T(z)=z* **then**
  $\sigma(z) \leftarrow z$
else
  $r(z) \leftarrow \sqrt{T(z) + z} \mod G(z)$
  Compute $u(z), v(z)$ from $u(z) = v(z)r(z) \mod G(z)$
  $\sigma(z) \leftarrow u(z)^2 + zv(z)^2$
end
Determine the roots of $\sigma(z)$ and therefore get e.
return $c \leftarrow x + e$.

Running Time Analysis of Patterson's Algorithm

Engelbert, Overbeck and Schmidt [9] summarize the running time of the previous steps necessary for the decoding of a binary $[n, k, d]-$Goppa code over $\mathbb{F}_{2^m}$. Thus assume that the Goppa polynomial has degree $\deg G(z) = t$ and the coefficients are in $\mathbb{F}_{2^m}$.

- Computation of the syndrome polynomial $S(z)$ using the parity check matrix H, takes $(n-k)n \in \mathcal{O}(n^2)$ binary operations.
- Computation of the inverse $T(z) = S(z)^{-1} \mod G(z)$ with the EEA takes $\mathcal{O}(t^2m^2)$ binary operations.
- Computation of $r(z) = \sqrt{T(z) + z} \mod G(z)$ with the EEA takes again $\mathcal{O}(t^2m^2)$ binary operations.
- In general also the modified EEA, which stops earlier than the common EEA, takes $\mathcal{O}(t^2m^2)$ binary operations.
- The running time of the last step — the search for all roots of $\sigma(z)$ — governs all the other running times. The search can be done in $n(tm^2 + tm)$ binary operations.

Since $(n-k) \leq mt$ the overall running time of Patterson's algorithm is $\mathcal{O}(ntm^2)$.

2 Cryptography

In the beginning of the first chapter we saw the information transmission process from a coding theoretic viewpoint. If we want to securely transfer data, we have to additionally introduce a step for encrypting the message block before transmitting it over an insecure channel. All the steps for error correcting are still needed, since every channel bears some sources of errors in practice. We observe that coding theory strikes up to three time when looking at the secure information transmission process from a cryptographic point of view:

Information $I \Rightarrow$ source (en)coding $\Rightarrow x = (x_1, x_2, \ldots, x_k) \in A^k$ message block $\Rightarrow$ **encrypt message** $x \mapsto z \Rightarrow$ channel (en)coding $\Rightarrow c = (c_1, c_2, \ldots, c_n) \in C \subseteq A^n$ codeword $\Rightarrow$ submission via noisy channel (this is where errors might alter the message) $\Rightarrow$ received message block $y = c + e \Rightarrow$ decoding $\Rightarrow \tilde{z} \approx z \Rightarrow$ **decrypt message** $\tilde{z} \mapsto \tilde{x}$ inverse source coding $\Rightarrow$ (received information) $\tilde{I} \approx I$ (information before transmission).

The term cryptography stands for information security in a wide sense nowadays. It deals with the protection of confidential information, with granting access to systems for authorized users only and providing a framework for digital signatures. Cryptographic methods and how to apply them, should not be something only couple of experts know about and just a slightly bigger group of people is able to use. In contrast, cryptography should become an easy to use technology for various fields of communication. It can be as easy as a lock symbol appearing in the Internet browser, when logging on to bank's website, signalizing a secure channel, for example. Generally speaking any information that needs to be electronically submitted or stored secretly could profit from an easy access to good cryptographic methods. On the contrary, there is an idiom that says:

Security is [easy, cheap, good]. Choose two.

The topic of privacy and protection against unwanted adversaries has never been more important. Thus it is desirable to make cryptography commonly accessible at least for a certain minimum security level.

Another interesting topic — out of scope here though — is the need for people to get aware of the fact where and which information about us is collected, processed and used everyday. Thus we not only have to describe and research ways to encrypt information but to create awareness and actually use the given methods to protect our personal information in a way that we agree. This is more of an general advice, but now we focus on the methods and algorithms again that are already deployed in practice and furthermore analyze to what extend nowadays security protocols and encryption algorithms fulfill their task.
There exist certain clearly defined use-case scenarios, where cryptography provides solutions. Although in big systems the weakest link of the chain are often the users, we keep the focus on the aspects we can control. In this respect we use mathematical methods as a tool to guarantee cryptographic strength of the underlying procedures.

How to find the right cryptographic primitives for one's needs?

First of all one has to think about what data might need protection. Then of course, what is the purpose of protecting the data. Questions like "Who might be interested in the data?" or "How long does the data need protection?" and "Are active or passive attacks a possible threat?" are considered in this next stage. Finally, the appropriate cryptographic methods and a suitable security level (compare with Section 2.5) is chosen and set up.

Code-based cryptography

Those were interesting questions but now we focus on the appearance of coding theory in cryptography in the following sections were we describe the influence of linear codes on public key primitives like public key cryptosystems, signatures and finally identification primitives.
The three areas of privacy, authentication and integrity are covered by modern cryptography. The aim of coding theory on the other hand described by one word is reliability. Both fields cover different areas in the field of digital information processing. It is interesting to explore how they interact in modern cryptography.
Additionally to the (linear) code used for encryption, other error correcting codes may and will be used for channel coding and even another code for source coding (as motivated in the overview above 1.1). This of course, will not lead to confusion, but it again points out the influence of different branches of coding theory in information processing.

Overview

In this chapter, we first make a short detour to theoretical computer science in Section 2.1. To be more precise we deal with computational complexity theory, because therein lies the cause that the mathematical theory can be applied to modern purposes in information processing.
In the Section 2.1.1 we concentrate on problems based on coding theory. From a complexity theoretic perspective this is a good choice, because other than the factorization of integers or the hidden subgroup problem (see 4.3.4), the decoding problem has been proved to be hard. The question why security based on this provable hard problem may still be not enough and the term "provable security" (see 2.1.1) are elaborated in detail there.

In the sections named Public Key Cryptography 2.2, Signatures 2.3 and Authentication 2.4 we address the three main topics of asymmetric cryptography. We will see that cryptography is not merely about encrypting messages. It provides solutions to many security-related aspects.
Additionally to security considerations in Section 2.5 we also concentrate on two public key cryptosystems that have drawn much attention in the scientific community and experts in cryptography. Other widely used cryptosystem are mentioned shortly and we point out their disadvantages. Facts about their performance are given and problems of those systems, as well as threats that exist, are addressed. Moreover possible improvements that have been discussed in the literature and recent papers on the topic are presented.

2.1 Complexity Theory

Complexity theory, as part of theoretical computer science, will help us in this section to find appropriate problems that can be used for good cryptographic schemes.
Now follows a short introduction to complexity theory to an extend that is needed in this work. We start with some definitions.

Definition 2.1. *$\mathcal{P}$ is defined as the class of deterministic polynomial time complexity problems.*

Heuristically, $\mathcal{P}$ can be described as the class of problems that gain moderately in difficulty when the input size is increased.
The demand for resources to solve a problem of size $n + 1$ is not too large compared to a problem of size n. Problems were an algorithm of the class $\mathcal{P}$ exist, are sometimes referred to be "easy", in the sense that computers can solve them using enough resources in a moderate time frame. It is "feasible" to compute the solutions for problems in $\mathcal{P}$.

Definition 2.2. *$\mathcal{NP}$ is defined as the class of problems that there exists a non-deterministic algorithm with polynomial time complexity of the input size.*

How to actually solve an instance of such an $\mathcal{NP}$ problem after an arguable amount of time, is not clear in the first place. Although a solution once found can be verified in polynomial time, an algorithm constructing a solution in a deterministic way usually requires a lot more time. As mentioned below no fast algorithm is known yet for any problem in this class, thus solutions to problems in $\mathcal{NP}$ are sometimes referred to be "infeasible to compute".
General applicable algorithms to solve $\mathcal{NP}$ are guessing and brute-force testing, which we use synonymous for an exhaustive search for solutions.
Of course the inclusion $\mathcal{P} \subseteq \mathcal{NP}$ holds, but the question if the statement $\mathcal{NP} \subseteq \mathcal{P}$ (and thus $\mathcal{P} = \mathcal{NP}$) is also true, is one of the millennium problems listed in 2000 by the Clay Mathematics Institute in Cambridge. A solution to this question is worth 1 000 000 US$, which is an additional motivation for mathematicians and computer scientists.
A class of particular interest is the class $\mathcal{NP}$-`complete` — problems that are "at least as hard" as problems in $\mathcal{NP}$. There are more than 3000 $\mathcal{NP}$ problems of practical interest known in computer science and not even one deterministic algorithm with polynomial

time complexity is known to the researchers so far. Instead, most of those problems have a time complexity that grows exponentially fast. Except for rather trivial lower bounds, hardly anything is known.
An interesting question is "Do faster algorithms exist for a specific $\mathcal{NP}$-complete problem?". This is, in fact, one big question in cryptography, because we do not know if a system based on such a problem is as secure as assumed. We refer to Section 2.3, where the existence of one-way functions is discussed. Next we focus on specific problems of interest in cryptography.

2.1.1 Suitable Problems and Algorithms

Now we want to present some sources of hard problems that are used for cryptographic purposes. Two well known systems are based on number theoretic problems. The factorization problem and the discrete logarithm problem. Although they are not proven to be $\mathcal{NP}$-complete, these two problems are of broad practical relevance to date and have been topic to numerous scientific discussion in the last decades.

Factorization Problem

The RSA public key cryptosystem — named after Rivest, Shamir and Adleman — is based on the integer factorization problem: Decompose an integer $N = pq$, which is the product of two big unknown prime numbers p, q.
We remark that although we present the version of the RSA cryptosystem with φ, the function $\varphi(N) := (p-1)(q-1)$ can be replaced by the least common multiple instead of the product $\lambda(N) := \text{lcm}(p-1, q-1) \leq \varphi(N)$ to save computational effort.
The setup for the public key requires a non-negative integer $e \geq 2$, $\gcd(e, \varphi(N)) = 1$ together with the product N. The private key is the factorization of N that is given by the prime numbers p and q and the multiplicative inverse of e, which is a positive integer d, such that $ed \equiv 1 \mod \varphi(N)$ holds. Using the factors p and q it is possible to calculate d with ease.
Given a plain text encoded as a number m with $2 \leq m \leq N-2$, the encrypted plain text is obtained by computing $c := m^e \mod N$.
Receiving $1 \leq c \leq N-1$, the owner of the private key can calculate $c^d \mod N$, which of course equals $m = m^{ed} = (m^e)^d \mod N$ the original message.
Although it is an interesting topic — studying the secure choices for p, q, e themselves or the algorithms, for computing the occurring modular powers efficiently, for instance — we refer to the huge amount of literature on this important branch of cryptography. In Chapter 4 however, we will see the weakness of RSA under the assumption of a powerful quantum computer and the speedup that can be achieved compared to the best classical algorithms.

Discrete Logarithm Problem

The ElGamal cryptosystem is based on the discrete logarithm problem (DLP) for a cyclic group G generated by an element g and can be stated as follows. Given an element $b \in G = \langle g \rangle$, find the unique integer r such that $g^r = b$ in G with $r < |\langle g \rangle|$.

The setup requires a positive integer h as private key. The information about the construction of the group $(G = \langle g \rangle, g)$ as well as the element $b := g^h$ form the public key.
To send a message, which we assume to be encoded as an element $m \in G$, the sender chooses a random positive integer k and sends the pair (g^k, mb^k).
The plain text can be recovered by the receiver with the private key h by computing

$$(mb^k) \cdot (g^k)^{-h} = mb^k g^{-hk} = mg^{hk} g^{-hk} = m.$$

The feasibility of the algorithms to solve the DLP depends heavily on the structure of the underlying group G. The simplest case for G is the residue class of integers with addition. If one changes G to be points on an elliptic curves, the problem gets harder. This introduced the use of elliptic curves in cryptography and the prefix "EC" became widespread and since then labels methods that have been adapted to perform calculations on elliptic curves.
Both problems, the factorization problem and the DLP can be seen as hidden subgroup problem for appropriate finite Abelian groups which will be of importance in Chapter 4 about quantum computing.
In 1978 Merkle and Hellman introduced a competitor to RSA, the Diffie-Hellman Knapsack problem. It was based on the number theoretic problem called SUBSET SUM. Although the general case of SUBSET SUM is $\mathcal{NP}$-complete, the transformed version used by Merkle and Hellman is computationally "easy". The system was broken in 1983, which showed that the transformation was not strong enough. Improved versions have also been broken by an algorithm with polynomial time complexity.
Interestingly cryptographic problems that are based on linear codes, which will be discussed in the next section of this thesis, are not even among the most common used cryptographic problems listed on Wikipedia [38] that meet hardness assumptions. This work's intend is to point out the advantages of linear codes in cryptography and to make such systems more widely known.

Coding Theoretic Methods in Cryptography

In 1978 Berlekamp, McEliece and van Tilborg [3] showed that Maximum Likelihood Decoding (as defined in Section 1.2) of a general linear code C is $\mathcal{NP}$-complete.
The code is given in terms of the parity check matrix H. Given an arbitrary binary vector y, the task is to find the error word e of minimum Hamming weight such that $He = s_y$, where $s_y := Hy$ is the syndrome of y.

Algorithm 2: Maximum Likelihood Decoding (MLD)

Input : H, y
Output: error word e of minimum Hamming weight such that $He = s_y, s_y := Hy$

To be more precise, the authors actually reformulated the problem stated as Algorithm 2 to an associated decision problem that gives either the answer "yes" or "no", providing information whether there exists a word e of weight $w(e) \leq t$ such that $He = Hy$. To get the desired word of minimal Hamming weight, the general approach is to start asking

(by running the algorithm that answers the decision problem), whether there is a word of weight $t := 1$ and increase t by 1 until the answer is "yes" for the first time.
The authors further proved a second problem to be $\mathcal{NP}$-complete problem, which can be stated as follows:
Given the number t of erroneous positions, find a codeword $c \in C$ of weight $w(c) = t$. In the literature about complexity theory, these results are summarized as

$$\text{COSET WEIGHTS, SUBSPACE WEIGHTS } \in \mathcal{NP}\text{-complete}.$$

There is no algorithm known that fulfills those two tasks in polynomial time depending on the input size (length of y respectively c). The existence of such an algorithm would provide a major breakthrough in complexity theory.
Since a polynomial time algorithm would solve one of the famous millennium problems listed by the Clay Mathematics Institute in Cambridge, the question $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$ would be answered "yes". Most experts nowadays are skeptical about a positive answer ever to be found to this problem, it seems more likely that $\mathcal{P} \subsetneq \mathcal{NP}$.
A critic might think the situation in Algorithm 2, where H is obtained long before y and thus may has been analyzed a lot, is a case of greater practical relevance.
However, Bruck and Naor [8] showed in 1990 that although H might be analyzed and preprocessed for as long as desired, the so called Maximum Likelihood Decoding with Preprocessing (MLDP) problem (see Algorithm 3 for the formulation) remains $\mathcal{NP}$-complete. In the proof they reduce MLDP to the simple max cut problem (SMC) that has already been proven to be $\mathcal{NP}$-complete.

Algorithm 3: Maximum Likelihood Decoding with Preprocessing (MLDP)

`// Preparation: Preprocessing of` H
Input : $s_y := Hy$
Output: error word x of minimum Hamming weight such that $Hx = s_y$

The fact that H is not part of the input allows "arbitrary preprocessing", which is not defined more closely by the authors and thus means any information that can be derived. In the same paper they also ask the interesting question "Does every linear code have an efficient decoder?" with relevance to coding theory. The fact that MLDP $\in \mathcal{NP}$-complete shows that a positive answer to this question is unlikely, since $\mathcal{P} \subsetneq \mathcal{NP}$ is conjectured.
Bruck and Naor conclude that: "knowledge of the code does not help in general in designing an efficient decoder simply because there exist codes that probably do not have an efficient decoder".
It is important to remark that in complexity theory the term "hard" refers to the worst-case, whereas in cryptography we need problems that are "hard" to solve for most instances, which means they need to be hard in the average-case.
Based on the fact that the general decoding problem for a random linear code is hard in the average-case — and it is likely to remain hard, unless $\mathcal{P}$=$\mathcal{NP}$ is proven one day — McEliece proposed a public key cryptosystem. We refer to Section 2.2.1 for more details on how a linear code with rich structure is disguised to appear as a random linear code.

Some Definitions

The terms one-way function, trapdoor function and hash function as well as the notion of provable security are briefly discussed in this section.

Definition 2.3. *A function $f : X \to Y$ is called one-way, if*

- *$y = f(x)$ is "easy" to compute that means there is an algorithm $A \in \mathcal{P}$ for this task.*
- *Given $y \in f(X)$ it is "hard" to compute an $x \in X : f(x) = y$. There is no (probabilistic) polynomial time algorithm for this task.*

It is interesting that there is an explicit function which has been demonstrated to be one-way if and only if one-way functions exist at all. But still the mere existence of one-way functions is not known.

Definition 2.4. *Let $f : X \to Y$ be a one-way function.*

- *f is called trapdoor function if f is usually "hard" to invert, but given additional, secret information (the trapdoor), there is a polynomial time algorithm to compute the inverse.*
- *f is called a collision-free hash function, if no polynomial time algorithm given the input x can find $y \neq x$ with colliding images $f(x) = f(y)$.*

  Sometimes the definition of a hash function is extended to randomized algorithms only finding collisions with sufficiently small probability.

