Some Notes on Code-Based Cryptography

Löndahl, Carl

2015

Some Notes on Code-Based Cryptography

Carl Löndahl

LUND UNIVERSITY

Doctoral Dissertation
Information Theory
Lund, December 2014

Carl Löndahl
Department of Electrical and Information Technology
Information Theory
Lund University
P.O. Box 118, 221 00 Lund, Sweden

Series of licentiate and doctoral dissertations
ISSN 1654-790X; No. 67
ISBN 978-91-7623-119-7

Typeset in Palatino and Helvetica using LaTeX 2_ε.
Printed in Sweden by Tryckeriet i E-huset, Lund University, Lund.

Abstract

THIS thesis presents new cryptanalytic results in several areas of coding-based cryptography. In addition, we also investigate the possibility of using convolutional codes in code-based public-key cryptography.

The first algorithm that we present is an information-set decoding algorithm, aiming towards the problem of decoding random linear codes. We apply the generalized birthday technique to information-set decoding, improving the computational complexity over previous approaches.

Next, we present a new version of the McEliece public-key cryptosystem based on convolutional codes. The original construction uses Goppa codes, which is an algebraic code family admitting a well-defined code structure. In the two constructions proposed, large parts of randomly generated parity checks are used. By increasing the entropy of the generator matrix, this presumably makes structured attacks more difficult.

Following this, we analyze a McEliece variant based on quasi-cylic MDPC codes. We show that when the underlying code construction has an even dimension, the system is susceptible to, what we call, a squaring attack. Our results show that the new squaring attack allows for great complexity improvements over previous attacks on this particular McEliece construction.

Then, we introduce two new techniques for finding low-weight polynomial multiples. Firstly, we propose a general technique based on a reduction to the minimum-distance problem in coding, which increases the multiplicity of the low-weight codeword by extending the code. We use this algorithm to break some of the instances used by the TCHo cryptosystem. Secondly, we propose an algorithm for finding weight-4 polynomials. By using the generalized birthday technique in conjunction with increasing the multiplicity of the low-weight polynomial multiple, we obtain a much better complexity

than previously known algorithms.

Lastly, two new algorithms for the learning parities with noise (LPN) problem are proposed. The first one is a general algorithm, applicable to any instance of LPN. The algorithm performs favorably compared to previously known algorithms, breaking the 80-bit security of the widely used $(512, \frac{1}{8})$ instance. The second one focuses on LPN instances over a polynomial ring, when the generator polynomial is reducible. Using the algorithm, we break an 80-bit security instance of the Lapin cryptosystem.

Contents

Preface

This thesis contains results from research performed by the author at the Department of Electrical and Information Technology at Lund University. Parts of the material have been presented at international conferences. Publications and reports have appeared as follows.

- T. Johansson, C. Löndahl, »An Improvement to Stern's Algorithm,« *Internal report*, September 2011.
- C. Löndahl, T. Johansson, »A new version of McEliece PKC based on convolutional codes,« in T. W. Chim and T. H. Yuen, editors, *Information and Communications Security*, volume 7618 of *Lecture Notes in Computer Science*, pp 461–470. Springer-Verlag, 2012.
- T. Johansson, C. Löndahl, »A new algorithm for finding low-weight polynomial multiples and its application to TCHo,« in L. Budaghyan, T. Helleseth, M. G. Parker, editors, *Workshop on Coding and Cryptography*. Preproceedings, 2013.
- C. Löndahl, T. Johansson, »Improved Algorithms for Finding Low-Weight Polynomial Multiples in $\mathbb{F}_2[x]$ and Some Cryptographic Applications,« in *Design, Codes and Cryptography*, volume 73 issue 2, pp 625–640. Kluwer Academic Publishers, 2013.
- C. Löndahl, T. Johansson, M. Koochak Shooshtari, M. Ahmadian-Attari, M. Reza Aref, »Squaring Attacks on McEliece Public-Key Cryptosystems Using Quasi-Cyclic Codes of Even Dimension,« submitted to *Design, Codes and Cryptography*.
- Q. Guo, T. Johansson, C. Löndahl, »Solving LPN using Covering Codes[†],« in T. W. Chim and T. H. Yuen, editors, *Asiacrypt 2014, Taiwan, China,* volume 8873 of *Lecture Notes in Computer Science*, pages 1–20. Springer-Verlag, 2014.

- Q. Guo, T. Johansson, C. Löndahl, »A New Algorithm for Solving Ring-LPN with a Reducible Polynomial,«, submitted to *IEEE Transactions on Information Theory*.

The paper marked with † was selected for the best-paper award. During my time as a Ph.D. student, I have also co-authored the following publications, which are not included in this dissertation.

- M. Ågren, C. Löndahl, M. Hell, T. Johansson, »A Survey on Fast Correlation Attacks,« in C. Carlet, editor, *Cryptography and Communications,*, volume 4 issue 3-4, pp 173–202. Springer-Verlag, 2012.
- M. Ågren, M. Hell, T. Johansson, C. Löndahl, »Improved message passing techniques in fast correlation attacks on stream ciphers,« in *Proceedings of 7th International Symposium on Turbo Codes & Iterative Information Processing*, pp. 260-264, 2012.

The research work included in this dissertation is supported in part by the Swedish Research Council through Grant No. 621-202-4259.