The factorization problem, DLP or decoding a random linear code provide candidates for trapdoor functions in practice, although as we stressed before the existence is still unknown.
An instance of each of those problems is easy to solve, if some additional information is given whereas it is believed to be computational infeasible to solve a general instance.
Although we will not discuss further details here, interestingly the existence of one-way functions also implies the existence of the following secure cryptographic primitives [40]:

- Pseudorandom number generators, (Aim: Deterministically produce random number sequences that share many properties with truly random number sequences.)
- Bit commitment schemes, (Aim: A fixes a value and later reveals the commitment.)
- Private-key encryption schemes that are secure against adaptive chosen-cipher text attacks (see Section 2.2.1),
- Digital signature schemes that are secure against adaptive chosen-message attacks,
- Message authentication codes (see Section 2.3).

Provable Security

Cryptography wants to provide methods that are, in a mathematical sense, provable secure. The idea of underpinning the term provable security with methods from the field of computational complexity theory leads to different definitions as well as it leads to some confusion.
In this thesis however we speak of provable security, if — in order to break a system — the attacker has to solve the underlying problem intended by the designer. This means that we do not consider a specific implementation nor so-called side-channel attacks.In the cryptographic schemes we presented, which are based on the suitable problems listed above, an attacker is usually modeled as an adversary of the communication channel, where secrets are exchanged. This is as much information as the adversary gets.
Mathematical proofs now try to show that — under these assumptions — the only possible way to unveil the secret is to solve the hard problem.
As Koblitz and Menezes [19] remark that: "Throughout the history of public-key cryptography almost all of the effective attacks on the most popular systems have succeeded not by inverting the one-way function, but rather by finding a weakness in the protocol.".
With this in mind we move on and discuss important cryptographic schemes.

2.2 Public Key Cryptography

Public key cryptography is fundamental to modern communication and the methods we present in this section are actively involved in nowadays applications (see Section 2.6).
Since key distribution is the main problem in symmetric cryptosystems, asymmetric cryptosystems — also called public key cryptosystems (PKS) — were introduced in 1976 by Diffie and Hellman. PKS do not have the setup requirement that N communication partners need $\binom{N}{2}$ secret keys to be distributed over a secure communication channel. Instead each participant requires a pair of keys that is individually generated. The pair consists of a part that needs to stay private, the private key, and a public key that may be published openly, for example on the Internet. A digital signature scheme can directly be derived if the PKS has a certain property (see Section 2.3).

Key Agreement

Diffie and Hellman originally introduced their key agreement scheme based on the discrete logarithm problem. Assume Alice and Bob want to communicate. This is how they agree on a session key that can be used for establishing a secure channel using common public key encryption.
After fixing a finite commutative group G and an element g of large order in G, Alice chooses a random integer h, then she computes $g^h \in G$ and finally sends the result g^h to Bob. Bob on the other hand chooses a random integer k, then computes and sends $g^k \in G$ to Alice. Both are now enabled to compute their joint key g^{hk}. An adversary of their communication channel has to solve the DLP for the cyclic subgroup $\langle g \rangle \leq G$ in order to compute h or k out of g^h or g^k, respectively.

Public Key Infrastructures

Although public key infrastructures (PKI) exist, where companies ensure that a key belongs to a person or user by providing certificates, PKS can be used without such an infrastructure by manual distribution of the public key. Of course it is much more comfortable if the keys for communication are available on key servers on the Internet, but a user has to trust the company prior to using this PKI. Key servers are one example how the public key may be distributed to every communication partner.
On the other hand there is an alternative approach with advantages over both PKI and manual distribution, called "web of trust" (WOT). There users sign the keys of others that have identified themselves personally. Trust is then granted transitively through the web that results from many users participating. This means if Alice A fully trusts Bob B, who himself trusts C, it is suggested that A can trust C. We want to remark that there are certain disadvantages of WOT, but this topic will not be pursued in further detail. Instead, from now on we will always assume that keys are correctly distributed.
The first step to introduce coding theory to cryptography was done by McEliece more than 30 years ago. The McEliece cryptosystem proposed in 1978 [24] takes an easy instance of a problem that in general belongs to the class $\mathcal{NP}$-`complete` and disguises it as a random instance of the problem, which is hard to solve. This idea is similar to the Merkle-Hellman Knapsack cryptosystem (see Section 2.1.1), but McEliece uses a problem from coding theory instead of number theory — and whereas the first problem was broken long ago, McEliece's cryptosystem withstands cryptanalysis so far.
The problem of decoding a general linear code is a hard problem, where no fast algorithm exist so far. The additional information, how to transform the decoding of a general linear code to the problem of error correcting a binary linear Goppa codes, where fast algorithms are known, forms a good private secret suitable for a PKS.
Back then McEliece's PKS seemed impractical to use, because there were simpler schemes that were believed to be secure enough in the sense that no possible attack were known. Today theoretic attacks involving the quantum computer (see Section 4.3.4) could render widespread used public key cryptography more or less useless. This possible threat was the reason that the scientific community became even more interested in alternative asymmetric ciphers that can substitute the affected schemes.

2.2.1 The McEliece PKS

In 1987 McEliece [24] suggested the implementation of a PKS by randomly selecting the generator matrix of a binary $[1024, 524, 101]$ Goppa code C (there are many such codes) and disguising the code as a general linear code (there is a vast amount of such codes). Two matrices were introduced that disguise the original generator matrix. Assume Alice wants to set up the McEliece PKS:
Key generation. She chooses the code C such that the parameters fit with her desired security level (see Section 2.5), and that she has an efficient decoding algorithm for C at hand. Assume the code C is able to correct up to t errors and the generator matrix of the code is G, then Algorithm 4 provides a private key to be stored and kept secret and the suitable public key to distribute.

Algorithm 4: McEliece key generation

Input : $(k \times n)$ generator matrix G, error correcting capability t
Output: public key (G', t), private key (S, G, P)

Choose a $(n \times n)$ permutation matrix P
Choose a regular binary $(k \times k)-$matrix S
Compute $(k \times n)$ matrix $G' = SGP$

Encryption. Bob who wants to send his message M to Alice, retrieves the public key (G', t) and therefore implicitly knows n, k. If the message is too long, the encryption Algorithm 5 splits the message M in blocks m of suitable length $|m| = k$.

Algorithm 5: McEliece encryption

Input : message block m, public key (G', t) and thus implicitly n, k
Output: encrypted message block c

foreach block m **do**
  Compute $c' = mG'$
  Randomly generate a vector $z \in \mathbb{F}_q^n$ with non-zero entries at $\leq t$ positions
  Compute $c = c' + z$, the cipher text block
end

Decryption. Assume Alice gets the McEliece encrypted cipher text blocks $c_1, c_2, \ldots$ and wants to read the message $M = m_1 m_2 \ldots$, then Algorithm 6 describes the decryption process.

Algorithm 6: McEliece decryption

Input : encrypted message block c, private key (S, G, P)
Output: message $M = m_1 m_2 \ldots$

foreach block c **do**
  Compute $\bar{c} = cP^{-1}$
  The fast decoding algorithm of the code C corrects t errors. $\bar{c} \to \bar{m}$.
  Compute $m = \bar{m}S^{-1}$, the clear text message block.
end
// The inverted permutation matrix P^{-1} and the inverted matrix S^{-1} can be precomputed once and for all.

Next we give the proof that decrypting an encrypted message yields the original message, as desired.

Proof. The receiver of the encrypted block c has the private key (S, G, P) and can easily compute P^{-1} and S^{-1}. As well as z, the permuted version zP^{-1} has weight t. Since

$$\bar{c} = cP^{-1} = mG'P^{-1} + zP^{-1} = (mS)G + zP^{-1}$$

holds, the desired plain text codeword mSG has at most distance t from cP^{-1}. The efficient decoding algorithm corrects the t errors. The result is the codeword $(mS)G \in C$ from which $\bar{m} = mS$ can be computed. Finally, we obtain the original message block $m = \bar{m}S^{-1}$, by multiplication with S^{-1}. □

If we want to use the word trapdoor function (see Definition 2.4), the easy direction in this case is the matrix multiplication $G' \leftarrow SGP$ during McEliece encryption. The hard way is the matrix decomposition of $G' \rightarrow SGP$. The two matrices S, P actually disguise the Goppa code as a general linear code.
Obviously, since $G' = SGP$ (the decomposition, of course, is unknown to the attacker) might be well analyzed by an attacker before a cipher text is actually intercepted, the many possibilities of the permutation P and the "scrambler" S are enough to hide the actual code C and thus its effective decoding algorithm.

Reducing Public Key-Length

Assume G' is the public generator matrix of a code C used for cryptographic purposes after key generation. Using Gaussian elimination one can achieve a corresponding systematic form $\bar{G} = (I_k|M)$, where I_k is the $(k \times k)$ identity matrix and M is a $(k \times (n-k))$ matrix. Since each step of the transformation of G' to $\bar{G}$ is invertible, both essentially generate the same linear code C. We mention that the public key owner might need to permute the columns to get a systematic generator matrix. The inverse of these transformations then of course have to be kept for decoding purposes. Although the decoding gets more complicated for the owner of the key, the public key size can be considerably reduced as follows.
The argumentation above shows that a systematic generator matrix $\bar{G}$ can be used instead of the matrix G' as the public key. This reduces the public key size requirements from $k \cdot n$ to $k \cdot (n-k)$ field elements. It is sufficient to store the non-trivial $(k \times (n-k))$ part. For binary codes, one can directly see how many bits can be saved using this additional method, for example.
But, the main disadvantage of reducing key length like this is that it completely disables the encryption, since the message will just be sent in plain text and certain bits will be appended. Warning, using the McEliece cryptosystem like this makes no sense; it resembles conventional channel coding and does not encrypt the data at all! The reason why this approach is not discarded right away is because of a necessity pointed out in Section 2.2.1. This approach, along with being aware of the immediate security flaw and methods to deal with it, is sometimes referred to as "modern McEliece".

Drawbacks and Attacks

More than 30 years ago the original parameters $n = 1024, k = 524, t = 50$ were proposed to achieve a certain security level at that point of time. They have been adjusted to $n = 1632, k = 1269, t = 34$ because of attacks that were explored since then. See Section 2.5 for today's recommendations. Due to the chosen-cipher text attack described below, modifications needed to be made to McEliece's original system.

Using the updated and to date valid recommendation for the parameters, the modification of using a systematic form (as in Section 2.2.1) leads to smaller public keys with $k \cdot (n-k) = 460647$ bits. The key size is, compared to other PKS, rather large. This is the most criticized fact about the McEliece PKS.
Berson [6] showed the following weakness that required the parameters to be adjusted:

Theorem 2.5. *The message-resend attack applies to the McEliece cryptosystem and is thus a weakness.*

Proof. Suppose the same plain text x is encrypted twice and since the encryption is probabilistic this, most likely, yields two different cipher texts $y_1 = xG' + z_1, y_2 = xG' + z_2$ with $w(z_1), w(z_2) \leq t$. We emphasize that z_1, z_2 are two random vectors that are nonzero only at t positions at most.
Computing $b := y_1 - y_2 = z_1 - z_2$ yields a vector that has non-zero entries where error positions of both cipher texts y_1, y_2 are. Positions where b is zero were most likely error free, because it is unlikely that both z_1, z_2 have the same value at the same position, if chosen randomly.
This observation helps retrieving the sent message x without knowing H, M or P. □

The message-resend attack presented does not depend on the code and can be generalized to a so called related-message attack. This attack tries to exploit a linear relation between messages that were encrypted to recover them.

CCA2-security

Assume an attacker of a cryptosystem is faced with the task given a cipher text and two different plain texts to determine which one of them was encrypted to yield the cipher text at hand.

Definition 2.6. *A cryptosystem is called "adaptive chosen cipher text attack"-secure (or CCA2-secure) if such an attacker — with unlimited computational power and with access to a decryption oracle — can do no better than guessing.*

The two plain texts are "indistinguishable" to the attacker in this sense, which is sometimes denoted as IND-CCA2-secure. The idea of this notion of security is that apart from obvious and trivial observations, no additional information is leaked by a cipher text that has been encrypted using a CCA2-secure PKS.
Although this is a topic for an actual implementation of the McEliece PKS that is naturally concerned with efficiency, shorter keys obtained by choosing a systematic generator matrix (as discussed in Section 2.2.1) is justifiable, because Overbeck [28] argues that these "modifications do not alter the security of the system as long as a semantically secure conversion is used" and "such a conversion is needed anyway". So McEliece's cryptosystem should only be used in the CCA2-secure variant.
A detailed survey of possible conversion to make the McEliece cryptosystem CCA2-secure has been carried out by Kobara and Imai [18, Ch. 4].
Although the public keys are still large with respect to other PKS, the mentioned weakness to the message-resend attack does not occur in the similar Niederreiter cryptosystem discussed in Section 2.2.2, although it is not CCA2-secure.

We come back to mention more drawbacks, like the low information rate that was criticized before, which is $R = \frac{524}{1024}$ for the original suggested values of the underlying $[1024, 524, 101]$ Goppa code. Also here the Niederreiter cryptosystem has some advantages and a theorem about the rather good information rate will be presented in the next section. The large public keys of the McEliece cryptosystem and code-based cryptosystems in general is frequently named as reason that it never gained much practical relevance.
Possible attacks on the McEliece cryptosystem with the originally suggested parameters are discussed now. Assume an attacker, like every communication partner, has the public key $(G' = SGP, t)$ and further has intercepted a codeword $y = G'x$ and naturally wants to read the original message x. There are four possible threats:

1. Guess S and P. S is a regular (524×524) matrix and there are 1024! permutations.
2. MLD to find closest codeword. Compare y with the 2^{524} possible words.
3. Syndrome decoding. This can be done with $2^{1024-524} = 2^{500}$ comparisons.
4. Select 524 random positions and hope they are not erroneous. The probability of success that means no error in this set of 524 elements is very small given these parameters
$$\frac{\binom{1024-50}{524}}{\binom{1024}{524}} \approx 7,2 \cdot 10^{-17}.$$

Obviously these four attacks are infeasible for the original parameters and remain safe for a careful choice of new parameters for the code length and the dimension.

Exchange of the Code

Since the McEliece PKS is not bound to the underlying code, the idea of substituting the proposed binary Goppa codes by other Goppa codes, even other classes of codes arose. We already discussed some possible parameter adaptations and their limitations with respect to security flaws.
Moreno and Janwa [17] start with the fact that there is not merely one $[1024, 524, 101]$ Goppa code but there are many of inequivalent Goppa codes with these parameters. During the key generation, a user of the McEliece scheme has the freedom of choosing one suitable Goppa polynomial of degree 50, for example.
The security relies on the fact that a high work factor is required to find the right code. Since the work factor of the McEliece PKS with proper parameters is regarded to be sufficient, the idea of reducing the code length, while keeping a high work factor led Janwa and Moreno [17] to focus on the big class of algebraic-geometric codes. On the one hand AG codes with shorter codeword length can be used to reduce the public key size, but unfortunately the decoding algorithms for this general class are not efficient enough so far. It is still a research topic to extend good decoding algorithms to bigger classes of codes.
The geometric view on codes also introduces many new possibilities in varying the parameters and thus enhancing the work factor for cryptographic purposes and quite a few attempts were made to establish encryption systems upon them.

For the particular case of substituting Goppa codes with generalized Reed-Solomon codes Sidelnikov and Shestakov [36] have shown that there is an attack to the McEliece PKS. This rules out a big class of codes for and moreover implies an insecurity in Niederreiter's PKS, since it is equivalent to the McEliece cryptosystem as proven in Section 2.2.3.

2.2.2 The Niederreiter PKS

To define the Niederreiter cryptosystem $\mathcal{C}$ (following [25, section 6.4]) one needs any linear $[n,k,d]-$code C over $\mathbb{F}_q$ with an efficient decoding algorithm. This code, given implicitly by the $((n-k)\times n)$ parity check matrix H, needs to be secret. Additionally one generates a $(n\times n)$ permutation matrix P and an arbitrary, regular $((n-k)\times(n-k))$ matrix M over $\mathbb{F}_q$. These three matrices form the private key. Define $t := \lfloor \frac{d-1}{2} \rfloor$ the maximum error correcting capability of the code C.
The product $H' := MHP$ of dimension $((n-k)\times n)$ is the public key. H' can be seen as the disguised version of H that ideally cannot be reconstructed from H' unless one gains knowledge of M and P.
The admissible plain texts in this cryptosystem are words $x \in \mathbb{F}_q^n$ with weight $w(x) \leq t$.
The encryption process for a given x, is the computation of $y := H'x$. This is deterministic, contrary to the McEliece cryptosystem, where a random vector $z \in \mathbb{F}_q^n$ is involved.
Being the owner of the private key M, P, H and receiving a cipher y we proof the correctness of the decryption process.

Proof. Let $y = H'x = MHPx \in \mathbb{F}_q^{n-k}$ be an encrypted word. The first step in the decryption process is the computation of $y' = M^{-1}y = HPx = Hx'$ if one sets $x' = Px$. Since $w(x') \leq t, x'$ can be interpreted as an correctable error vector. The decoding algorithm of C is capable of correcting the t errors in an efficient way, thus retrieving x' from $y' = Hx'$. Finally computing $x = P^{-1}x'$ yields the originally sent message. □

Definition 2.7. *The q-ary entropy function is defined by*

$$H_q(x) = x\log_q(q-1) - x\log_q x - (1-x)\log_q(1-x).$$

Definition 2.8. *Let $S(\mathcal{C})$ be the number of possible plain texts and $T(\mathcal{C})$ the number of possible cipher texts of a cryptosystem $\mathcal{C}$ over the alphabet $\mathbb{F}_q$.*
Then the information rate of the cryptosystem

$$\mathcal{R} = \mathcal{R}(\mathcal{C}) := \frac{\log_q S(\mathcal{C})}{\log_q T(\mathcal{C})}$$

may be viewed as the amount of information contained per bit in cipher text.

Example 2.9. *Let $\mathcal{C}$ be a Niederreiter cryptosystem deployed over an $[n,k,d]-$code C, the number of possible plain texts are all length n words with weight $\leq t := \lfloor (d-1)/2 \rfloor$. Therefore $S(\mathcal{C}) = \sum_{j=0}^{t} \binom{n}{j}(q-1)^j$, because for each $j \leq n$ we can choose j positions to be some value $\neq 0$ and there are $(q-1)$ such values. The number of possible cipher texts is $T(\mathcal{C}) = q^{n-k}$, since the public key maps length n input words to length $(n-k)$ cipher texts. We have*

$$\mathcal{R}(\mathcal{C}) := \frac{\log_q(\sum_{j=0}^{t}\binom{n}{j}(q-1)^j)}{n-k}.$$

Next we present a theorem about the information rate of some Niederreiter cryptosystems [25, Theorem 6.4.2]:

Theorem 2.10. *Let q be a prime power and let $0 < \theta < \frac{q-1}{2q}$. Then there exists a sequence $(\mathcal{C}_i)_{i\in\mathbb{N}}$ of Niederreiter cryptosystems based on linear $[n_i, k_i, d_i]-$codes C_i over $\mathbb{F}_q$ with lengths $n_i \to \infty$ for $i \to \infty$ and weight conditions $w(x) \leq t_i := \lfloor \frac{d_i-1}{2} \rfloor$ for the admissible plain texts $x \in \mathbb{F}_q^{n_i}$, such that*

$$\lim_{i\to\infty} \frac{t_i}{n_i} = \theta, \quad \lim_{i\to\infty} \mathcal{R}_i \geq \frac{H_q(\theta)}{H_q(2\theta)}.$$

The theorem can be interpreted as follows. The information rate of the cryptosystems may stay reasonable good for long codes, when at the same time the ratio of error correcting capability over code length remains constant.
The proof of the theorem as well as the following example are given in [25, section 6.4].

Example 2.11. *Consider Niederreiter cryptosystems $(\mathcal{C}_i)_{i\in\mathbb{N}}$ over binary linear codes. Since $H_2(\frac{1}{4}) = -\frac{1}{4}\log_2(\frac{1}{4}) - \frac{3}{4}\log_2(\frac{3}{4}) = 0,5+0,311\cdots > 0,81$ and $H_2(\frac{1}{2}) = 1$, there exists a choice of θ sufficiently close to $\frac{1}{4} = \frac{q-1}{2q}$ such that Theorem 2.10 yields*

$$\lim_{i\to\infty} \mathcal{R}_i \geq \frac{H_2(\theta)}{H_2(2\theta)} \geq \frac{H_2(\frac{1}{4})}{H_2(\frac{1}{2})} > 0,81.$$

This shows that asymptotically 81 out of 100 bits cipher text carry information.

Unlike the RSA cryptosystem, the McEliece cryptosystem and the Niederreiter cryptosystem cannot be directly converted to digital signature schemes. In Section 2.3 we will have a closer look at signatures.
We remark that for the Niederreiter PKS it is sufficient to store the non-trivial $((n-k)\times k)$ part of the parity check matrix instead of the full $((n-k)\times n)$. This reduces the public key as discussed in Section 2.2.1. Although it is not that obvious, the CCA2-conversion (see Section 2.2.1) still has to be done in order to achieve this notion of security. As Overbeck pointed out in [28], a weaker form of an adaptively chosen cipher text attack, namely the reaction attack, can still derive some information from a given cipher text — a fact not allowed in the CCA2 model. Hence the "modern" version with the systematic parity check matrix might as well be used since a CCA2-conversion is necessary anyway.

Drawbacks and Attacks

In the paper from the year 2006 [9] Engelbert, Overbeck and Schmidt discuss the security of McEliece-type PKS. In particular, the authors describe the attack by Shestakov and Sidelnikov on the original Niederreiter PKS that determines secret parameters in polynomial time. In [9, Algorithm 3.2.1] they present an algorithm how to recover the private key of a Niederreiter PKS based on GRS codes, where the public key $H' \in \mathbb{F}_q^{n\times(s+1)}$ and the maximum number of errors t are given that thus breaks the cryptosystem.
We stay conformal with their notation and remark that we have to set $s := n - k - 1$; compare with (1.4). The attack takes advantage of the highly structured GRS code's parity check matrix over $\mathbb{F}_q$. We note that GRS codes are MDS.

This attack renders a lot of possibilities to choose the underlying code insecure, including the original Niederreiter PKS, which is based on GRS codes. Still — if deployed over irreducible binary Goppa codes — the algorithm is not applicable and thus the Niederreiter PKS based on this class remains secure. The fact that Goppa codes are not GRS codes, but subfield subcodes of GRS codes, is enough since this leads to different matrices M in the definition $H' = MHP$ (see Section 2.2.2) used for constructing the public key.
Moreover the authors present the interpretation of H' as evaluated polynomials over $\mathbb{F}_q$ by Shestakov and Sidelnikov. This viewpoint leads to a structural attack that is not applicable if the matrix H' was generated using a parity check matrix H of a Goppa code over $\mathbb{F}_2$. They argue that if the field has $q = 2^m$ elements for some m the construction of the public key will yield different invertible matrices $M \in \mathbb{F}_{2^m}^{(s+1)\times(s+1)}$ for GRS and $M \in \mathbb{F}_2^{m(s+1)\times m(s+1)}$ for Goppa code to hide the secret code structure. H' is said to have "no obvious structure, as long as M is unknown".
There have been efforts to generalize this and similar structural attacks to subcodes of GRS codes.
In their 2010 work [37] "Cryptanalysis of the Niederreiter Public Key Scheme Based on GRS Subcodes" the Wieschebrink did further cryptanalysis and concludes that "there seems to be no straightforward way to generalize the attack" to the case of irreducible binary Goppa codes and "a more detailed analysis of the attack in this respect remains part of future work".
Next we will see how closely related the McEliece and Niederreiter cryptosystems are. This result shows that there is no general structural attack for McEliece-type PKS using Goppa codes either so far.

2.2.3 Equivalency of McEliece and Niederreiter Cryptosystem

In [25, Theorem 6.4.1] the authors show the following:

Theorem 2.12 (equivalent security)**.** *With corresponding choices of code parameters, the McEliece cryptosystem and the Niederreiter cryptosystem achieve an equivalent level of security.*

Proof. Recall the definitions of both the McEliece and the Niederreiter cryptosystem.
Suppose first that we have an algorithm A to retrieve a vector x in polynomial time that has been encrypted to y using the McEliece cryptosystem.
The attacker has the cipher vector $y = xG + z, w(z) \leq t$, and the publicly known matrix G. The algorithm A is able to efficiently determine the vector x.
Now we consider the setting of the Niederreiter PKS. Assume we have a parity check matrix H of a code C and a cipher vector $y' = Hx', w(x') \leq t$. Using the Gaussian elimination algorithm from linear algebra, we can compute basis vectors v of the system of linear equations $Hv = 0$ and therefore a generator matrix G of C. This can be done in polynomial time. Next we compute a column vector u that solves $Hu = y'$. Now

$$H(u - x') = 0 \Leftrightarrow u - x' \in C \Leftrightarrow \exists a \in \mathbb{F}_q^k : \ (u - x')^T = aG.$$

This is equivalent to $u^T = aG + x'^T, w(x') \leq t$. Algorithm A retrieves a in polynomial time, since the error x' has weight at most t. Finally x' can easily be computed. Hence

the Niederreiter cryptosystem is broken in polynomial time by the transformations given above and algorithm A.
Conversely, suppose we know an algorithm B to break the Niederreiter cryptosystem. This means that given a parity check matrix H of C and a vector $y' = Hx'$ with $w(x') \leq t$, the algorithm B determines the vector x' in polynomial time.
Assume we have the generator matrix G of C and an vector y encrypted by the McEliece PKS, so $y = xG + z$,$w(z) \leq t$ holds. Again by Gaussian elimination, we can compute a parity check matrix H of C efficiently in polynomial time. We have

$$yH^T = xGH^T + zH^T = zH^T.$$

Since $Hz^T = Hy^T$,$w(z^T) \leq t$, the algorithm B can be applied and retrieves the vector z^T. Finally, $y - z = xG$ yields the plain text vector x in polynomial time; hence the McEliece cryptosystem has been broken by B and the transformations specified above. □

This proof especially shows that every attack on the McEliece PKS might be directly applied on the Niederreiter PKS and vice versa.

2.3 Signatures

Another important topic is message authentication. After Alice and Bob have established an encrypted communication channel, there is always the problem that there might be an imposter that plays Alice's role when communicating with Bob and the other way round. Bob and Alice might never suspect anything, because everything looks normal.Such an attacker is called man-in-the-middle.
The goal of data authentication is to enable the communication partners to check whether the received message is from the claimed sender and furthermore detect if it was altered during transmission. These aims are referred to as authenticity respectively integrity.

Message Authentication Code

A message authentication code (MAC) is a suitable one-way function that depends on a key and adds message dependent information to a message thus providing authenticity and integrity. Since MACs are symmetric cryptographic primitives, the same key used for adding information to a message is needed by the recipient to verify the message. Often hash functions (one-way function candidates that always output a word of fixed length, given an arbitrary input) are used for this purpose.
A digital signature on the other hand is the asymmetric version of message authentication. As long as a key pair can be identified with a unique person signatures can provide authenticity and integrity in a comfortable and safe way.
Like a real signature on a piece of paper, a digital signature can be appended to digital documents. A private key, known to belong to a certain user, can ensure the integrity of the digital signature. Unlike most signatures, digital signatures are not human readable; they are strings of letters accompanying the document. We refer to Section 2.6 for a practical example.

An advantage over classical, written signatures is that the document's origin can be verified by everyone with access to the public key and the content of a signed document can not be altered without making the signature invalid.
To depict a real world analogy, digitally signing a document is like sealing the envelope holding the document with a wax seal uniquely corresponding to the sender.
We remark the property that enables arbitrary PKS to work as a signature scheme is that the decryption algorithm needs to be able to be applicable to every plain text — without restrictions.

A Signature Scheme using Hash Functions

We now discuss a generic method of setting up a signature scheme using hash functions. Assume Alice has a key pair suitable of performing public key encryption.
General signature generation: Algorithm 7 shows how Alice can prove her message's origin. Alice and Bob agree on a common, secure hash function h and a PKS. Let f_e denote the encryption function of the PKS (depending on Alice's public key) respectively decryption function f_d (depending on Alice's private key).
A hash function h, as introduced in Definition 2.4, is a cryptographic primitive which is used in a way that it maps the input of arbitrary length to a string of fixed length.
Alice wants to prove the authenticity and the integrity of message M she sends to Bob.

Algorithm 7: Signature generation

Input : Alice's private key, message M
Output: signed message S

Compute hash of message $s' \leftarrow h(M)$
Scramble (=decrypt) the hash value using the private key $s \leftarrow f_d(s')$
Append signature to the message $S = [M, s]$

General signature verification: Upon receiving a message S and retrieving Alice's public key, Bob tries to verify, whether S originates from Alice. Algorithm 8 summarizes the steps to perform this task.

2.4 Authentication

Authentication — not to be confused with message authentication — is the process of getting access to a server, for example.
Authenticating ones identity to a server is usually done via a secret password. A copy of the password itself or of its hash value needs to be stored on the server to compare it when the user attempts to log in. This may be insecure and many techniques are known to crack such password files or hash files stored on the server if they are accessible for an attacker.
Assuming a working public key infrastructure, there exists a better method than storing plain text or the hash value.

Algorithm 8: Signature verification

Input : signed message $S = [M, s]$, Alice's public key
Output: logical value, deciding whether M is from Alice and M is untampered

Split signature S in it's two components s and M
Restore (=encrypt) the hash value using the public key $\bar{s} \leftarrow f_e(s)$
Compute $s' \leftarrow h(M)$

```
// Assuming Alice is the only person with access to the private key:
```
if $s' = \bar{s}$ **then**
    return true; `// The message originates from Alice.`
else
    return false; `// Either the message does not originate from Alice or the content of the message has been changed.`
end

Public key authentication of a user to a server works by sending the signature of the user's password that has been created with the user's private key.
Ylonen summarizes the authentication process in [43, Ch. 7]: The server

- MUST check that the key is a valid authenticator for the user and
- MUST check that the signature is valid.

If both conditions hold "the authentication request MUST be accepted; otherwise, it MUST be rejected". After accepting the public key authentication the server grants access or may require additional authentication steps.

2.5 Security Considerations

Cryptographic functions, used for computer security, often have the property that they protect certain data against attackers by using a secret of length usually much less than the message itself — the private key.
Thus the length of the secret key is an important security parameter. Following the axiom known as Kerckhoffs' principle, the security of a cryptographic system depends on the secrecy of the key alone. The rest — like details of implementation, methods or algorithms — need not to be kept a secret, in fact the details can be public knowledge. Some argue the routines even should be well known and therefore are likely better analyzed and thus more suitable for cryptographic systems.

2.5.1 Symmetric Schemes

For the well studied Advanced Encryption Standard (AES), the key size required to provide a certain level of security is considered to be the same as for symmetric cryptographic

schemes in general. To apply general considerations about key lengths to a specific cryptosystem, of course, relies on the assumption that the symmetric scheme itself is secure and cannot be circumvented.
Since AES is believed to be secure from a cryptoanalytic viewpoint — which is underpinned by a lot of research — a brute force attack is essentially the only threat. The feasibility of a systematic key search to decrypt encrypted data naturally depends on the number of possibilities there are. Hence if the number of key bits is large enough this attack is not feasible and the symmetric cryptosystem secure.
To determine the actual number of bits suitable for a specific purpose, one has to take into consideration:

- the lifetime of data,
- possible attackers,
- and their computing power P.

A predicted short lifetime of data reduces the need for a long key. Moore's law of exponential growth of computing power — it doubles roughly every 18 months — on the other hand may require a longer key, if the lifetime of data is more than, say 2 years.
To estimate the computing power P of an attacker mentioned above, one needs to simulate an attacker and take the resources at hand into account. The formula of describing P might include the number of processors in common computers (CPU) or graphic cards (GPU) that might be available for the attacker, for example.
This leads to an n-bit key where the resources of breaking the system cost too much time or money compared to the worth of the data.
The European Network of Excellence in Cryptology II (ECRYPT2) recommends the minimum key sizes to protect data for a few months against different attackers with varying resources.
There they choose the number n of key bits, in a the way that $2^n/P$ is "somewhat larger than the lifetime of the protected data". The recommendations are provided in Table 2.1.
The key bit recommendations taken from the ECRYPT2 paper are not adapted to a specific symmetric cryptosystem but consider a brute force key search as the best attack, which is reasonable if "the cryptosystem as such can be assumed to be secure for the lifetime of protected data".
Before taking a look at the table, we have to introduce the occurring abbreviations. The abbreviation PC stands for the common personal computer that may be connected in huge numbers via the Internet, for example. The field-programmable gate array (FPGA) is a more specialized device than the all purpose PC. An FPGA is an integrated circuit that needs to be configured after manufacturing and can thus be adapted to cryptographic purposes that therefore carry out those tasks more efficiently. Finally an application-specific integrated circuit (ASIC) is an integrated circuit that is designed for a particular use and hence can be even more customized for the execution of cryptographic tasks.
An interesting estimate, also mentioned in the paper [27], is the total computational power of the Internet, which (in the year 2006) was about 2^{85} operations per year. A certain fraction of this is controlled by hackers and malicious software. Hence the minimal requirement for some months of protection is set to 58 bits of symmetric key size.

Attacker	Budget USD	Hardware	Minimum bits security
Hacker, Malware	0 to 400	PC, FPGA	58-77
Small organization	10 000	PC, FPGA	69
Medium organization	300 000	FPGA, ASIC	69
Large organization	10 000 000	FPGA, ASIC	78
Intelligence agency	300 000 000	ASIC	84

Table 2.1: Minimum symmetric key size in bits for various attackers

A high security level may not be necessary and sometimes even not be reasonable for all types of applications. Depending on the application this might lead to lower security requirements for a certain task. Environments with limited hardware and computational power are an example where such considerations are made.
Thus ECRYPT2 meant to "define some security levels and quantify what security they reach and in which cases they might be acceptable" and they provided the Table 2.2 accordingly.