Acknowledgments

The completion of this thesis has been a long journey, with support from many contributors. First and foremost, my deepest gratitude goes to my supervisor Thomas Johansson. Without your profound knowledge and countless sharp ideas, this work would not have been possible. You have constantly encouraged me and lead my own ideas and sometimes scattered thoughts in the right direction.

My colleagues and friends at the department, and in particular the members of the crypto group, have contributed greatly to my personal and professional time at Lund University. Especially, I want to thank Martin Hell, Paul Stankovski and Martin Ågren for their support in so many ways; always happy to answer more or less clever questions.

I want to thank my co-author Qian Guo. It has been a pleasure working with you during the final year of my Ph.D. studies. I also want to thank room mate Christopher Jämthagen for many interesting discussions, both in the office and on the badminton court. I wish to thank all who contributed by proofreading my thesis. My thanks also go to Fredrik Rusek, with whom I had many enlightening discussions. I also want to thank the people in the fika and Friday-lunch group for many laughs and the occasional sugar rushes.

Last, but not least, I want to express my sincere gratitude to my family – my mom Kerstin, my dad Kenneth and my sister Malin, for your patience and for greatly supporting me during my entire life. Most importantly, my deepest thanks goes to my love Stina and the little person soon to see the light of day.

Carl
Lund, December 2014

1

Introduction

> Just as the technology of printing altered and reduced the power of medieval guilds and the social power structure, so too will cryptologic methods fundamentally alter the nature of corporations and of government interference in economic transactions.
> – *Timothy C. May, Crypto Anarchist Manifesto*

Some secrets need to be kept. Ever since the cradle of civilization and the invention of writing, mankind has tried to hide information from prying eyes. Historical evidence indicates that cryptographic schemes were used by many early civilizations[1], such as the Egyptians, Hebrews and Greeks. During the Medieval era, quite sophisticated ciphers were discovered, but it would take until the First and Second World Wars for ciphers to escalate beyond ad-hoc approaches.

Cryptology is the mathematical science of ciphers, and incidentally, its etymology is the greek *kryptós logia* which means the »study of secrets«. Cryptology is an umbrella term for its two subfields *cryptography* and *cryptanalysis*. These subfields relate to each other much like a cat-and-mouse game; while cryptography is the art of constructing algorithms and protocols, cryptanalysis takes the adversarial viewpoint of cryptology in breaking or analyzing the security of the cryptographic constructions.

Prior to the dawn of the digital revolution, cryptography was basically used exclusively to ensure secrecy when communicating secret information between two parties over a non-secure channel, such as a radio link or a phone line. But during the modern era of computers and its vast flows of

[1] Probably most well-known, but maybe not most significant among the ancient ciphers, is the Roman cipher commonly called the »Caesar cipher«.

information, cryptography evolved rapidly and became something amazing. On the one hand, it became a necessity for data security, authentication, digital signatures etc. – something electronic commerce, banking and credit cards could not exist without. On the other hand, cryptography has become a tool in politics. Anonymization by strong cryptography allows journalists and whistleblowers to keep their integrity. It also has given birth to movements such as crypto anarchism and crypto capitalism, that employ cryptography to protect privacy and political freedom.

One of the pillars of cryptography is unarguably *Kerckhoffs's principle*, adopted by most cryptographers. Kerckhoffs's principle essentially states that the security of a cryptographic construction must depend only on the secret key. Any remaining part of the construction should be assumed to be known to a potential adversary.

Roughly speaking, the aim of a cryptographic encryption scheme is to render a *message* unintelligible in some way such that the resulting *ciphertext* reveals little or no information about the content to an adversary, while still enabling the receiver to reverse the procedure in order to obtain the original message. These two procedures are called *encryption* and *decryption*.

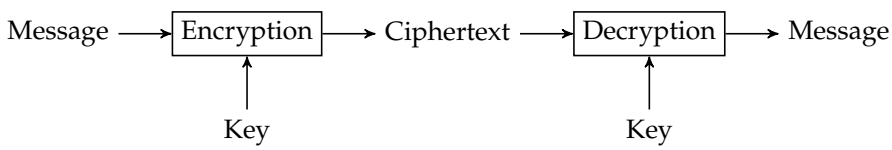


Figure 1.1: A typical model of encryption and decryption.

Encryption schemes (and cryptography as a whole) can be divided into *symmetric* and *asymmetric* (or *public-key*). Cryptography should not be confused with steganography, which is a term for hiding information rather than obfuscating it.

Symmetric encryption allows for two parties to communicate securely, but they need to agree beforehand upon a common secret key. The same key is used for encryption and decryption. The first modern asymmetric cryptographic scheme was introduced in the groundbreaking work of Diffie and Hellman [DH76], in which the authors vaticinated the needs of a world-wide spanning network which today is the Internet, a place where communication between strangers occurs constantly. This raised a rather severe problem with symmetric cryptography – how to securely exchange keys. The exchange has historically been done by a trusted courier. In a network interconnecting a whole planet and beyond, this is not a practical solution.

Public-key encryption is advantageous over its symmetric counterpart in

the sense that it allows two separate and independent parties to securely communicate without any shared information. One might argue that to decrypt a message, you need a key; to encrypt a message, you also need that key; if not, the ciphertext is independent of the key and thus, anyone can decrypt. This argument has an implicit and false assumption – the keys need not to be the same. Indeed, this is reason for the choice of the term asymmetric. The authors of [DH76] suggested to base the cryptosystem upon a mathematical procedure, in which knowing the procedure does not imply knowing how to reverse the procedure (find the inverse). The paper by Diffie and Hellman suggested a cryptosystem, the *DH cryptosystem*, based on a mathematical problem called the *discrete logarithm problem*, which initiated a hunt for problems of similar nature. Shortly after, several different problems were suggested, thereamong the *knapsack problem*.