Bits	Protection	Comment on usage
32	Attacks in real-time	authentication tags
64	Very short-term	not for confidentiality
72	Short-term	against medium organizations
80	Very short-term against agencies	smallest general-purpose protection
96	Legacy standard level	about 10 years protection
112	Medium-term protection	about 20 years protection
128	Long-term protection	about 30 years protection
256	Foreseeable future	good protection against quantum computers

Table 2.2: Security levels and symmetric key size equivalent

Although a certain security margin has been added to the numbers in the table, they remain estimates. Development in the area of cryptanalysis especially discovering new attacks or improvements in the field of quantum algorithms (see Chapter 4) could change these tables.
We could go on listing interesting tables about security considerations for specific systems or against certain well-defined attackers. Instead we recommend the "Cryptographic Key Length Recommendation" website by Giry [15].
There the description points out that although publications and recommendations are widely available "choosing an appropriate key size to protect one's system from attacks remains a headache as you need to read and understand all these papers". Like the tables from above, both academic and private organizations assume a certain setting and provide estimated minimum key sizes for different levels of security.
The purpose of this website is to "summarize reports from well-known organizations allowing you to quickly evaluate the minimum security requirements for one's system. You can also easily compare all these techniques and find the appropriate key length for

one's desired level of protection". The authors of this website have taken various papers into account that give advice how to resist structural, mathematical attacks. Specific algorithmic attacks or hardware flaws, for example, were not considered by them and recommendations are thus to be seen in this light.

2.5.2 Asymmetric Schemes

The analysis of the security of asymmetric cryptography schemes is more difficult than the analysis of symmetric schemes considered in the last section. Although computationally hard problems are used as a source for asymmetric systems, they could not be proven to be secure by mathematical means so far. The security is mostly based on assumptions, even if there are strong indicators for it.
The keys in an asymmetric cryptography setting usually have more bits than keys of a symmetric scheme of comparable security level. Too large keys are negative for the performance of the asymmetric scheme. It is always a trade-off and since attacks exist that are more effective than brute force guessing, the secret keys may not be too short either.
According to the detailed discussion in chapter 6 of the ECRYPT2 report, Table 2.3 shows the minimum key lengths for asymmetric security accumulated (see [27, Table 7.2]).

Security (bits)	RSA	DLP field size	DLP subfield	EC
80	1248	1248	160	160
128	3248	3248	256	256
256	15424	15424	512	512

Table 2.3: Security levels and key size equivalent of common PKS

The figures and their validity must be seen in the light of best algorithms known to the scientific community. The conversion between symmetric key size recommendations and appropriate asymmetric key sizes has been discussed in the report [27, Chapters 5,6,7] in detail.
We remark that hybrid schemes exist where the good properties from both symmetric and asymmetric cryptosystems are combined. Asymmetric schemes are merely used to protect a symmetric key that secures the communication itself. Nevertheless, security considerations have to take every compound into account, because a "weak link in the chain" is undesirable.

2.5.3 McEliece and Niederreiter PKS

In 2008 Bernstein, Lange and Peters [5] showed how to attack the McEliece cryptosystem more effectively and thus reduced the security. They also gave a guideline to defend the system, by adjusting the parameters so that it is not vulnerable to their attack, which is up to date the best attack known. We remark that the same parameters apply to the Niederreiter PKS, since in Section 2.2.3 it is shown that the two cryptosystems are equivalent from a cryptoanalytic point of view.

bits	n	k	d	$\deg G(z)$	errors	size of key
80	1632	1269	≥ 69	33	34	460 647
128	2960	2288	≥ 115	56	57	1 537 536
256	6624	5129	≥ 235	115	117	7 667 855

Table 2.4: Security levels and according parameters of the McEliece PKS

In Table 2.4 the parameters suitable to achieve a certain level of security are given according to the paper by Bernstein, Lange and Peters, where $G(z)$ is the Goppa polynomial used to construct a binary $[n, k, d]-$Code. One column shows the recommended errors to be added and another one the size of the public key in bits calculated with the formula $k \cdot (n - k)$ that is valid if only the non-trivial part of the generator matrix needs to be stored.

key size (bytes)	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
n	1744	2480	3408	4624	6960
$\deg G(z)$	35	45	67	95	119
errors added	36	46	68	97	121
security level (bits)	84.88	107.41	147.94	191.18	266.94

Table 2.5: Parameters of the McEliece PKS with limited key size

The key length is always mentioned as a major disadvantage of the McEliece cryptosystem. There may be constrained environments for the deployment. Table 2.5 gives an overview of suitable parameters if the key size is limited due to storage constraints.

2.6 Applications

Apart from the rather general cryptographic applications like PKS, authentication and methods providing signatures and authenticity we dealt with in previous sections, we now want to briefly point out some more applications where cryptographic methods appear in everyday life. Cryptography that works in the background and users do not even need to be aware of its presence is desirable. For then it obviously is an easy to use technology that is cheap — or even free. The idiom at the start of Chapter 2 reminds us that although it might be fairly good it is probably not the most secure method available.

Connecting to Wireless Networks

The first step of users coming in touch with strong cryptography is often the connection to a wireless network that is protected via “Wi-Fi Protected Access II” (WPA2). Most common is the authentication by using a pre-shared symmetric key but more sophisticated authentication methods based on the AES are also available. The actual communication between the user and the access point is then encrypted via the symmetric cipher AES.

The keys for the algorithm are repeatedly generated, because they expire after a preset time for security reasons.

E-Mail

In the classical case of sending signed messages via e-mail — the domain of public key cryptosystems, there is usually a block of characters marked as signature appended to the text and for each attachment a separate file with a `.sig` ending is created.
For examples users of GnuPG (`http://www.gnupg.org/`) signing an e-mail produce a text block that looks like this:
`-----BEGIN PGP SIGNATURE-----`
...
`-----END PGP SIGNATURE-----`
If the e-mail is encrypted the original text is replaced by some cipher text enclosed in the
`-----BEGIN PGP MESSAGE-----`
...
`-----END PGP MESSAGE-----`
delimiters. Each attachment is replaced by a file with a `.pgp` ending and additionally the message can then be signed by appending a block like before.

Surfing on the Internet

Another prominent example of cryptography in everyday life is the Transport Layer Security (TLS) protocol, a hybrid protocol combining both symmetric and asymmetric cryptographic primitives. Most Internet users access the world wide web via their favorite browser that establishes connections to servers based on protocols, for instance TLS. In the browser's address bar the prefix `https` (instead of `http`) indicates the use of a secure channel. This is often accompanied by a lock, to symbolize the encrypted communicating between the user and the server which is verified by some certificate authority.
A secure channel to do online financial transactions, for example, is desirable. Apart from that, the use of a secured channel in most online activities has no drawbacks for the user. Therefore it is preferable to use encrypted communication to unsecured connections for surfing on the Internet.

Online Transactions Using an Alternative Currency

Another interesting application of cryptography not directly connected with exchanging messages is Bitcoin. Cryptographic primitives form the base for the "most widely used alternative currency" [33] called Bitcoin. Bitcoins (`http://bitcoin.org/`) are a global mean of payment and have no central bank as support. These digital coins are used in a peer-to-peer network and only rely on cryptographic protocols. Each transaction between the anonymous users is stored in a database and spent money is marked with digital signatures.
Bitcoin is an alternative or at least a new approach to exchanging money via Internet.

3

Example of McEliece and Niederreiter PKS based on Goppa Codes using Sage

Sage is a free open source alternative to common mathematics software. Sage — short for Software for Algebra and Geometry Experimentation — is a volunteer based project whose source code is laid open for inspection, see `http://www.sagemath.org`.
Using the idea of implementing code-based cryptosystems in Sage by Risse [32], this section will provide two examples of the best known code-based PKS, namely McEliece and Niederreiter.
We directly use some methods from the paper by Risse but at the same time extend others to a fully working implementation. We will construct a Goppa code from an irreducible Goppa polynomial with the computer algebra system Sage like in the paper. Furthermore we present a full implementation of Patterson's decoding algorithm for Goppa codes. Other than in [32] our environment is finally used to set up and demonstrate both, the McEliece and the Niederreiter PKS.
The Python-like Sage code will be displayed along with the output of the calculations.
The code presented below will on the one hand provide a Sage implementation of Goppa codes as well as methods for experimentation with the two code-based public key cryptosystems. At the same time the code will be evaluated and is available for instant typesetting in LaTeX— a handy feature. The calculations are done by Sage and the output can be easily displayed. This is yet another demonstration how different OpenSource projects marvelously interact, namely Sage and LaTeXvia the `sagetex` package.
Coming back to the example, we use the following parameters conformal with the notation of Section 1.4.2 to construct a binary Goppa code C over $F := \mathbb{F}_{2^m}$, $q = 2$ with the codeword length $n = 8$ and the degree of the field extension $m = 3$. The choice $t = 2 < \frac{n}{m}$ in combination with a monic, irreducible polynomial g of degree $\deg g = t$ will yield a code of dimension $k := 2 \geq n - tm$ and minimal distance $d = 5 \geq 2t + 1$. This results in the error correcting capability $2 = \lfloor \frac{d-1}{2} \rfloor = \lfloor \frac{5-1}{2} \rfloor$ of the code C.

```
n = 8; m = 3; k = 2; t = 2; q = 2

A = GF(q);                      # binary alphabet
F.<beta> = GF(q^m);             # field with primitive element beta
```

```
Ring = PolynomialRing(F,'z'); # polynomial ring over F in z
z = Ring.gen();               # the generator z of Ring

g = z^t+z+1;                  # Goppa polynomial of degree t

L = [0,1,beta,beta^2,beta^3,beta^4,beta^5,beta^6]; # locator set
```

The primitive element β fulfills $\beta^3 = \beta + 1$, as can be seen when looking closer at the support for the Goppa code $L = [0, 1, \beta, \beta^2, \beta + 1, \beta^2 + \beta, \beta^2 + \beta + 1, \beta^2 + 1]$ which is a full enumeration of all the elements in F.

```
def goppaCheck(g):
    for i in range(n):
        if g(L[i])==F(0):
           print 'alarm: g(x^'+str(i)+')=0';
           return False;
    return g.is_irreducible();
```

We see the Goppa polynomial $g(z) = z^2+z+1$ is irreducible in $F[z]$ and $g(\gamma) \neq 0,\ \forall \gamma \in L$ because `goppaCheck(g)` returns True. The next step is to define the parity check matrix of the generalized Reed-Solomon code as in Equation (1.4).

```
H_GRS = matrix([[L[j]^(i) for j in range(n)] for i in range(k)]);
H_GRS = H_GRS*diagonal_matrix([1/g(L[i]) for i in range(n)]);
```

$$H_{GRS} = \begin{pmatrix} 1 & 1 & \beta^2 & \beta^2+\beta & \beta^2 & \beta & \beta & \beta^2+\beta \\ 0 & 1 & \beta+1 & \beta^2+1 & \beta^2+\beta+1 & \beta^2+\beta+1 & \beta^2+1 & \beta+1 \end{pmatrix}$$

This next Sage block will compute the parity check matrix of C, which will be used for our example. The columns are expanded by substituting the field elements $\gamma \in F = \mathbb{F}_{2^m}$ with their representation over the alphabet $A = \mathbb{F}_2$ like in Section 1.3.5 about subfield subcodes.

```
H_Goppa = matrix(A,m*H_GRS.nrows(),H_GRS.ncols());

for i in range(H_GRS.nrows()):
    for j in range(H_GRS.ncols()):
        be = bin(eval(H_GRS[i,j].int_repr()))[2:];
        be = '0'*(m-len(be))+be; be = list(be);
        H_Goppa[m*i:m*(i+1),j] = vector(map(int,be));
```

$$H_{Goppa} = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Now we are able to compute a generator matrix G_{Goppa} as right kernel of H_{Goppa}.

```
Krnl = H_Goppa.right_kernel();
G_Goppa = Krnl.basis_matrix();
```

$$G_{Goppa} = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

The following construction shows that one would get the same code if this alternative parity check matrix is used. The notation H_{CXY} is taken from the proof of Theorem 1.15.

```
H_CXY = matrix([[1,0],[1,1]]);
H_CXY = H_CXY*H_GRS;

H_CXYGoppa = matrix(A,m*H_CXY.nrows(),H_CXY.ncols());

for i in range(H_CXY.nrows()):
    for j in range(H_CXY.ncols()):
        be = bin(eval(H_CXY[i,j].int_repr()))[2:];
        be = '0'*(m-len(be))+be; be = list(be);
        H_CXYGoppa[m*i:m*(i+1),j] = vector(map(int,be));


Krnl = H_CXYGoppa.right_kernel();
G_CXYGoppa = Krnl.basis_matrix();
```

$$H_{CXY} = \begin{pmatrix} 1 & 1 & \beta^2 & \beta^2+\beta & \beta^2 & \beta & \beta & \beta^2+\beta \\ 1 & 0 & \beta^2+\beta+1 & \beta+1 & \beta+1 & \beta^2+1 & \beta^2+\beta+1 & \beta^2+1 \end{pmatrix}$$

$$H_{CXYGoppa} = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

The same code C is obtained from both parity check matrices because their respective null-spaces — the two generator matrices — are equal:

$$G_{CXYGoppa} = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} = G_{Goppa}.$$

Since the field F has characteristic 2, Frobenius' automorphism makes the "freshmen's dream" come true — the equation $(a+b)^2 = a^2 + b^2$ is valid $\forall a, b \in F$. This fact will be used by Patterson's algorithm, to be more specific when a polynomial $p(z) \in F[z]$ is written as a sum of even and odd parts $p(z) = a(z)^2 + zb(z)^2$.
Before giving some procedures that are needed for the calculations in our example, first some words about solving the key equation (1.11) and therefore especially Equation (1.15) dealt with in Section 1.4.2.

The call `modifiedEEA(a,b, stop)` (with `stop=false`) computes (d, u, v) that satisfy $u \cdot a + v \cdot b = \gcd(a, b) = d$ — this is the common extended Euclidean algorithm. The values returned by the `modifiedEEA(a,b, stop)` in the case of stopping the EEA earlier (`stop=false`) are permuted (u, d, v), as the thorough reader has noticed. Despite the different notation in this implementation oriented chapter and the chapter discussing the mathematical theory earlier, it is clear that solutions u, d to $d = u \cdot a \mod b$ are to be found. In Section 1.4.2 Equation (1.15) looked like this: $u(z) \equiv v(z)r(z) \mod G(z)$, with unknown $u(z), v(z)$.
Here we do not care about v, since the equation $u \cdot a + v \cdot b = d$ will be regarded $\mod b$. The variables `u, lastu` respectively `r, lastr` in `modifiedEEA` only store the latest, important values of the two sequences z_h respectively r_h that have been discussed in detail in Section 1.4.2 satisfying the degree restrictions as in Equation (1.16). To resolve possible (but unfortunately unavoidable) confusion of notations through different chapters, we summarize:

$$z_j \longrightarrow u(z) \longrightarrow d,$$
$$r_j \longrightarrow v(z) \longrightarrow u,$$

hence it is correct to return the values accordingly. Here are the procedures:

```
def split(p): # Splits the polynomial p into even part u and odd part v,
              # such that p(z) = u(z)^2 + z*v(z)^2 holds.
    F = p.parent()
    u = F([sqrt(c) for c in p.list()[0::2]]);
    v = F([sqrt(c) for c in p.list()[1::2]]);
    return (u,v);


def modifiedEEA(a, b, stop):

    deg=a.degree();
    r = b;    lastr = a;
    u = 0;    lastu = 1;
    v = 1;    lastv = 0;

    while r <> 0:
        (lastr, (q, r)) = r, lastr.quo_rem(r);

        (u, lastu) = (lastu - q*u, u);
        (v, lastv) = (lastv - q*v, v);

        # test stop condition: Are the degrees reached?
        bool = u.degree()<=floor((deg-1)/2) and r.degree()<=floor(deg/2);

        if (stop and bool):
            (u,d,v)=(lastr, lastu, lastv);
```

```
            # here: deg u + deg d = deg b = t and ua=d mod b
            return (d,u,v);

    (d,u,v)=(lastr, lastu, lastv);
    # here the return values satisfy: d=ua+vb
    return (d,u,v);


def inverse(p,g):      # Returns the inverse of polynomial p mod g(z)
    (d,a,b) = xgcd(p,g);
    return a.mod(g);


def checkCorrectness(x,y):
    if (y==x):
        print 'corrected received word==sent word';
        return true;

    print 'alarm: corrected received word=',y,'<>',x,'=sent word';
    return false;
```

When trying to provide a fully functional implementation of Patterson's algorithm in Sage, one needs to work with a modified extended Euclidean algorithm. We are now able to present a working algebraic decoding algorithm based on Patterson's idea implemented in Sage. The routine `patterson` implements Algorithm 1 given in Section 1.4.2.

```
def patterson(y, Mat, binMat, McEliece):

    e = vector(A,n);    # empty length n error vector
    z = Ring.gen();     # generator z
    b = F.gen();        # generator beta

    if(binMat):
        syn = y; # conversion of binary representation to field elements
        syn = syn.list(); syn = syn[::-1];  # reverse list

        s = 0; pot = 1; # build syndrome polynomial s in F[z] of degree <k
        for i in range(k):
            tmp = 0; bj = 1;
            for j in range(m):
                ##syn = (beta^(m-1) ... beta^2 beta 1 | . . .)
                #mi = k-i-1; tmp += syn[mi*m+j]*bj;

                # syn = (1 beta beta^2 ... beta^(m-1) | . . .)
                tmp += syn[i*m+j]*bj;
                bj *= b;  # beta powers
```

```
        s += pot*tmp;
        pot *= z; # z powers for syn = (1 | z | . . . | z^(k-1))
else:
    syn = Mat*y;      # compute syndrome using the matrix Mat over F
    syn = syn.list();
    syn = syn[::-1];  # reverse list
    s = Ring(syn);    # represent s as polynomial of degree <k in F[z]

if s.is_zero():       # no errors occurred
    if(McEliece):
        return y;
    return e;

# compute the error locator polynomial sigma
sigma = z;            # initialize sigma with z
T = inverse(s,g);

if T<>sigma:
    (g0,g1) = split(g);
    w = g0*inverse(g1,g); w = w.mod(g);

    (T0,T1) = split(T+z);
    R = (T0+w*T1).mod(g);

    (d,u,v) = modifiedEEA(g,R, true);

    sigma = u^2+ z*v^2; print 'sigma =',sigma;

# mark zeros of sigma as errors
for i in range(n):
    if sigma(L[i])==F(0): # an error occurred at position i
        e[i] = 1;

if(McEliece):
    return y+e; # return corrected vector
return e;       # else: return error vector
```

The full implementation of Patterson's algorithm for binary Goppa codes along with definitions to encode plain text using McEliece's idea and a routine to check the correctness of our calculations enables us to start the first example.

3.1 McEliece PKS

The next two procedures help adding errors and therefore encrypting messages according to McEliece's idea.

```
def errors(t,n):          # Adds t random errors to a vector of length n.
    e = vector(A,n);
    for i in range(t):
        j = randint(0,n-1); e[j] = 1;
    return e;

def encryptMcEliece(u):   # Encrypts the message u
    c = u*G_pub;          # with the public key G_pub
    e = errors(t,n);      # and adds t errors
    y = c+e;
    return y,e;
```

To set up the McEliece PKS (see Section 2.2.1) we need to define the private key consisting of two matrices S and P.

```
S = matrix(A,k, [randint(0,1) for i in range(k^2)]);

while (rank(S)<k):
    S = matrix(A,k, [randint(0,1) for i in range(k^2)]);

rng = range(n); P = matrix(A,n);

for i in range(n):
    p = floor(len(rng)*random());
    P[i,rng[p]] = 1; rng = rng[:p]+rng[p+1:];
```

$$S = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, P = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

```
G_pub = S*G_Goppa*P;
```

Next we define the public matrix $G_{pub} = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$ with the help of S and P and let Alice generate a secret message $u \in A^k$ in order to transmit it to Bob.

```
u = vector(A,[randint(0,1) for i in range(k)]);  print 'message u=',u;
y,e = encryptMcEliece(u); print 'encrypted message y=',y;
```

The Sage output is the secret `message u=`$(0, 1)$ and moreover we are now able to transmit the `encrypted message y=`$(1, 0, 0, 0, 1, 1, 1, 0)$ over a public channel.
Before we will proceed with the demonstration how the receiver corrects the errors, some words on how to retrieve the information bits after that. We have to take a look at the generator matrix $G_{Goppa} = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$. We see from its structure that the second and third column are "systematic". We access the information bits by reading the (zero-based indices) 1 and 2, respectively. This is one issue that is not dealt with in a generic way for arbitrary Goppa codes with different parameters that have been specified in the beginning and hence has to be adapted individually for every case.
Bob receives y and decrypts the message using his private key P, S — or better the once and for all prepared inverted matrices P^{-1}, S^{-1} — and the parity check matrix H_{CXY}. The `patterson` method performs the decryption given the correct input for the McEliece setting.

```
# Bob receives y and decrypts it
print 'Bob receives y=',y;
yP = y*P.inverse(); print 'y*P^{-1}=',yP;

# McEliece=true, thus returns the already corrected codeword
# binMat=false, thus uses the provided Mat=H_CXY for syndrome calculation
yD = patterson(yP, H_CXY, false, true); print 'Bob decodes yD=',yD;

mm = vector(A,[yD[1],yD[2]]); # information bits are in positions 1 and 2
yS = mm*S.inverse();  print 'mm*S^{-1}=',yS;

checkCorrectness(yS,u);
```

The output of our McEliece example is

`Bob receives y:`	$y = (1, 0, 0, 0, 1, 1, 1, 0)$
`y*P^{-1}:`	$yP = (0, 0, 1, 0, 0, 1, 1, 1)$
`Bob decodes yD:`	$yD = (0, 0, 1, 1, 1, 1, 1, 1)$
`scrambled information bits mm:`	$(0, 1)$
`mm*S^{-1}:`	$yS = (0, 1)$

and finally `checkCorrectness(yS,u)` yields: True — the decryption $u = (0, 1) \rightarrow yS = (0, 1)$ was successful and the example is complete!

3.2 Niederreiter PKS

As in the previous example, we are able to use the `patterson` decoding method presented above to show how the Niederreiter PKS (see Section 2.2.2) works. We use the same parameters as before and thus work with the same Goppa code C. In this case the syndrome is input to the decoding algorithm, which thus needed to be adapted to handle decoding in the case without computing the syndrome using the parity check matrix as a first step. The following definitions help to carry out the calculations so we can set up the Niederreiter example.

```
def weightt(t): # admissible words are length n, weight t column vectors
    e = vector(A,n);
    for i in range(t):
        j = randint(0,n-1);
        while e[j]==1:
            j = randint(0,n-1);
        e[j] = 1;
    return e;

def encryptNiederreiter(u):
    c = u*H_pub.transpose();
    return c;
```

The construction of the public parity check matrix H_{pub} requires two private matrices M and P that hide the Goppa code structure from the public. M is a random regular matrix and P is a permutation thus we proceed constructing the public key as a disguised version of $H_{CXYGoppa}$ — the parity check matrix of C.

```
nk = (n-k);
M = matrix(A,nk, [randint(0,1) for i in range(nk^2)]);
while (rank(M)<nk):
    M = matrix(A,nk, [randint(0,1) for i in range(nk^2)]);

rng = range(n); P = matrix(A,n);

for i in range(n):
    p = floor(len(rng)*random());
    P[i,rng[p]] = 1; rng = rng[:p]+rng[p+1:];

H_pub = M*H_CXYGoppa*P;
```

We compute the public key H_{pub} and store the private key — or again for a performance gain — the inverse matrices of M respectively P for decoding reasons:

$$M = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}, P = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}.$$

$$H_{pub} = M \cdot H_{CXYGoppa} \cdot P = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

We let Alice generate some admissible plain text, which is a message encoded as vector u of length $n = 8$ and weight $t = 2$ in the Niederreiter setting.

```
# Assume the message M has been encoded as vector u:
u = weightt(t);              print 'message u=', u;

# Alice encrypts u with Bob's H_pub and sends y to him
y = encryptNiederreiter(u); print 'encrypted message y=',y;

# Bob receives y and decrypts it to a vector u and hence M
print 'Bob receives y=',y;

yM = M.inverse()*y;          print 'M^{-1}*y=',yM;

# yM can be interpreted as syndrome of the sought after error word
# McEliece=false, thus returns the error word necessary here
xD = patterson(yM, G_Goppa, true, false); print 'Bob decodes to xD=',xD;

x = P.inverse()*xD;          print 'P^{-1}*xD=',x;

checkCorrectness(u,x);
```

The output of our Niederreiter example is

`Bob receives y:`	$y = (0, 1, 1, 1, 1, 1)$
`M^{-1}*y:`	$yM = (0, 1, 1, 1, 0, 0)$
`Bob decodes xD:`	$xD = (1, 0, 0, 0, 0, 1, 0, 0)$
`P^{-1}*xD:`	$xS = (0, 0, 1, 0, 0, 0, 0, 1)$

and finally `checkCorrectness(u,x)` yields: True — the decryption was successful!
We summarize:

$$u = (0, 0, 1, 0, 0, 0, 0, 1) \rightarrow xS = (0, 0, 1, 0, 0, 0, 0, 1)\,,$$

which completes the example.

4 Quantum Computing

In the past the gain in computing power had a lot to do with miniaturization of the computer's components. We will soon be at the stage where it is not possible to further push this development because of physical limitations. No new, groundbreaking ideas but hard optimization work and thorough fine tuning of computer chips led to the improvements in the past decades.

Recently, in 2012, researchers from IBM Research Division at "Almaden Research Center" [21] were able to show that the incredible small number of 12 atoms are sufficient to store 1 bit of classical information.

In the introductory book [42, Ch. 1.1] Williams extrapolates the trend in miniaturization and claims that the one atom per bit storage requirement will soon be reached — around the year 2020. On such a small scale the best fitting model of physics tells us that quantum effects take over. Classical physics, which is good to explain macroscopic phenomena, is not appropriate to describe small scale particles — this is the domain of quantum physics.

If we desire further improvements of computational performance we need to control the negative side effects that occur at the quantum scale or exchange the classical approach that uses electronic circuits. This is why researchers discuss new computer architectures and — apart from other possible candidates — try to establish the quantum computer, which uses quantum effects in a new, beneficial way to carry out computations.

The focus of this thesis is not on physics, nevertheless some results shall be presented in this chapter to motivate the efforts made in investigating new cryptosystems and cryptanalyzing common ones. Furthermore, the question if classical cryptosystems are still needed once a functional quantum computation infrastructure is established, is dealt with.

A drawback we will see — similar to the previous discussion of how to establish secure information processing and transmission classically — is the gap between the theory and the application side. Unfortunately, the theory leaves some big questions unanswered, such that most of security proofs are based on assumptions that seem likely to be true but remain uncertain. Therefore we have to believe that the known attacks against a system cryptographers recommend parameters for are really the best so far and cannot be circumvented easily. On the other hand a possible solution to this dilemma providing perfect secrecy — namely quantum cryptography (see Section 4.6) — exists, but still we have to believe in the currently best fitting physical model of our world. As Williams puts

it in [42, Ch. 13.3.2]: "...the laws of quantum physics are a fundamental aspect of nature, verified experimentally to an exceedingly high level of precision, and are impossible to circumvent. If they were, then all sorts of bizarre implications would ensue, such as the ability to communicate messages faster than the speed of light".

4.1 Quantum Bit

The fundamental difference of classical information and quantum information is that the unit of quantum information — the quantum bit (or qubit for short) — is not restricted to just two states, say 0 and 1, like the classical bit. A qubit can represent 0 and 1, but also much more.

The theoretical model of a qubit can be based on any quantum system that has two distinguishable states, which do not change uncontrollably.

To read the value of a qubit, the quantum system needs to be measured. The laws of quantum mechanics tell us that measurement may change the system. Therein lies the main difference of the macroscopic storage and manipulation of bits and the microscopic implementation of a qubit.

Let $|0\rangle$ and $|1\rangle$ denote (in Bra-ket or Dirac notation) the two basis states of a fixed quantum system. Because of the quantum system's subatomic size, quantum mechanics shows us that a qubit is in more than just one of the two states $|0\rangle$ and $|1\rangle$. It is in both states simultaneously with a certain probability. The value of a qubit is fixed — to just one of those two distinct states — upon measurement.

Formally, a qubit is in a superposition of the basis states, say

$$|\psi\rangle = a_0\,|0\rangle + a_1\,|1\rangle, \quad |a_0|^2 + |a_1|^2 = 1$$

with complex coefficients $a_0, a_1 \in \mathbb{C}$. Measurement of a qubit in the state $|\psi\rangle$ does not give the coefficients a_0, a_1 to fully determine the current state; instead it yields only one of the basis states. The state $|0\rangle$ is obtained with probability $|a_0|^2$ and hence $|1\rangle$ with probability $|a_1|^2 = 1 - |a_0|^2$. Mathematically a qubit is modeled as a vector of length 1 in a 2-dimensional complex vector space and its canonical basis $\left\{|0\rangle = \binom{1}{0}, |1\rangle = \binom{0}{1}\right\}$ is called computational basis in this context. In this manner we can easily extend this interpretation to n-qubit quantum systems:

Let $|0\rangle, \ldots, |j\rangle, \ldots, |2^n - 1\rangle$ denote the basis states of a n-qubit quantum system, hence they represent vectors in $\underbrace{\mathbb{C}^2 \times \mathbb{C}^2 \times \cdots \times \mathbb{C}^2}_{n \text{ times}} = \mathbb{C}^N$, where $N = 2^n$. Following [42, 1.4.1] we write $|j\rangle$ for the computational basis state corresponding to the decimal number j. Sometimes it is handier to use the binary representation that is filled with 0 from the left

to length n if necessary.

$$\begin{aligned}
|0_{dec}\rangle = |0\ldots000_{bin}\rangle &= (1,0,0,0,\ldots,0)^T \\
|1_{dec}\rangle = |0\ldots001_{bin}\rangle &= (0,1,0,0,\ldots,0)^T \\
|2_{dec}\rangle = |0\ldots010_{bin}\rangle &= (0,0,1,0,\ldots,0)^T \\
|3_{dec}\rangle = |0\ldots011_{bin}\rangle &= (0,0,0,1,\ldots,0)^T \\
&\vdots \\
|(N-1)_{dec}\rangle = |\underbrace{1\ldots111}_{n}{}_{bin}\rangle &= (\underbrace{0,0,0,0,\ldots,1}_{N=2^n})^T
\end{aligned}$$

A general state in such an n-qubit quantum system is, analogously to the 1-qubit case, a length 1 vector with complex coefficients:

$$|\psi\rangle = \sum_{j=0}^{N-1} a_j|j\rangle, \quad a_j \in \mathbb{C}, \quad \sum_{j=0}^{N-1} |a_j|^2 = 1.$$

We remark that by using quantum registers with more than one qubit, a new phenomenon called entanglement appears. Such composite quantum systems can be in states that are not representable as a product of its component systems states and are thus said to be "entangled". At the end of Chapter 2.2.8 in their standard work [26] the authors Nielsen and Chuang go so far as to say: "For reasons which nobody fully understands, entangled states play a crucial role in quantum computation", which shows the importance of research in this emerging area.
Some seemingly inherent properties of the well known information theoretic unit bit turns out to be valid in the macroscopic domain of classical information processing only. They are not necessarily true at the quantum scale; in fact some properties may seem unfamiliar from a macroscopic point of view since they are not true anymore. Table 4.1 contrasts some classical assumptions with the quantum mechanical facts (see [42, Table 1.2]).

Assumptions: Classically true statement...	...false on a quantum scale
A bit always has a definite value.	Definite only value after it is read.
A bit can only be 0 or 1.	A qubit can be in a superposition.
A bit can be copied not affecting its value.	A qubit in an unknown state can't be copied without necessarily changing its state.
A bit can be read without affecting its value.	Reading a qubit in a superposition of 0 and 1 will change it.
Reading one bit in the computer has no affect on any other (unread) bits in the memory.	Reading one entangled qubit will necessarily affect the other qubit.
To compute the result of a computation, one must run the computer.	False, there are architectures that need not be "turned on" to operate.

Table 4.1: Assumptions about the properties of bits that are no longer true for qubits.

4.2 Quantum Computer

Apart from the theoretical model of a quantum computer and its capability the question of an actual physical realization arises. To put it short; it seems hard to build a stable, powerful quantum computer to date. Small scale examples do exist in a lab environment and still a lot of research is done in this field which may lead to a functional and practical quantum computer in the near future.
The 2012 physics Nobel prize was awarded to Wineland and Haroche for their "work on understanding the quantum world — work which may eventually help make quantum computing possible".
Wineland [14] put it that way: "To build such a quantum computer is an enormous practical challenge. One has to satisfy two opposing requirements: the qubits need to be adequately isolated from their environment in order not to destroy their quantum properties, yet they must also be able to communicate with the outside world in order to pass on the results of their calculations. Perhaps the quantum computer will be built in this century. If so, it will change our lives in the same radical way as the classical computer transformed life in the last century.".
DiVincenzo published criteria [12] to determine whether a quantum system meets the requirements for a practical quantum computer and quantum information. Here are his "Five (plus two) requirements for the implementation of quantum computation".

1. A scalable physical system with well characterized qubits; which means the number of qubits must not be bounded.

2. The ability to initialize the qubits to a simple state, such as $|\psi\rangle = |00\ldots0\rangle$.

3. Long relevant decoherence times, much longer than the gate operation time; which ensures the stability of the system and addresses the possible collapse of states.

4. A universal set of quantum gates; which means there should be a small number of implemented operations, with which it is possible to model any unitary transformations — the core of most quantum algorithms.

5. A qubit-specific measurement capability; to read out the result of a computation.

6. The ability to inter-convert stationary and flying qubits; which is important for the exchange of quantum information.

7. The ability faithfully to transmit flying qubits between specified locations; to process information over distances.

For an overview about some quite different ideas how to implement qubits, we refer to [42, Ch. 15]. There, a few possible realizations are discussed and it is shown that quantum computers based on these quantum systems are polynomially equivalent to one another. This means one can simulate any other quantum computer (efficiently) once a particular architecture has been successfully built.
Finally, quantum computers require quantum algorithms to operate intelligible upon information. This is where mathematicians once more come into play. Assuming the existence

of a functional quantum computer, mathematicians started to develop algorithms suitable to perform tasks on this new computer architecture some time ago. It turns out that some problems can be solved in significantly less running time by quantum algorithms compared to classical algorithms deployed on classical electronics circuit hardware.