The knapsack problem is easy to describe, without involving in-depth mathematics. Think of having collection of various items, where the weight of each item is known. Suppose that you are told it is possible to put some of the items in the knapsack such that the knapsack has a certain weight. The problem consists of finding which items' weights together add up to that certain weight. Assume that the collection of items can be used to describe a message, and that the collection is known to everyone. Given a message, everyone can find the weight. However, given the weight, it is hard to find the message. To conclude, it is easy perform the mathematical procedure, but inverting the procedure is hard. This is what we call a *one-way function*.

One-way functions can be used to construct public-key cryptosystems, if they can be equipped with a *trapdoor*. Knowing the trapdoor, the mathematical procedure is easy to invert. A trapdoor could be a pattern which can be exploited to find the solution without trying all possibilities. Without going into the details, Merkle-Hellman cryptosystem [MH78] is a public-key cryptosystem constructed from the knapsack problem, but was broken and is therefore not used today. At the same time, Rivest, Shamir and Adleman created the RSA cryptosystem [RSA78], which still remains unbroken and is one of the most used public-key cryptosystems today.

The RSA cryptosystem is based on multiplication and factoring; multiplying numbers is easy – even large ones, but factoring a number is very hard. This is the short story of RSA. In the middle of the 90's, Shor [Sho94] formulated an algorithm which has a remarkable trait; it can factor integers very efficiently. There is only one drawback: to run it, one needs a *quantum computer*[2]. If the quantum revolution in computing had not yet begun – then Shor's paper certainly ignited the spark.

[2]The algorithm can be simulated on a classical computer, but then with no advantage over classical factoring algorithms.

Quantum computers, introduced by Manin [Man80] and Feynman [Fey82] in the early 80's, exploit quantum-mechanics, which, roughly speaking, allow the quantum computers to perform simultaneous computations on a massive scale. With a quantum computer, cryptosystems based on problems from number theory, including discrete logarithm, are rendered insecure. The series of overwhelming discoveries lead researchers into a new direction: *post-quantum cryptography* – the pursue to create cryptography resistant to quantum attacks. In fact, a cryptosystem from the same era as the Diffie-Hellman and RSA cryposystems was already known to be post-quantum secure (or rather, »not post-quantum insecure«). The *McEliece cryptosystem*, introduced in [McE78] by McEliece had, mostly because of its comparably inefficient key representation, long been left in the dark, but now gained interest in the research community. In contrast to RSA and DH, McEliece is based on a fundamentally different problem, which has not yet been proven to be solvable by a quantum computer.

The McEliece cryptosystem has given birth to a field of its own, called *code-based cryptography*. This field is the focus of attention for this thesis.

1.1 OUTLINE

The main idea of the subsequent sections of this chapter is to give a broad picture of the mathematics and different concepts used in this thesis. It does not contain the full amount of details wanted from a textbook, so we will attempt to provide the reader with sufficient references for further reading whenever possible.

We begin with the global notation, then we proceed with some brief notes on probability theory and combinatorics. This is followed by information-theory basics and hypothesis testing. Then, we discuss algorithms and complexity theory and finally conclude with the birthday paradox and its algorithmic implications.

In Chapter 2, we introduce the fundamentals of coding theory and linear codes. Then, we discuss different types of decoding, theoretical results in coding and the computational problems that arise when decoding information in the presence of noise. The remainder of the chapter presents some typical families of codes used in this thesis.

Chapter 3 contains a survey of different information-set decoding algorithms, used to solve problems in decoding. Then, we present one of the contributions of this thesis, which is a new information-set decoding algorithm. We show that the algorithm is more efficient than the state-of-the-art algorithms at the time.

In Chapter 4, we introduce code-based cryptography and the McEliece

cryptosystem. We also propose a new variant of McEliece based on convolutional codes. In response to an attack on the proposed cryptosystem, we give new updated parameters for an 80-bit security level.

Chapter 5 introduces a new cryptanalytic technique called a squaring attack. We use it to attack a fairly new variant of McEliece based on even-dimension QC-MDPC codes. The new technique significantly improves attack complexity over other known approaches.

Chapter 6 provides two new algorithms for finding low-weight polynomial multiples. The first is a very general algorithm, applicable to any target weight. The second focuses on weight-4 polynomial multiples, having applications in cryptanalysis such as correlation attacks on stream ciphers.

Chapters 7 and 8 are dedicated to the LPN-problem. In the first chapter, we give a new state-of-the-art algorithm based on covering codes. In the last chapter, we attack the Ring-LPN problem instances used in Lapin. Although it is applied on a specific cryptosystem, it has theoretical ramifications – in particular, we give new design criteria for Ring-LPN instances with a reducible polynomial.