Quantum Parallelism

What is the main property that makes a quantum computer so powerful?
A quantum computer represents 2^n values using n qubits. A quantum gate, given as some reversible function f, applied to these n qubit therefore takes $\mathcal{O}(n)$ time. Preceding measurement to read the result of this computation at this stage would yield only one value $f(x)$ for some value x, although theoretically f has been evaluated for every one of the 2^n input values. Thus quantum parallelism, as this phenomenon is called, on its own is not immediately useful. Quantum parallelism needs to be combined with another inherent property of quantum systems — interference.
If there are alternative, correct solutions to a computation, their interference can be used in a clever way to increase the probability to measure its value. We refer to [42, Ch. 1.4.3] for a detailed outline and the modern view that there exists a superposition of values stored in a quantum register which influences the probability of measuring a specific value.
As Nielsen and Chuang put it in their book [26, Ch. 1.4.3]: "The difference is that in a classical computer [these two] alternatives forever exclude one another; in a quantum computer it is possible for the [two] alternatives to interfere with one another to yield some global property of the function f".
A classical computer on the other hand with a n bit register needs 2^n processors working in parallel doing 1 operation. Equivalently, 1 processor can be used repeating some calculation $\mathcal{O}(2^n)$ times in order to perform a single gate operation on each of the 2^n values representable by n bits.
The authors Rieffel and Polak of the survey [31] describe it this way: "Quantum parallelism circumvents the time/space trade-off of classical parallelism through its ability to provide an exponential amount of computational space in a linear amount of physical space".
Exploiting these properties of quantum systems intelligibly leads to the computational superiority of quantum computers over the classical computer architecture. We remark that the set of functions that are computable by quantum computers is the same as the functions that are computable by classical computers. In this sense these two architectures are equally powerful.

4.3 Quantum Algorithms

Analogously to classical computing, where manipulation of bits are necessary to perform computations, quantum computations requires the controlled manipulation of qubits. The basic manipulations of qubits are called quantum circuits or quantum gates and are unitary transformations speaking in mathematical terms. A sequence of quantum gates, with the goal to compute a certain function, is then called quantum algorithm.

Instead listing many different quantum algorithms, we will only show the most important algorithms or their main ideas. The essence of quantum computation should become visible here. We first provide an early predecessor that showed the astonishing capability of quantum computers compared to classical computers, which initially gave rise to this field.

4.3.1 Algorithm of Deutsch-Jozsa

Deutsch's algorithm is a deterministic quantum algorithm, to decide whether a function is constant or balanced. The formulation in the easiest case is as follows.

Let $f : \{0,1\} \to \{0,1\}$ be a function operating on bits. f is said to be constant if $f(0) = f(1) \in \{0,1\}$ and else balanced, since 0 and 1 both appear in the image.

Although Deutsch's algorithm is of no great practical relevance, it has shown the capability of quantum computers to solve certain problems faster than classical computers.

First we need to make the oracle function f (no full definition available but an algorithm providing the result upon giving an input) to be a unitary quantum gate — an operation suitable for a quantum computer. Thus define the invertible function (see [42, 1.7.2])

$$U_f|x\rangle|y\rangle \mapsto |x\rangle|f(x) \oplus y\rangle,$$

where $\oplus$ denotes the XOR operation defined as $1 \oplus 1 = 0 \oplus 0 := 0$ and $1 \oplus 0 = 0 \oplus 1 := 1$. This is a function from $\{0,1\}^2$ to $\{0,1\}^2$ called "f-controlled-NOT" gate (or CNOT for short).

The idea of Deutsch was that instead of classically evaluating the function 2 times in order to be sure which case — balanced or constant — it is, to use the following equivalence in a clever way to achieve the same result: $f(0) = f(1) \Leftrightarrow f(0) \oplus f(1) = 0$.

Another tool, in preparation for the first quantum algorithm here in this thesis, is the Hadamard transform written as matrix $H := \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$. This means the two basis states $|0\rangle, |1\rangle$ are mapped on an equally weighted superposition state $H|0\rangle = H\binom{1}{0} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $H|1\rangle = H\binom{0}{1} = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. Since $H = H^T = H^{-1}$ is an unitary matrix, we have:

$$H\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = HH|1\rangle = HH^{-1}|1\rangle = |1\rangle, \tag{4.1}$$

$$H\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = HH|0\rangle = HH^{-1}|0\rangle = |0\rangle. \tag{4.2}$$

The idea of creating these superpositions of all currently representable values can be generalized using n Hadamard gates on a register holding n qubits to establish such an equally weighted superposition of all $0, 1, \ldots, 2^n - 1$ values.

The algorithm (basically taken from [39]) can be stated like this:

Initialize $|\psi_1\rangle = |x\rangle|y\rangle \leftarrow |0\rangle|1\rangle$, and apply the Hadamard transform on both registers.

$$
\begin{aligned}
|\psi_2\rangle = |x\rangle|y\rangle &\leftarrow H|x\rangle H|y\rangle \\
&= \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right) \cdot \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \\
&= \frac{1}{2}\Big(|0\rangle|0\rangle - |0\rangle|1\rangle + |1\rangle|0\rangle - |1\rangle|1\rangle\Big). \\
|\psi_3\rangle = |x\rangle|y\rangle &\leftarrow U_f|x\rangle|y\rangle \\
&= \frac{1}{2}\Big(|0\rangle|0 \oplus f(0)\rangle - |0\rangle|1 \oplus f(0)\rangle\Big) + \\
&+ \frac{1}{2}\Big(|1\rangle|0 \oplus f(1)\rangle - |1\rangle|1 \oplus f(1)\rangle\Big) = \\
&= \frac{1}{2}\Big(|0\rangle \cdot (|f(0)\rangle - |1 \oplus f(0)\rangle)\Big) + \\
&+ \frac{1}{2}\Big(|1\rangle \cdot (|f(1)\rangle - |1 \oplus f(1)\rangle)\Big) = \\
&= \frac{1}{2}\Big((-1)^{f(0)}|0\rangle \cdot (|0\rangle - |1\rangle) + (-1)^{f(1)}|1\rangle \cdot (|0\rangle - |1\rangle)\Big) = \\
&= \frac{1}{2}\Big((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle\Big) \cdot (|0\rangle - |1\rangle).
\end{aligned}
$$

$|\psi_3\rangle$ is the state after evaluating U_f, which can be cleverly rewritten, as can be seen above to yield the result in the last line. We see that $|\psi_3\rangle = \pm\frac{1}{2}(|0\rangle + |1\rangle) \cdot (|0\rangle - |1\rangle)$ if and only if f is constant and $|\psi_3\rangle = \pm\frac{1}{2}(|0\rangle - |1\rangle) \cdot (|0\rangle - |1\rangle)$ otherwise.

We apply the Hadamard gate to the first register of $|\psi_3\rangle$, say $|\psi_4\rangle = |x\rangle \leftarrow H|x\rangle$. The fact mentioned in Equation (4.2) tells us that if we measure $|x\rangle$ and the output is $|0\rangle$, then f is definitely a constant function and balanced otherwise. This trick allows to decide the Deutsch problem by evaluating f once instead of 2 times.

The appealing strength of this algorithm can be seen, when it is generalized to the $n = 2k$ qubits version called after Deutsch-Jozsa. Let $f : \{0,1\}^n \to \{0,1\}$ and assume f is guaranteed to be either constant or balanced, meaning exactly $\frac{n}{2} = k$ values are mapped to zero and the rest (again k values) therefore to 1. The algorithm (again adapted from [39]) can be stated as follows, where as usual $N = 2^n$:

Initialize a $(n+1)$ qubit register with zeros but set the last qubit to 1. Next apply the Hadamard transform H on each qubit individually and regard the first n qubits separately.

$$
\begin{aligned}
|\psi_1\rangle = |x\rangle|y\rangle &\leftarrow |0\rangle|0\rangle \dots |0\rangle|1\rangle. \\
|\psi_2\rangle = |x\rangle|y\rangle &\leftarrow H|0\rangle H|0\rangle \dots H|0\rangle H|1\rangle \\
&= (H \otimes H \otimes \dots \otimes H)|x\rangle H|1\rangle = \\
&= \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle \cdot \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \\
&= \frac{1}{\sqrt{2N}} \sum_{x=0}^{N-1} |x\rangle \cdot (|0\rangle - |1\rangle). \\
|\psi_3\rangle = |x\rangle|y\rangle &\leftarrow U_f|x\rangle|y\rangle \\
&= \frac{1}{\sqrt{2N}} \sum_{x=0}^{N-1} |x\rangle(|f(x)\rangle - |1 \oplus f(x)\rangle) = \\
&= \frac{1}{\sqrt{2N}} \sum_{x=0}^{N-1} (-1)^{f(x)}|x\rangle(|0\rangle - |1\rangle) = \\
&= \left(\frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} (-1)^{f(x)}|x\rangle\right) \cdot \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \\
|\psi_4\rangle = |x\rangle|y\rangle &\leftarrow (H \otimes H \otimes \dots \otimes H)|x\rangle I|y\rangle \\
&= \left(\frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} (-1)^{f(x)} \frac{1}{\sqrt{N}} \sum_{z=0}^{N-1} (-1)^{g(x,z)}|z\rangle\right) \cdot \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \\
&= \left(\frac{1}{N} \sum_{z=0}^{N-1} \sum_{x=0}^{N-1} (-1)^{g(x,z)+f(x)}|z\rangle\right) \cdot \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle).
\end{aligned}
$$

In the fourth step, at the assignment of $|\psi_4\rangle$, we apply Hadamard transforms to the first n qubits again and the identity I to the last qubit leaving it unchanged. The function $g(x,z) := x_0z_0 + x_1z_1 + \dots + x_{n-1}z_{n-1} \mod 2$ is the sum of the bitwise product. According to Equations (4.1) and (4.2) for $x = 0, 1$ we can write $H|x\rangle = \frac{1}{\sqrt{2}}(|0\rangle + (-1)^x|1\rangle)$. Generalizing this idea to a product of 2 Hadamard transformed qubits (and by induction to a product of n Hadamard transformed qubits) yields:

$$
\begin{aligned}
H|x\rangle H|z\rangle = H|x\rangle \otimes H|z\rangle &= \frac{1}{\sqrt{2}}(|0\rangle + (-1)^x|1\rangle) \cdot \frac{1}{\sqrt{2}}(|0\rangle + (-1)^z|1\rangle) = \\
&= \frac{1}{2}((-1)^{0+0}|00\rangle + (-1)^{x+0}|10\rangle) + (-1)^{0+z}|01\rangle) + (-1)^{x+z}|11\rangle).
\end{aligned}
$$

Thus the final step in determining whether f is balanced or constant is to measure the contents of the first register. The claim is that $|z\rangle = |00\dots0\rangle$ if and only if f is constant and some other result $|z\rangle \neq |00\dots0\rangle$ if f is balanced.

The amplitude of $|z\rangle = |00\ldots0\rangle$ is given by

$$\left|\frac{1}{N}\sum_{x=0}^{N-1}(-1)^{g(x,z)+f(x)}\right|^2 = \left|\frac{1}{N}\sum_{x=0}^{N-1}(-1)^{f(x)}\right|^2$$

because $g(x,z)=0$ here. We directly observe that a constant value $f(x)=0,\ \forall x$, respectively $f(x)=1,\ \forall x$ yields $|\pm 1|^2 = 1$ — certainty to measure $|z\rangle = |00\ldots0\rangle$!

In the case that f is balanced the 1 and the (-1) in the sum will cancel each other to yield a zero possibility of measuring $|z\rangle = |00\ldots0\rangle$.

Hence, this quantum algorithm needs only one evaluation of f (or more precisely U_f) in step three and some overhead to answer the Deutsch-Jozsa problem.

Classically f needs to be evaluated $\frac{1}{2}2^n + 1 = 2^{n-1}+1$ times to answer the Deutsch-Jozsa problem. We saw that the generalized quantum algorithm for f with n qubits input decides whether it is constant or balanced with only 1 function evaluation — a huge improvement compared to the exponential number of $2^{n-1}+1$ evaluations in the classical case!

4.3.2 Quantum Fourier Transform

Let $|0\rangle, |1\rangle, \ldots, |N-1\rangle$ denote the N basis states and let $\omega_N := e^{\frac{2\pi i}{N}} \in \mathbb{C}$ be the $N-$th root of unity, which means $\omega_N^k = 1$ for $k = N$ (and $k = 0$ of course) but $\omega_N^k \neq 1$ for $0 < k < N$. Since it allows an easier description we assume $N = 2^n$.

We represent the quantum state $|\psi\rangle = \sum_{j=0}^{N-1} a_j|j\rangle$ as a vector of the coefficients in the computational basis $(a_0, a_1, \ldots, a_j, \ldots, a_{N-1})^T \in \mathbb{C}^N$ and additionally require the amplitudes to be normalized $\sum_{j=0}^{N-1}|a_j|^2 = 1$.

The quantum Fourier transform (QFT) can be viewed as the discrete Fourier transform (DFT) with a normalization factor. As well as the DFT we can write the QFT as matrix:

$$F = \frac{1}{\sqrt{N}}\begin{pmatrix} \omega_N^{0\cdot0} & \omega_N^{0\cdot1} & \ldots & \omega_N^{0\cdot(N-1)} \\ \omega_N^{1\cdot0} & \omega_N^{1\cdot1} & \ldots & \omega_N^{1\cdot(N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_N^{(N-1)\cdot0} & \omega_N^{(N-1)\cdot1} & \ldots & \omega_N^{(N-1)\cdot(N-1)} \end{pmatrix}.$$

It turns out the QFT is unitary $F^{-1} = F^T$ and it is thus guaranteed to be implementable on a quantum computer.

The QFT is defined to act on the basis states $|j\rangle,\ 0 \leq j < N$ in the following way

(compare with Section 4.1 for the basis state representation as a vector):

$$\begin{aligned}
\mathrm{QFT}_N\,|j\rangle &:= F \cdot (0,\ldots,0,1,0\ldots,0)^T = && (4.3)\\
&= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega_N^{k\cdot j} |k\rangle = && (4.4)\\
&= \frac{1}{\sqrt{N}} \sum_{k_1=0}^{1} \sum_{k_2=0}^{1} \cdots \sum_{k_n=0}^{1} \omega_N^{j\cdot(2^n \sum_{l=1}^{n} k_l 2^{-l})} |k_1 k_2 \ldots k_n\rangle = && (4.5)\\
&= \frac{1}{\sqrt{N}} \sum_{k_1=0}^{1} \sum_{k_2=0}^{1} \cdots \sum_{k_n=0}^{1} \prod_{l=1}^{n} e^{(2\pi i j k_l 2^{-l})} |k_l\rangle = && (4.6)\\
&= \frac{1}{\sqrt{N}} \prod_{l=1}^{n} \left(\sum_{k_l=0}^{1} e^{(2\pi i j k_l 2^{-l})} |k_l\rangle \right) = && (4.7)\\
&= \frac{1}{\sqrt{N}} \prod_{l=1}^{n} \left(|0\rangle + \omega_{2^l}^{j} |1\rangle \right). && (4.8)
\end{aligned}$$

The result can be obtained by representing the integers $0 \leq k < N$:

$$(k)_{dec} = (k_1 k_2 \ldots k_n)_{bin} = \sum_{l=1}^{n} k_l 2^{n-l} = 2^n \sum_{l=1}^{n} k_l 2^{-l}$$

as binary fractions (see [42, 3.4.4]). After a few calculation steps and the fact that $\omega_N := e^{\frac{2\pi i}{N}}$ we used for Equation (4.5) we have matrix product representation of QFT_N.
Additionally, this result especially shows that the QFT of a computational basis state is a direct product of single qubit states and is, therefore by definition, unentangled!
Finally, using the linearity of quantum operations we can transform any superposition of computational basis states. Application of the QFT operator to a quantum state given as a vector of amplitudes in the basis states will yield "a new state vector (in the same basis) that will be peaked in probability amplitude at frequencies which contribute the most strongly to the signal" (see [42, Ch. 3.4.5]) — a quite useful fact!
The fast Fourier transform (FFT) — a clever divide and conquer implementation of the DFT — can classically be implemented to use $\mathcal{O}(N \log N) = \mathcal{O}(2^n n)$ operations, whereas the QFT can be implemented on a quantum computer using $\mathcal{O}(\log^2 N) = \mathcal{O}(n^2)$ operations — which is an exponential speedup!
The importance of the QFT lies in the fact that most quantum algorithms that demonstrated the astonishing exponential speedup compared to their classical counterparts use this transformation at some point of the computation.

4.3.3 Grover's Algorithm

The task is that we want to find $x = w$ that satisfies a certain search criterion, under the N different elements $|x\rangle$ of a given unsorted database. The input is a function f given as an oracle $f : \{0,1\}^N \to \{0,1\}$ for a search space of a priori unknown structure as well as an element w of the search space. The oracle f returns $f(w) := 1$ if the sought-after

element was found and $f(x) := 0, x \neq w$ otherwise. It is assumed that to call the oracle for an answer is possible in polynomial time.
Without going into further detail (see [42, Ch. 5.3] for details and a geometric viewpoint), we state the algorithm in pseudo-code as well as give a comprehensive, non-technical description here.

Algorithm 9: Grover's algorithm

Input : Database with N unsorted entries $|x\rangle$ and the sought-after $|w\rangle$.
Output: The complete database entry matching the search criterion.

Initialize the $\log N$-size qubit register with the state $|\psi\rangle = \frac{1}{\sqrt{N}} \sum_x |x\rangle$.

foreach $1 \leq j \leq \frac{\pi\sqrt{N}}{4}$ **do**
    GroverIteration() ; `// uses the 'amplitude amplification' technique`
    `// Each iteration step increases the probability of measuring` $|w\rangle$.
end
The final measurement yields a solution with high probability.
return Database entry belonging to $|w\rangle$.

We present a metaphor by the spouse of a quantum computer scientist Fuchs that describes Grover's algorithm in non-technical terms or even cuisine analogies: "Grover's quantum searching technique is like cooking a soufflé. You put the state obtained by quantum parallelism in a "quantum oven" and let the desired answer rise slowly. Success is almost guaranteed if you open the oven at just the right time. But the soufflé is very likely to fall – the amplitude of the correct answer drops to zero – if you open the oven too early."
The significance of Grover's algorithm for this thesis lies in its applicability to $\mathcal{NP}$ problems and limitations from a computational complexity theoretic viewpoint. In the article [1] the authors showed that this general unstructured search problem — in which any $\mathcal{NP}$ problem can be reformulated — stated above can not be solved faster than $\mathcal{O}(\sqrt{N})$ by a quantum computer.
As indicated by Algorithm 9, the iteration step (which can be done in polynomial time) needs to be carried out $\mathcal{O}(\sqrt{N})$ times. Thus Grover provided an algorithm that achieves this lower bound of running time. The algorithm additionally needs $\mathcal{O}(\log N)$ storage.
To summarize, Grover found a quantum algorithm that can be applied to any $\mathcal{NP}$ problem reformulated as a search problem. It yields a quadratic speed-up in the generic case. As application to cryptography we remark that this algorithm can readily be used to search for secret keys protecting data of arbitrary cryptosystems. This is a possible threat to cryptosystems that forces key lengths essentially to be doubled in order to remain at the same level of security they have been designed to for classical computer architecture.

4.3.4 Shor's Algorithm

We proceed with one of the most famous quantum algorithms. In the year 1994 Peter Shor gave the first algorithm for quantum computers with major significance for actual applications and in 1996 he published an enhanced version of the paper [35].

Generally speaking Shor's algorithm is a probabilistic algorithm that is composed of two parts — a classical part and a quantum algorithmic part.
In this section we use the letter N for a non-negative composite integer, which is common practice in number theory and comes handy for the complexity theory considerations too. This will not lead to confusion with the N that used to be a power of 2 in previous sections to make the descriptions easier there.

The Hidden Subgroup Problem

The original algorithm is able to decompose a non-negative composite integer N in its prime factors on a quantum computer — which is most interesting if N is product of exactly two prime numbers. The algorithm has a running time considerably shorter than any known algorithm to do the same task to date and can also be applied to solve the related DLP. In fact, both problems can be seen as instances of the more general "hidden subgroup problem" (HSP). Shor's idea can be used to find the hidden subgroup $H \leq G$, where G is an abelian group.

The Factorization Problem

In this section we present Shor's algorithm to factorize the number $N = 15$ as an example. In general we want to find the prime factors of a non-negative number N.
With a preprocessing step that can done on a classical computer, we can assume that N is a non-negative composite integer and therefore a product of primes. Additionally — running polynomial time tests — we can ensure that N is not a prime power. The hardest case that remains is that N consists of only two of prime factors such that $N = pq$. Since Shor's algorithm is probabilistic, there are cases when it returns the trivial factors $p = 1, q = N$. Fortunately the probability for these events is $< \frac{1}{2}$ and the algorithm can be run several times to increase the chances of finding a proper prime factor p.

Classical Part

The classical part of the algorithm consists of well known number theoretic facts and the correctness is easily verified. During Algorithm 10 the values of x and r satisfy the following: $N|(x^r-1) = (x^{\frac{r}{2}}-1)(x^{\frac{r}{2}}+1)$ and $N \nmid (x^{\frac{r}{2}}-1)$ because r is the order of $x \mod N$. Since the condition $N \nmid (x^{\frac{r}{2}}+1)$ holds in the last step of the algorithm, there is a common factor p in N and $(x^{\frac{r}{2}}-1)$. If $p = N$ then $N|(x^{\frac{r}{2}}-1)$ which contradicts the construction of r as the order of x. If $p = 1$ then the extended Euclidean algorithm can be used to obtain numbers u, v such that $(x^{\frac{r}{2}}-1)u + Nv = \gcd((x^{\frac{r}{2}}-1), N) = 1 = p$. Multiplying this equation by $(x^{\frac{r}{2}}+1)$ implies that $N|(x^{\frac{r}{2}}+1)$ — impossible by construction.
Thus Algorithm 10 yields a proper factor p of N.

Quantum Part

The task of the quantum part is, given $x \in G = (\mathbb{Z}_N^*, \cdot)$ find the order r of x. This is the number of elements in the generated subgroup $H = \{x^j \mod N \mid j \in \mathbb{Z}\}$.

Algorithm 10: Classical part of Shor's algorithm

Input : $N = pq$
Output: prime factors p, q

Randomly choose a number $x,\ 1 < x < N$.
if $1 < p := \gcd(x, N) < N$ **then**
  return $p, N/p$; `// It is extremely unlikely to find a factor here.`
end
Quantum part of Shor's algorithm computes the order r of $x \in \mathbb{Z}_N^*$
`//` Hence $x^r \equiv 1 \mod N$ `or` $N|(x^r - 1)$.
if *r is odd or* $x^{\frac{r}{2}} \equiv -1 \mod N$ **then**
  Restart the algorithm ; `//` Another $x,\ 1 < x < N$ `may be suitable.`
end
Compute $p := \gcd(x^{\frac{r}{2}} - 1, N)$ and $q := N/p$.
`//` Since $(x^{\frac{r}{2}} - 1)(x^{\frac{r}{2}} + 1) = x^r - 1$ `these` p `and` q `are the two factors.`
return p, q

Most authors do not present this part in full detail, since it is a bit technical. Instead they present some versions which cover special cases that simplify the notation. We follow this attitude and present merely an example for factorizing $15 = 3 \cdot 5$, written in terms of quantum computations. Apart from this we recommend [42, Ch. 6.3] for a more detailed yet still not fully generic coverage of Shor's algorithm. Finally, the following example can be found in [42, Ch. 6.3.2] where additional visualizations are provided.
The reasons for the example-based approach is that first of all it gives a quicker insight about what happens than to discuss the generic case. Secondly, the number 15 was the first number to be factored by this algorithm on an actual quantum computer (see the 2001 milestone in Section 4.4). Last but not least, the choice $N = 15$ is justified because it is the first number that is not a prime power or even and thus the calculation of the order is non-trivial.
Let us assume the classical part chose $x = 13$ at random which is co-prime to $N = 15$.

1) Choose $Q = 2^t$, such that $N^2 \leq Q \leq 2N^2$ holds. $Q = 2^8 = 256$ fulfills the inequality and a quantum register $|\phi_1\rangle$ with $t = 8$ qubits — the domain register — as well as a second image register $|\phi_2\rangle$ is initialized with zeros:
$$|\psi_1\rangle = |\phi_1\rangle|\phi_2\rangle = |0\rangle|0\rangle.$$

2) Assign an equally weighted superposition of all Q numbers to the first register $|\phi_1\rangle$:
$$|\psi_2\rangle = \frac{1}{\sqrt{Q}} \sum_{j=0}^{Q-1} |j\rangle\,|0\rangle\,.$$

3) Compute the modular powers $f(x) := x^j \mod N$ at once for all $0 \leq j < Q$ using quantum parallelism via the quantum gate realization U_f of f:
$$|\psi_3\rangle = \frac{1}{\sqrt{Q}} \sum_{j=0}^{Q-1} |j\rangle\,\left|x^j \mod N\right\rangle\,.$$

$|\phi_2\rangle$ thus contains the powers $13^j \mod 15, j = 0, 1, 2, \ldots$, or more specific, an equally weighted superposition of states representing the values $1, 13, 4, 7, 1, 13, \ldots$, which obviously is a periodic sequence with period 4.

4) Measure the contents of the second register $|\phi_2\rangle$ and assume one obtains the result b. This has the instant effect that $|\phi_1\rangle$ holds an equally weighted superposition of only states representing j, such that $b \equiv x^j \mod N$.

   Assume we measured the decimal value 1, then $|\phi_1\rangle = \lambda(|0\rangle + |4\rangle + |8\rangle + |12\rangle + \ldots)$.

5) Applying the inverse of the QFT on the first register $|\phi_1\rangle$ produces peaks at integer multiples of $\frac{1}{r}$ — reciprocals of the unknown period r.

   Application of the quantum gate (with ω_Q a Q-th root of unity) thus yields:

$$|\psi_4\rangle = \frac{1}{Q} \sum_{j=0}^{Q-1} \sum_{k=0}^{Q-1} \omega_Q^{kj} \; |k\rangle \; |x^j \mod N\rangle.$$

6) Steps 2) to 5) need to be iterated $\mathcal{O}(t) = \mathcal{O}(\log Q)$ times in order to produce, say the following measurement output: (128, 64, 0, 192, 0, 128, 128, 64, 0, 192, 192, 64). Non-zero values help to find out the period:

$$\frac{64}{256} = \frac{1}{4}, \frac{128}{256} = \frac{1}{2} = \frac{2}{4}, \frac{192}{256} = \frac{3}{4},$$

   therefore the period of the sequence and thus the order of $x \in \mathbb{Z}_N^*$ is $r = 4$ here.

In general the result y of the measurement of the first register in step 6) does not necessarily yield such values, where the period can be directly obtained. Non-zero values are only an approximation of multiples of the reciprocals of the period. In [42, Ch. 6.3.1] it is elaborated in more detail that usually one needs the continued fraction expansion of the number $y/Q = [a_0; a_1, a_2, \ldots]$, in order to determine the period r. With this sequence $a_0, a_1, \ldots$ one can define two recursions that help to approximate the fraction y/Q:

$$y/Q = [a_0; a_1, a_2, \ldots] = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \ddots}},$$

$$h_l = a_l h_{l-1} + h_{l-2}, \quad h_{-1} = 1, h_{-2} = 0,$$
$$k_l = a_l k_{l-1} + k_{l-2}, \quad k_{-1} = 0, k_{-2} = 1.$$

The main idea is to compute the truncated continued fraction expansion of $y/Q = [a_0; a_1, \ldots, a_l] = h_l/k_l$ until $k_l \geq N$ for the first time. The denominator k_{l-1} of the so-called convergents $(h_l/k_l)_{l \geq -2}$ is the sought-after period.
Just as a small example assume the value $y = 66$ would have been read in Step 6) instead of (the more likely result of) 64. The truncated continued fraction expansion $[0, 1/3, 1/4, 8/31]$ stopped after the first denominator was larger than $N = 15$ and hence yields the result $r = 4$.

With the order at hand from the quantum part, we can finish off the example and continue with the next (classical) steps of Algorithm 10 to factorize $N = 15$. Since $r = 4$ is even and $13^{\frac{4}{2}} \not\equiv -1 \mod 15$ we obtain $p = \gcd(13^2 - 1, 15) = 3$ and $q = \frac{15}{3} = 5$.
We remark the probabilistic nature of Shor's algorithm: If the random number $x = 14$ would have been chosen at random in the beginning, then we would have got the period $r = 2$. Since $(14^1 + 1) = 15$ the algorithm would restart before the last step and thus skip the obviously trivial factorization $p = \gcd(14^1 - 1, 15) = 1$ and $q = 15/1 = 15$.
To sum the results up: If there was a quantum computer with a quantum register of size $\log Q = 2 \log N \in \mathcal{O}(\log N)$, where $N^2 \leq Q \leq 2N^2$, Shor's algorithm can efficiently be applied to factorize N and therefore break RSA in $\mathcal{O}((\log N)^3)$ quantum gate operations. The terms sub-exponential as well as super-polynomial are used to describe the running time of the best known classical algorithms to solve the factorization problem so far. Shor's idea leads to a polynomial time algorithm in $n := \log N$, which is the appropriate measure of the size of a number. The speedup thanks to Shor's quantum algorithm over the best known classical algorithm is:

$$\mathcal{O}\left(e^{(C+o(1))n^{\frac{1}{3}}(\log n)^{\frac{2}{3}}}\right) \xrightarrow{\text{Shor}} \mathcal{O}(n^3).$$

In 2012 the record for the largest number factored using Shor's algorithm on a quantum computer so far was $21 = 3 \cdot 7$ (see [23]).

The Discrete Logarithm Problem

Let g be a primitive element of $(\mathbb{Z}_p^*, \cdot)$ and $x \in \mathbb{Z}_p^*$ where p is a prime number. Again, the discrete logarithm problem (DLP) is the task to compute r such that $g^r = x \mod p$.
To apply Shor's idea to the DLP, it is sufficient to make small modifications. The sought-after subgroup $H \leq G = (\mathbb{Z}_p^*, \cdot) \times (\mathbb{Z}_p^*, \cdot)$ can be described as the kernel of the function $f(a, b) := g^a x^{-b} \mod p$. f is a homomorphism of G onto $(\mathbb{Z}_p^*, \cdot)$ whose kernel contains multiples of the pair $(r, 1)$. After obtaining the kernel of the map, we note that the element $(r, 1)$ yields a solution r to the DLP since

$$f(r, 1) = g^r x^{-1} \equiv 1 \mod p \Leftrightarrow g^r \equiv x \mod p.$$

We saw that the two important number theoretic problems widely used in cryptosystems nowadays (namely in RSA and ElGamal) are broken by Shor's algorithm running on a quantum computer of proper size. Furthermore, the discrete logarithm problem on elliptic curves (ECDLP) is affected by this algorithm. To illustrate the impressive impact there, we remark that the best known classical attack against the DLP for elliptic curves gained an exponential speedup, where N denotes the number of points on the curve:

$$\mathcal{O}(\sqrt{N}) = \mathcal{O}\left(e^{\frac{\log N}{2}}\right) \xrightarrow{\text{Shor}} \mathcal{O}((\log N)^3).$$

4.4 Quantum Computing Milestones

Wikipedia [41] provides an overview of important discoveries and achievements connected to the quantum computer. We present a small collection of those "milestones" that are

of importance for this thesis. In this enumeration, NMR stands for the architecture of quantum computers based on "nuclear magnetic resonance" effects. The model of NMR quantum computers, where qubits are implemented as charged atomic particles trapped in electromagnetic fields and manipulated with laser beams, were used in early experiments. Unfortunately they do not seem to be a promising branch for future development, since there are problems to scale them from small to large qubit systems — a requirement of practical applications and one of DiVincenzo's criteria (see Section 4.2).

1980 The idea of quantum computing was proposed by Manin.

1984 The BB84 protocol was proposed by Bennett and Brassard for the distribution of cryptographic keys.

1993 Simon invented an oracle problem for which a quantum computer would be exponentially faster than a conventional computer.

1994 Peter Shor discovers an important algorithm. [...] It solved both the factorization problem and the discrete log problem. Its invention sparked a tremendous interest in quantum computers.

1996 Lov Grover invented the quantum database search algorithm. [...] The algorithm can be applied to a much wider variety of problems. Any problem that had to be solved by random, brute-force search, could now have a quadratic speedup.

1998 First experimental demonstration of a quantum algorithm; a 2-qubit NMR quantum computer was used to solve Deutsch's problem.

1998 First execution of Grover's algorithm (on an NMR computer).

2001 First execution of Shor's algorithm; the number 15 was factored.

2005 The first quantum byte (qubyte, a collection of 8 entangled qubits) has been created.

2008 A qubit was stored for over 1 second in an atomic nucleus.

2011 Scientifically confirmed 14 qubit register.

2011 The company D-Wave claims to have the first commercially available quantum computer called D-Wave One. They already *claimed* to have produced a 28-qubit quantum computer in 2007 and a 128 qubit computer chip in 2008, though this claims have yet to be verified by the scientific community.

2012 D-Wave claims to carry out quantum computations using 84 qubits.

2012 IBM announced that they are "on the cusp of building systems that will take computing to a whole new level".

2012 A two-qubit quantum computer that can easily be scaled up in size and functionality at room temperature has been presented.

For the last two milestones see `http://en.wikipedia.org/wiki/Quantum_computer`.