1.2 NOTATION

Throughout this thesis, we will use the notation $\mathbb{F}_q$ for a finite field with q elements, $\mathbb{R}$ for the set of real numbers, $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$ for the set of integers and $\mathbb{N} = \{0, 1, 2, \dots\}$ for the natural numbers. Unless stated otherwise, all vectors are row vectors and given in bold face letters, i.e., $\mathbf{x} \in \mathbb{F}_q^{1\times k}$ for some integer $k > 0$. Indices start from 1 (unless there is a reason not to), i.e.,

$$\mathbf{x} = \begin{pmatrix} x_1 & x_2 & \cdots & x_k \end{pmatrix}.$$

For simplicity, we write $\mathbf{x} \in \mathbb{F}_q^k$. Sequences over a finite field are treated as vectors in a vector space, i.e.,

$$(x_i)_{i\in\{1,2,\dots,k\}} = \mathbf{x} \in \mathbb{F}_2^k.$$

if $x_i \in \mathbb{F}_2$ for $i \in \{1, 2, \dots, k\}$. The canonical inner product (or scalar product) between two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{F}_q^k$ is denoted

$$\langle \mathbf{x}, \mathbf{y} \rangle \in \mathbb{F}_q.$$

Matrices $\mathbf{A} \in \mathbb{F}_q^{k\times n}$ are given in the capital bold face letters. The transpose of $\mathbf{x} \in \mathbb{F}_q^{1\times k}$ is $\mathbf{x}^{\mathrm{T}} \in \mathbb{F}_q^{k\times 1}$, and equivalently for $\mathbf{A}$ the transpose is $\mathbf{A}^{\mathrm{T}} \in \mathbb{F}_q^{n\times k}$. Horizontal concatenation of two vectors $\mathbf{x} \in \mathbb{F}_q^k$ and $\mathbf{y} \in \mathbb{F}_q^l$ will be denoted

$$\mathbf{x}\|\mathbf{y} \in \mathbb{F}_q^{k+l},$$

while for two matrices $\mathbf{A} \in \mathbb{F}_q^{k \times n}$ and $\mathbf{B} \in \mathbb{F}_q^{l \times n}$ we use the notation

$$\begin{pmatrix} \mathbf{A} & \mathbf{B} \end{pmatrix} \in \mathbb{F}_q^{(k+l) \times n}.$$

A matrix $\mathbf{A} \in \mathbb{F}_q^{k \times n}$ is a row vector where each entry is a column vector, i.e.,

$$\mathbf{A} = \begin{pmatrix} \mathbf{a}_1^{\mathrm{T}} & \mathbf{a}_2^{\mathrm{T}} & \cdots & \mathbf{a}_n^{\mathrm{T}} \end{pmatrix}.$$

When referring to the ith column vector of a matrix $\mathbf{A}$, we write $\mathbf{a}_i$. Equivalently, if we refer to the ith row vector, we write $\mathsf{Row}\,(\mathbf{A}, i)$.

We denote linear subspaces as $\mathcal{U} \subset \mathbb{F}_q^n$. The dual (orthogonal subspace) of $\mathcal{U}$ is denoted

$$\mathcal{U}^{\perp} = \{\mathbf{x} \in \mathbb{F}_q^n : \langle \mathbf{x}, \mathbf{y} \rangle = 0,\ y \in \mathcal{U}\}.$$

For a random variable X having a distribution D, we write $X \sim \mathsf{D}$. If x is a variable taking a value according to a distribution D, we write

$$x \xleftarrow{\$} \mathsf{D}.$$

If $\mathcal{S}$ is a set, then the same notation as above

$$x \xleftarrow{\$} \mathcal{S}$$

means that the value of x is drawn uniformly at random from $\mathcal{S}$.

Note: We will abuse the functional names f and g extensively, so we advise the reader to think of it as a function with characteristics connected to the current context.

ASYMPTOTIC NOTATION

Frequently, we will use asymptotic notation. Below is a list of the ones occurring in this thesis;

$$\begin{aligned}
&f(n) \in \mathcal{O}\,(g(n)) && |f(n)| \leqslant k \cdot g(n) \text{ for some positive } k,\\
&f(n) \in \Omega\,(g(n)) && |f(n)| \geqslant k \cdot g(n) \text{ for some positive } k,\\
&f(n) \in \Theta\,(g(n)) && k \cdot g(n) \leqslant |f(n)| \leqslant l \cdot g(n) \text{ for some positive } k, l,\\
&f(n) \in o\,(g(n)) && |f(n)| \leqslant k \cdot g(n) \text{ for every positive } k,\\
&f(n) \sim g(n) && f(n)/g(n) \to 1 \text{ when } n \to \infty.
\end{aligned}$$

1.3 PROBABILITY THEORY AND COMBINATORICS

Randomness is a fundamental characteristic of the physical world and is a necessity for virtually all cryptography. Random numbers are a natural component in generation of cryptographic keys, and often also play a key role in the cryptographic algorithm as a whole. Alas, randomness is incredibly difficult to embody mathematically, and generation of truly random numbers must rely on unpredictable processes in the physical world. In many theoretical models in cryptography, constructions are given access to true randomness (random oracles). In practice, such assumptions about randomness are inherently false – many cryptographic schemes have been broken due to lack of randomness[3].

This section introduces some basic concepts on probability theory and related facts that constitute useful tools in the area of cryptographic construction and cryptanalysis. Although we desire to be as thorough as possible, large parts of the theory will be only be implicitly defined. For more details, see for instance [ASE91].

Given a set of *symbols* $\mathcal{X}$, called the *alphabet*, a (discrete) *random variable* X is defined from its probability mass function $\mathbb{P}(X = x)$. We say that the probability mass function $\mathbb{P}(X = x)$ determines the *probability distribution* of X. Moreover, we say that two random variables X and Y are *pair-wise independent* if and only if

$$\mathbb{P}(X = x, Y = y) = \mathbb{P}(X = x) \cdot \mathbb{P}(Y = y) \tag{1.1}$$

for all values of x and y. The *expectation* of a variable X with alphabet $\mathcal{X}$ is

$$\mathbb{E}(X) = \sum_{x \in \mathcal{X}} x \cdot \mathbb{P}(X = x), \tag{1.2}$$

provided that this sum converges absolutely. The *variance* of X is defined as

$$\mathrm{Var}(X) = \mathbb{E}\left(X^2\right) - \mathbb{E}(X)^2. \tag{1.3}$$

The expectation operator $\mathbb{E}(\cdot)$ is linear in the sense that for any two random variables X and Y and constants $\alpha, \beta, \gamma \in \mathbb{R}$, it holds that

$$\mathbb{E}(\alpha \cdot X + \beta \cdot Y + \gamma) = \alpha \cdot \mathbb{E}(X) + \beta \cdot \mathbb{E}(Y) + \gamma. \tag{1.4}$$

Regarding the expectation operator, we will succumb to some abuse of notation. If $\mathcal{S}$ is a random set or $\mathcal{L}$ is a random list, we consider its cardinality

[3] Or as Donald Knuth [Knu98] phrases it: *random numbers should not be generated with a method chosen at random.*

or length a random variable. Expected cardinality and expected length is denoted $\mathbb{E}(|\mathcal{S}|)$ and $\mathbb{E}(|\mathcal{L}|)$, respectively.

Returning to the distribution of a random variable X, a special and very simple kind of distribution called the *Bernoulli distribution* will be of great interest to us.

Definition 1.1 (Bernoulli distribution) Let $\rho \in [0,1]$ and X be a binary discrete random variable with probability mass function

$$\mathbb{P}(X = x) = \begin{cases} 1-\rho & \text{if } x = 0, \\ \rho & \text{if } x = 1, \end{cases} \tag{1.5}$$

then we say that X follows the Bernoulli distribution Ber_ρ.

Commonly, Bernoulli trials are used when modeling a toss of a coin, which typically has only two outcomes: heads or tails. For an unbiased (or symmetric) coin we have $\rho = \frac{1}{2}$. If the coin is asymmetric and $\rho = \frac{1}{2}(1 \pm \epsilon)$ for some real-valued number $\epsilon \in [0,1]$, we say that it has *bias* ϵ (or is ϵ*-biased*).

Many of the results presented in this thesis rely on probabilistic arguments. The following bounds are very common in probability analysis. *Markov's inequality* is a useful bound that holds for any probability distribution.

Theorem 1.1 (Markov's inequality) For any random variable X and $a > 0$ it holds that

$$\mathbb{P}(|X| \geqslant a) \leqslant \frac{\mathbb{E}(X)}{a}. \tag{1.6}$$

Another bound, *Chebyshev's inequality* bounds the probability that a random variables deviates from its mean.

Theorem 1.2 (Chebyshev's inequality) For any random variable X and $a > 0$ it holds that

$$\mathbb{P}(|X - \mathbb{E}(X)| \geqslant a) \leqslant \frac{\mathrm{Var}(X)}{a^2}. \tag{1.7}$$

Another incredibly useful result from the mathematical folklore is the *piling-up lemma*. We state the lemma as follows.

Lemma 1.3 (Piling-up lemma) Let $X_1, X_2, ..., X_n$ be a set of n independent binary random variables, where $\mathbb{P}(X_i = 0) = \frac{1}{2} \cdot (1 + \epsilon_i)$ for $0 < i \leqslant n$ and $\epsilon_i \in [0,1]$. Then,

$$\mathbb{P}(X_1 + X_2 + \cdots + X_n = 0) = \frac{1}{2} \cdot \left(1 + \prod_{i=1}^{n} \epsilon_i\right). \tag{1.8}$$

In the later parts of this thesis, we will draw binary matrices at random from a uniform distribution. A beautiful and well-known result, somewhat surprisingly, states that a square matrix over a finite field $\mathbb{F}_q$ with randomly drawn entries has a non-zero determinant with constant probability.

Lemma 1.4 (Random-matrix determinant) For any $k \in \mathbb{Z}^+$, it holds that

$$\mathbb{P}\left(\det \mathbf{A} \neq 0 \mid \mathbf{A} \xleftarrow{\$} \mathbb{F}_q^{k\times k}\right) \geqslant \exp\left(-\frac{2\ln 2}{q-1}\right). \tag{1.9}$$