4.5 Post Quantum Cryptography

4.5.1 Impact of the Quantum Computer on Cryptosystems

We have seen that if a quantum computer of appropriate size would be available, some currently used cryptosystems would be shaken. Some ciphers like the asymmetric cryptosystems RSA, ElGamal would not be safe anymore, others like the symmetric AES are vulnerable to a generic search attack, but they can be fixed by adapting the key length to postpone the immediate threat. As will be discussed in Section 4.6 quantum mechanics also provides some solutions for the dilemma it has brought to the cryptographic community.
Nevertheless there are also classical cryptosystems that withstand the currently known opportunities the quantum computer provides. In his apocalyptic introduction to post-quantum cryptography Bernstein [10] asks "Is cryptography dead?" and thereafter provides four classes of public key cryptography systems that survive. Those cryptosystems are hash-based, code-based, lattice-based and multivariate-quadratic-equations cryptosystems.
There is no formal definition when a cryptographic system gets the attribute to be "post-quantum" secure. Instead, to qualify as a post-quantum system, it needs to be broadly believed to be classical secure as well as none of the currently known quantum algorithms should be applicable. Those four classes seem to fulfill these criteria in the sense that they have been heavily cryptanalyzed in the past and no way to break them with a quantum computer has been discovered so far.
In this thesis we focused on the cryptosystems by McEliece (see 2.2.1) and Niederreiter (see 2.2.2), both based on linear codes. In the following we discuss their biggest advantage.
The ideas that break the current state-of-the-art cryptosystems is due to two quantum algorithms, namely Grover's and Shor's. They are not applicable with the same impact on these code-based schemes as we address in Sections 4.5.2 and 4.5.3, therefore do not face immediate threat by quantum computers, unless completely new ideas emerge.
However, since a quantum computer of appropriate size does not seem to be there soon, also the ECRYPT2 report [27, 6.4] on the security of currently available ciphers states that their recommendations "assume (large) quantum computers do not become a reality in the near future".
Anyhow, Bernstein [10] further gives three important reasons why the cryptographic community already investigate post-quantum cryptography to such an extend. He stresses that it needs time to improve the efficiency, build confidence and improve the usability of post-quantum cryptosystems. He also urges to be aware of the distinctions between quantum and post-quantum cryptography and states "I see tremendous potential in post-quantum cryptography and very little hope for quantum cryptography.".

4.5.2 McEliece and Niederreiter PKS resist QFT

Dinh, Moore and Russell [11] used the quantum Fourier transform to mount "structural attacks" against code-based cryptosystems, which means these attacks try to reveal encryption details or even the secret key given the public key.

They showed that "the natural reduction of McEliece to a hidden sub-group problem yields negligible information about the secret key, thus rule out the direct analogue of the quantum attack that breaks, for example, RSA."
This result is limited to the application of the quantum Fourier transform and does not apply to other attacks — quantum or classical.
Since it seems most major speedups achieved by quantum computers are based on the QFT or similar ideas, this remains a remarkable result.
The paper gives detailed criteria how to choose the code parameters as well as restrictions for the matrices S and P in the McEliece cryptosystem (see Section 2.2.1), to be resistant against quantum Fourier transform.
They conclude that classical Goppa codes basically remain a secure choices of "McEliece-type cryptosystems against standard quantum Fourier sampling attacks", since another structural attack by Sidelnikov can only be mounted against a bigger class of codes as discussed in Section 2.2.2.
More specific, the authors successfully test their criteria on the secure code parameters $(n = 1632, k = 1269, t = 34)$ we mentioned in Section 2.2.1, and claim they are secure with respect to attacks based on the QFT.

4.5.3 Grover vs. McEliece

In the previous sections, we focused on structural attacks, however the best known attack against the McEliece cryptosystem with the $[n, k, d]-$code C so far is based on a random choice of k coordinates out of n and assuming they carry information. This means that the error vector added in the encryption process (see Algorithm 5) does not affect these k coordinates.
This idea leads to a technique called information-set decoding. In [4] Bernstein discusses how and if Grover's algorithm can speedup information-set decoding on a quantum computer. The basic information-set decoding algorithm presented there views the generator matrix G as a linear map from $\mathbb{F}_2^k \to \mathbb{F}_2^n$ and assumes a message x to be encrypted with McEliece's PKS. An information-set S of size k (dependent on a received message $y = x + e$) of the set $\{1, 2, \ldots, n\}$ is randomly chosen and the following two assumptions are made about S:

1. The natural projection $\pi : \mathbb{F}_2^n \to \mathbb{F}_2^S$ maps the error vector e (that was added to obfuscate y) to $0 \in \mathbb{F}_2^S$ — every chosen coordinate is error-free.

2. The composition $\mathbb{F}_2^k \to \mathbb{F}_2^n \to \mathbb{F}_2^S$ is an invertible $(k \times k)$ matrix.

Information-set decoding assumes the composition of message encryption and coordinate retrieval is an invertible $(k \times k)$ matrix and tries to construct the error vector thus enabling the computation of the message $x = y - e$. If one of these two assumptions is found to be untrue — either the $(k \times k)$ matrix is not invertible or the computed error vector does not have weight $w(e) = t$ — this method fails. Thus the information-set S was obviously not correct and another choice might be successful.
A closer analysis of the expected iterations before the first occurring of these events yields a running time of $\mathcal{O}(c^{(1+o(1))n/\log n})$ and is carried out in detail in the same paper.

Bernstein then introduces quantum information-set decoding, an approach to use Grover's quantum algorithm for information-set decoding.
Hence he defines a function f that does the algorithmic steps presented above and returns 0 if the computation could be finished successfully. In this formulation he reduced decoding to root finding of a function f; find x, such that $f(x) = 0$. A task that Grover's algorithm can carry out on a quantum computer!
He argues that this method yields a quadratic speedup compared to the classical information-set decoding algorithm with a complexity of $\mathcal{O}(c^{(1/2+o(1))n/\log n})$. Then he points out some enhancements and gives possible directions for further research.
The final conclusion of this paper is that quantum information-set decoding does not break McEliece's system or other code-based cryptosystems. Still, he recommends parameter adjustments in the presence of a powerful quantum computer.
In Section 4.3.3 we pointed out the generic speedup Grover's algorithm can achieve for any $\mathcal{NP}$ problem reformulated as a search problem. A precautious user that made worst-case assumption about the impact of Grover's quadratic speedup against code-based cryptosystems anyway, has of course already considered enlarging the key size to remain on the same security level.

4.6 Quantum Cryptography

Quantum mechanical effects can evolve to form a threat for the security of some widespread classical public key cryptosystems, on the other hand they also offer an entirely new form of cryptography.
The theory of establishing secure communications based on quantum mechanics is subsummarized as quantum cryptography and the advantages and limitations will be briefly addressed in the following to close the circle of topics dealt with in this thesis.

One-Time Pad and Provable Security

The discussion about provable security, we have started back in Section 2.1.1, now comes to one interesting aspect — the one-time pad, OTP for short.
None of the cryptosystems we presented so far could mathematically be proven secure, since they rely on assumptions. For example it is assumed that $\mathcal{NP}$-`complete` problems are indeed hard for computers to solve in the average case. Another assumption is that nowadays best algorithms and the computational power currently available are not sufficient to break these systems in practice. Although there are strong indications to believe this, these assumptions are based on the current (scientific) knowledge and can eventually change (see also [42, Ch. 13.11]).
Without going into detail we present the following interesting information theoretic facts. In 1949 Shannon [34] defined the term "perfect secrecy", where he also concludes that to achieve perfect secrecy for a message M one has to use a key K of (at least) equal length for the encryption process. Later he proved that the system known as one-time pad (OTP), where each bit of the binary message M is XORed (bitwise addition $\mod 2$) with the truly random key bits of K, is perfectly secure in this information theoretic sense.

In practice there remain some serious issues that are a threat to this encryption system, like the generation of random bits and a flawless implementation of the cryptosystem.
It is quite astonishing though that, if OTP is properly implemented and the key exchange is done via quantum key distribution (QKD) — for example via the BB84 protocol proposed by Bennett and Brassard, this system provides perfect secrecy. Since QKD does not need a functional quantum computer but rather some already available technology this seems promising. The BB84 protocol mentioned above produces a truly random cryptographic key that can be used for an instance of OTP to provide perfect secrecy — in the mathematically sense. A hybrid approach that is probably less secure, but does not require key lengths comparable to plain text lengths, is to use the safely distributed key material as input for, say, AES. The big advantage of this kind of key generation is that security of QKD rests upon quantum physical laws.
As mentioned in Section 2.5, in the end it is — as so often in the physical world — a trade-off between practicality and security to deploy cryptosystems. In [42, Ch. 13.9.2] the author states that although quantum cryptography is "suited for securing information that must be kept confidential indefinitely" it is "likely overkill for the majority of messages, like email, whose value, even if intercepted, is transient". Still quantum cryptography, like OTP with QKD for example, is secure "regardless of the mathematical sophistication, computational power, or algorithmic prowess of any eavesdropper" if carefully implemented.

Final Statement

To provide another reason, why a system that is perfectly secure in some sense may not be used much in practice and why such a system will not make the whole cryptographic research so far dispensable, we let Bruce Schneier (`http://www.schneier.com/crypto-gram-0210.html#7`) have the last words in this thesis: "If you think you know how to do key management, but you don't have much confidence in your ability to design good ciphers, a one-time pad might make sense. We're in precisely the opposite situation, however: we have a hard time getting the key management right, [...] but we're pretty confident in our ability to build reasonably strong algorithms. It's just not the weak point in our systems.".

List of Tables

List of Algorithms

Bibliography

[1] Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and Weaknesses of Quantum Computing. *SIAM J. Comput.*, 26(5):1510–1523, 1997. http://dx.doi.org/10.1137/S0097539796300933, website accessed 2013-03-14.

[2] E. R. Berlekamp. Goppa Codes. *IEEE Transactions on Information Theory*, pages 19:590–592, 1973.

[3] E. R. Berlekamp, R. J. McEliece, and Tilborg van H. C. A. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24, 1978.

[4] Daniel J. Bernstein. Grover vs. McEliece. In *Proceedings of the Third international conference on Post-Quantum Cryptography*, PQCrypto'10, pages 73–80, Berlin, Heidelberg, 2010. Springer-Verlag. http://dx.doi.org/10.1007/978-3-642-12929-2_6, website accessed 2013-03-14.

[5] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Attacking and defending the McEliece cryptosystem. Cryptology ePrint Archive, Report 2008/318, 2008. http://eprint.iacr.org/, website accessed 2012-12-21.

[6] Thomas A. Berson. Failure of the McEliece public-key cryptosystem under message-resend and related-message attack. In *Advances in Cryptology - CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 213–220. Springer Berlin Heidelberg, 1997. http://dx.doi.org/10.1007/BFb0052237, website accessed 2012-12-21.

[7] Bhaskar Biswas. Implementational aspects of code-based cryptography, 2010.

[8] Jehoshua Bruck and Moni Naor. The hardness of decoding linear codes with preprocessing. *IEEE Transactions on Information Theory*, 36:381–385, 1990.

[9] R. Overbeck D. Engelbert and A. Schmidt. A Summary of McEliece-Type Cryptosystems and their Security. Cryptology ePrint Archive, Report 2006/162, 2006. http://eprint.iacr.org/2006/162.ps, website accessed 2013-03-14.

[10] Erik Dahmen Daniel J. Bernstein, Johannes Buchmann, editor. *Post-Quantum Cryptography*. Springer, 2009. http://pqcrypto.org/, website accessed 2013-03-14.

[11] Hang Dinh, Cristopher Moore, and Alexander Russell. McEliece and Niederreiter cryptosystems that resist quantum fourier sampling attacks. In *Proceedings of the 31st annual conference on Advances in cryptology*, CRYPTO'11, pages 761–779, Berlin, Heidelberg, 2011. Springer-Verlag. http://dl.acm.org/citation.cfm?id=2033036.2033093, website accessed 2013-03-14.

[12] David P. Divincenzo. The physical implementation of quantum computation. *Fortschr. Phys*, 48, 2000.

[13] Gerhard Dorfer. Lecture notes on error correcting codes. http://dmg.tuwien.ac.at/dorfer/codes/index.html, website accessed 2012-12-21.

[14] Nobel Committee for Physics. Particle control in a quantum world. http://kva.se/Documents/Priser/Nobel/2012/fysik/pop_fy_en_12.pdf, website accessed 2013-03-14.

[15] Damien Giry. BlueKrypt — Cryptographic Key Length Recommendation, 2012. http://www.keylength.com, website accessed 2012-12-21.

[16] Tom Høholdt and Ruud Pellikaan. On the decoding of algebraic-geometric codes. *IEEE Trans. Inform. Theory*, 41:1589–1614, 1995.

[17] Heeralal Janwa and Oscar Moreno. McEliece Public Key Cryptosystems Using Algebraic-Geometric Codes. *Des. Codes Cryptography*, 8(3):293–307, 1996. http://dx.doi.org/10.1023/A:1027351723034.

[18] Kazukuni Kobara and Hideki Imai. Semantically Secure McEliece Public-Key Cryptosystems - Conversions for McEliece PKC. Springer Verlag, 2001.

[19] Neal Koblitz and Alfred J. Menezes. Another Look at "Provable Security". Technical report, 2004.

[20] R. Lidl and H. Niederreiter. *Introduction to Finite Fields and Their Applications*. Cambridge University Press, 1994.

[21] S. Loth, S. Baumann, C.P. Lutz, D.M. Eigler, and A.J. Heinrich. Bistability in atomic-scale antiferromagnets. *Science*, 335(6065):196–9, 2012. http://dx.doi.org/10.1126/science.1214131, website accessed 2013-03-14.

[22] F. J. MacWilliams and N. J. A. Sloane. *The theory of error correcting codes*. North-Holland mathematical library. North-Holland Pub. Co., 1978. http://books.google.at/books?id=LuomAQAAIAAJ, website accessed 2012-12-21.

[23] Enrique Martín-López, Anthony Laing, Thomas Lawson, Roberto Alvarez, Xiao-Qi Zhou, and Jeremy L. O'Brien. Experimental realization of Shor's quantum factoring algorithm using qubit recycling. *Nature Photonics*, 6(11):773–776, 2012. http://dx.doi.org/10.1038/nphoton.2012.259, website accessed 2013-03-14.

[24] R. J. McEliece. A Public-Key Cryptosystem Based On Algebraic Coding Theory. *DSN Progress Report 42-44: 114*, 1978. `http://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF`, website accessed 2012-12-21.

[25] Harald Niederreiter and Chaoping Xing. *Algebraic Geometry in Coding Theory and Cryptography*. Princeton University Press, Princeton, NJ, USA, 2009.

[26] M.A. Nielsen and I.L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. `http://books.google.de/books?id=-s4DEy7o-a0C`, website accessed 2013-03-14.

[27] European Network of Excellence in Cryptology II. ECRYPT II Yearly Report on Algorithms and Key Lengths, 2012. `http://www.ecrypt.eu.org/documents/D.SPA.20.pdf`, website accessed 2012-12-21.

[28] Raphael Overbeck and Nicolas Sendrier. Code-based cryptography. In Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen, editors, *Post-Quantum Cryptography*, chapter 4, pages 95–145. Springer Berlin Heidelberg, 2009. `http://dx.doi.org/10.1007/978-3-540-88702-7_4`, website accessed 2013-03-14.

[29] Rafael Misoczki Paulo S. L. M. Barreto, Richard Lindner. Decoding square-free Goppa codes over $\mathbb{F}_p$. *CoRR*, abs/1103.3296, 2011. `http://arxiv.org/abs/1103.3296`, website accessed 2013-05-01.

[30] X.-W. Wu R. Pellikaan and S. Bulygin. Error-correcting codes and cryptology. `http://www.win.tue.nl/~ruudp/courses/2WC11/2WC11-book.pdf`, website accessed 2013-02-28; Book in preparation to be published, preliminary version (23 January 2012).

[31] Eleanor Rieffel and Wolfgang Polak. An introduction to quantum computing for non-physicists. *ACM Comput. Surv.*, 32(3):300–335, 2000. `http://doi.acm.org/10.1145/367701.367709`, website accessed 2013-03-14.

[32] Thomas Risse. How SAGE helps to implement Goppa Codes and McEliece PKCSs, 2012. `http://www.weblearn.hs-bremen.de/risse/papers/UbiCC11/SAGE_Goppa_McEliece.pdf`, website accessed 2013-03-14.

[33] Dorit Ron and Adi Shamir. Quantitative Analysis of the Full Bitcoin Transaction Graph. Cryptology ePrint Archive, Report 2012/584, 2012. `http://eprint.iacr.org/2012/584`, website accessed 2013-03-14.

[34] C. E. Shannon. Communication Theory of Secrecy Systems. *Bell System Technical Journal*, 28, 1949.

[35] Peter W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.*, 26(5):1484–1509, October 1997. `http://dx.doi.org/10.1137/S0097539795293172`, website accessed 2013-03-14.

[36] V. M. Sidelnikov and S. O. Shestakov. On insecurity of cryptosystems based on generalized Reed-Solomon codes. *Discrete Mathematics and Applications*, 2:439–444, 1992.

[37] Christian Wieschebrink and Godesberger Allee. *Cryptanalysis of the Niederreiter Public Key Scheme Based on GRS Subcodes*. 2010.

[38] Wikipedia. Computational hardness assumption, 2012. `http://en.wikipedia.org/wiki/Computational_hardness_assumption`, website accessed 2012-12-21.

[39] Wikipedia. Deutsch-Jozsa-Algorithmus, 2013. `http://de.wikipedia.org/wiki/Deutsch-Jozsa-Algorithmus#Der_Quantenalgorithmus`, website accessed 2013-03-14.

[40] Wikipedia. Theoretical implications of one-way functions, 2013. `http://en.wikipedia.org/wiki/One-way_function#Theoretical_implications_of_one-way_functions`, website accessed 2013-03-14.

[41] Wikipedia. Timeline Of Quantum Computing, 2013. `http://en.wikipedia.org/wiki/Timeline_of_quantum_computing`, website accessed 2013-03-14.

[42] Colin P. Williams. *Explorations in Quantum Computing*. Springer, 2nd edition, 2011.

[43] T. Ylonen. The Secure Shell (SSH) Authentication Protocol, 2006. `http://tools.ietf.org/html/rfc4252#section-7`, website accessed 2012-12-21.