Proof. We use a counting argument on the non-singular matrices in $\mathbb{F}_q^{k\times k}$. The first row $\mathbf{a}_1$ of $\mathbf{A}$ can be chosen uniformly in $\mathbb{F}_q^k \backslash \{\mathbf{0}\}$, while the second row $\mathbf{a}_2$ must chosen such that it is not in the space spanned by $\mathbf{a}_1$. For the ith row, we have

$$|\mathbb{F}_q^k \setminus \operatorname{span}(\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_i)| = q^k - q^i, \tag{1.10}$$

since $\operatorname{span}(\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_i) = \left\{\sum_{j=1}^{i} \mathbf{a}_j \cdot x_j : x_1, x_2, \ldots, x_i \in \mathbb{F}_q\right\}$. Hence, the number of non-singular matrices is $\prod_{i=0}^{k-1}(q^k - q^i)$. By division with $|\mathbb{F}_q^{k\times k}| = q^{2k}$, we obtain

$$\begin{aligned}
\mathbb{P}\left(\det \mathbf{A} \neq 0 \mid \mathbf{A} \xleftarrow{\$} \mathbb{F}_q^{k\times k}\right) &= q^{-2k} \cdot \prod_{i=0}^{k-1}(q^k - q^i) = \prod_{i=0}^{k-1}(1 - q^{i-k}) \\
&\geqslant \prod_{i=0}^{k-1} \exp\left(-q^{i-k} \cdot 2\ln 2\right) \\
&\geqslant \exp\left\{-2\ln 2 \cdot \lim_{k\to\infty} \sum_{i=0}^{\infty} q^{i-k}\right\} \\
&= \exp\left(-\frac{2\ln 2}{q-1}\right).
\end{aligned} \tag{1.11}$$

■

As an example, for a binary $k \times k$ matrix $\mathbf{A} \in \mathbb{F}_2^{k\times k}$ we have

$$\mathbb{P}\left(\det \mathbf{A} \neq 0 \mid \mathbf{A} \xleftarrow{\$} \mathbb{F}_q^{k\times k}\right) \geqslant \frac{1}{4}, \tag{1.12}$$

for an arbitrary k.

1.4 INFORMATION THEORY

In this section, we outline some basics in information theory. For a more detailed introduction of the described concepts, we refer to [CT91].

Classical information theory concerns transmission of information over a *channel*. A channel is a medium through which the exchange of information takes place and the channel can take many forms. A typical model of information transmission is given in Figure 1.2.

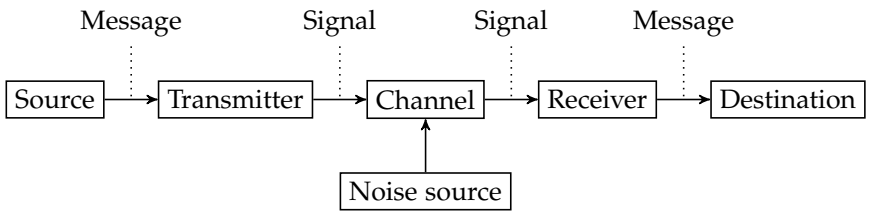


Figure 1.2: A model of an information-transmission system.

The mathematical study of information originates back to the celebrated landmark work of Shannon [Sha48]. Among many things, Shannon introduced the theory of *error-correcting codes* (which we will return to in Chapter 2) and a measure of the information content from a random source, called *entropy*.

Definition 1.2 (Entropy) Let X be a discrete random variable defined over the alphabet $\mathcal{X}$, taking values according to the probability distribution $P_X(x) = \mathbb{P}(X = x)$. The (Shannon) entropy of X is then defined as

$$H(X) \stackrel{\text{def}}{=} -\sum_{x \in \mathcal{X}} p_X(x) \cdot \log_2 p_X(x), \tag{1.13}$$

with the convention that $0 \cdot \log \frac{0}{q} = 0$ and $p \cdot \log \frac{p}{0} = \infty$ for $p > 0$.

The entropy $H(X)$ is a measure of the uncertainty associated with a random variable X. The entropy of a Bernoulli process, called the *binary entropy function* and denoted $h_2(p)$, is defined as follows,

$$h_2(p) \stackrel{\text{def}}{=} -p \cdot \log_2 p - (1-p) \cdot \log_2(1-p). \tag{1.14}$$

We emphasize that the entropy $H(X)$ of a random variable $X \sim \mathsf{Ber}_p$ and the binary entropy function $h_2(p)$ are essentially the same, but the former takes a random variable X as input and the latter takes a real-valued number

$p \in [0,1]$. In approximation of the binary entropy function, the following Taylor series expansion is useful,

$$h_2(p) = 1 - \frac{1}{2\ln 2} \sum_{i=1}^{\infty} \frac{(1-2p)^{2i}}{i \cdot (2i-1)}. \tag{1.15}$$

The binary entropy function can be extended to a larger alphabet $\mathbb{F}_q$. More specifically, for a positive integer $q \geqslant 2$, the *q-ary entropy function* $h_q : [0,1] \to \mathbb{R}$ is defined as

$$h_q(p) \stackrel{\text{def}}{=} p \cdot \log_q(q-1) - p \cdot \log_q p - (1-p) \cdot \log_q(1-p). \tag{1.16}$$

In Figure 1.3, we plotted the entropy function for different values of q.

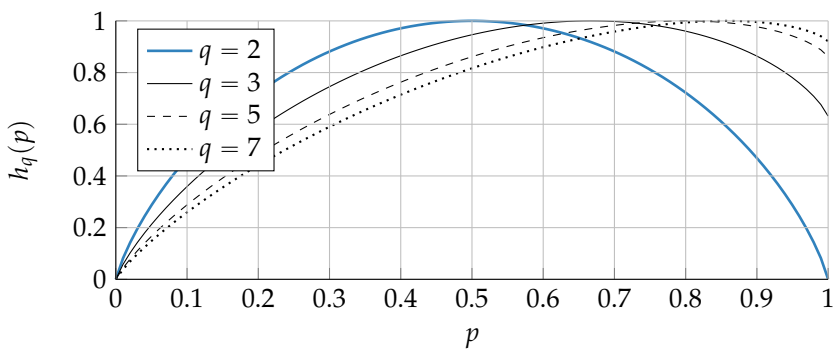


Figure 1.3: The entropy function h_q for different values of q.

While entropy is a measure of uncertainty about information, the *relative entropy* (sometimes called divergence or Kullback-Leibler distance) is a measure of distance between two distributions. Strictly speaking, the relative entropy is no proper metric because it neither satisfies the symmetric property nor the triangle inequality, but from time to time it can be useful to think of it as a distance. We define the relative entropy as follows.

Definition 1.3 (Relative entropy) Let $p_0(x)$ and $p_1(x)$ be probability distribution functions for the distributions D_0 and D_1 over alphabet $\mathcal{X}$. The relative entropy is then defined as

$$D(\mathsf{D}_0 \| \mathsf{D}_1) \stackrel{\text{def}}{=} \sum_{x \in \mathcal{X}} p_0(x) \cdot \log \frac{p_0(x)}{p_1(x)}. \tag{1.17}$$

The relative entropy gives the expected amount of information that is lost (in bits and per event) when a distribution D_0 is approximated by another distribution D_1.

Note: In the case of relative entropy, we use the symbol $\|$ as separation of the distributions, rather than as a concatenation operator.

Another distance measure in information theory is *Hamming distance*, which in contrast to relative entropy is a proper metric. The Hamming distance gives the difference between two strings or vectors of equal length.

Definition 1.4 (Hamming distance and weight) The Hamming distance between two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{F}_q^n$ is defined as the number of non-agreeing positions, i.e.,

$$d_{\mathrm{H}}(\mathbf{x}, \mathbf{y}) \stackrel{\text{def}}{=} |\{i \in \{1, 2, \ldots, n\} : x_i \neq y_i\}|. \tag{1.18}$$

Analogously, we define the *Hamming weight* of a vector $\mathbf{x}$ as the number of non-zero elements, i.e.

$$w_{\mathrm{H}}(\mathbf{x}) \stackrel{\text{def}}{=} |\{i \in \{1, 2, \ldots, n\} : x_i \neq 0\}|. \tag{1.19}$$

From above, it follows that $d_{\mathrm{H}}(\mathbf{x}, \mathbf{0}) = w_{\mathrm{H}}(\mathbf{x})$, where $\mathbf{0}$ is the all-zero vector of length n.

Using the Hamming distance, we can define geometrical objects, e.g. , *Hamming balls*. A Hamming ball with radius r is a set of points in a n-dimensional space, such that every point has Hamming distance at most r from the all-zero vector $\mathbf{0} \in \mathbb{F}_q^n$. We define it as follows.

Definition 1.5 (Hamming ball and sphere) A Hamming ball, denoted $\mathcal{B}_q(n, r)$, is a set of elements in $\mathbb{F}_q^n$ such that

$$\mathcal{B}_q(n, r) \stackrel{\text{def}}{=} \left\{\mathbf{v} \in \mathbb{F}_q^n : w_{\mathrm{H}}(v) \leqslant r\right\}. \tag{1.20}$$

Similarly, a *Hamming sphere*, denoted $\mathcal{S}_q(n, r)$, is a set of elements in $\mathbb{F}_q^n$ such that

$$\mathcal{S}_q(n, r) \stackrel{\text{def}}{=} \left\{\mathbf{v} \in \mathbb{F}_q^n : w_{\mathrm{H}}(v) = r\right\}, \tag{1.21}$$

implying

$$\mathcal{B}_q(n, r) = \bigcup_{i=0}^{r} \mathcal{S}_q(n, i). \tag{1.22}$$

In particular, when a Hamming ball or sphere is centered around a point $\mathbf{c} \in \mathbb{F}_q^n$ rather than around $\mathbf{0}$, we will write it as $\mathbf{c} + \mathcal{B}_q(n, r)$ and $\mathbf{c} + \mathcal{S}_q(n, r)$, respectively.

We will spend the remaining part of this section studying the cardinalities $|\mathcal{B}_q(n, r)|$ and $|\mathcal{S}_q(n, r)|$, in which asymptotic analysis of binomial coefficients will prove to be useful.

Proposition 1.5 For $\alpha \in [0,1]$ it holds that

$$\lim_{n\to\infty} \frac{1}{n} \cdot \log_2 \binom{n}{\alpha n} = h_2(\alpha) + o(1). \tag{1.23}$$

Proof. The Stirling approximation (see e.g. [Gri04]) asserts $n! \sim \left(\frac{e}{n}\right)^n \cdot \sqrt{2\pi}$. Therefore, it holds that

$$\binom{n}{\alpha n} \sim \frac{(n/e)^n}{(\alpha n/e)^{\alpha n} \cdot ((1-\alpha)\cdot n/e)^{(1-\alpha)n}} \cdot \frac{\sqrt{2\pi n}}{\sqrt{2\pi n \cdot \alpha} \cdot \sqrt{2\pi n \cdot (1-\alpha)}}.$$

The $(n/e)^n$ term cancels with $(n/e)^{\alpha n} \cdot (n/e)^{(1-\alpha)n}$, and $\sqrt{2\pi n}$ cancels with one of the terms in the denominator. The expression simplifies to

$$\binom{n}{\alpha n} \sim \frac{1}{(\alpha n/e)^{\alpha} \cdot ((1-\alpha)\cdot n/e)^{1-\alpha}} \cdot \frac{1}{\sqrt{2\pi n \cdot \alpha \cdot (1-\alpha)}}.$$

So, we have

$$\begin{aligned}\frac{1}{n}\log_2 \binom{n}{\alpha n} &\sim -\alpha \log_2 \alpha - (1-\alpha)\cdot \log_2(1-\alpha) - \frac{1}{2n}\cdot \log_2(2\pi \cdot \alpha \cdot (1-\alpha)) \\ &= h_2(\alpha) + o(1).\end{aligned}$$

We obtain the limit as n tends to infinity. ■

Corollary 1.6 For $\alpha \in [0,1]$ and $\beta \in [0,\alpha]$ it holds that

$$\lim_{n\to\infty} \frac{1}{n} \cdot \log_2 \binom{\alpha n}{\beta n} = \alpha \cdot h_2(\beta/\alpha) + o(1). \tag{1.24}$$

Using the q-ary entropy function, we can obtain asymptotically tight bounds for the size of a Hamming ball (and Hamming sphere).

Lemma 1.7 For $\alpha \leqslant [0,1]$ and $n \to \infty$, it holds that the volume of a Hamming ball $\mathcal{B}_q(n, \alpha \cdot n)$ is

$$|\mathcal{B}_q(n, \alpha \cdot n)| = \sum_{i=0}^{\alpha n} \binom{n}{i} (q-1)^i = q^{n h_q(\alpha) + o(n)}, \tag{1.25}$$

where $h_q(\cdot)$ is the q-ary entropy function.

Proof. The q-ary entropy function can be expressed in terms of the binary entropy function as follows

$$h_q(p) = p \cdot \log_q(q-1) + h_2(x) \cdot \log_q 2.$$

Let $r \stackrel{\text{def}}{=} \lfloor \alpha \cdot n \rfloor$. Using the result in (1.23) in conjunction with the above, we have that

$$\frac{1}{n} \cdot \log_q \left[\binom{n}{r} (q-1)^r \right] = \underbrace{\frac{r}{n} \cdot \log_q (q-1) + h_2 \left(\frac{r}{n} \right) \cdot \log_q 2}_{h_q(\alpha)} + o(1),$$

which yields an expression for the (overwhelmingly) largest term in (1.25). ■

1.5 HYPOTHESIS TESTING

Hypothesis testing is a statistical method that is highly relevant to cryptography. It has been extensively used in the past as a powerful tool in many different cryptanalytic contexts, where the problem of distinguishing between distributions is met. In this section, the notation of [CT91] is adopted.

We will now outline the problem of hypothesis testing. Let D_0 and D_1 be two different probability distributions defined over the same alphabet $\mathcal{X}$. The simplest form of hypothesis test is the *binary hypothesis test*; given a collection of elements $x_1, x_2, \ldots, x_n \in \mathcal{X}$ drawn at random (i.i.d.) from either D_0 or D_1, one should decide with distribution the collection of elements comes from. The two hypotheses are H_0, which is the *null hypothesis*, and the *alternative hypothesis* H_1, which are defined as follows

$$\begin{aligned} H_0 &: \mathsf{D} = \mathsf{D}_0, \\ H_1 &: \mathsf{D} = \mathsf{D}_1. \end{aligned} \tag{1.26}$$

For this purpose, one defines a *decision rule* – a function $\delta : \mathcal{X} \to \{0, 1\}$ taking a sample of $\mathcal{X}$ as input and defining a guess for each possible $x \in \mathcal{X}$, i.e.,

$$\delta = \begin{cases} 0, & \mathsf{D} = \mathsf{D}_0, \\ 1, & \mathsf{D} = \mathsf{D}_1. \end{cases} \tag{1.27}$$

Associated to the decision rule are two different types of incorrect decisions:

- *Type I error*: Rejecting the null hypothesis H_0 when it is true. This is sometimes called a false positive. The error probability is

$$\alpha \stackrel{\text{def}}{=} \mathbb{P}\left(\text{reject } H_0 \mid H_0 \text{ is valid}\right). \tag{1.28}$$

- *Type II error*: Rejecting the alternative hypothesis H_1 when it is true. This is sometimes called a false negative. The error probability is

$$\beta \stackrel{\text{def}}{=} \mathbb{P}\left(\text{reject } H_1 \mid H_1 \text{ is valid}\right). \tag{1.29}$$

We illustrate the error probabilities α and β in Figure 1.4.

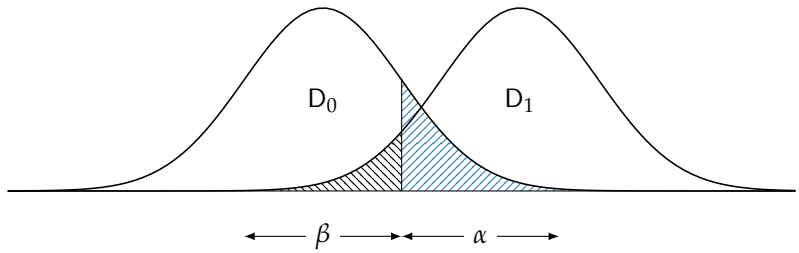


Figure 1.4: The error probabilities α and β of a hypothesis test.

THE NEYMAN-PEARSON LEMMA

A very important result, known as the *Neyman-Pearson lemma*, derives the shape of the optimal hypothesis test between two simple hypotheses. It can be formulated as follows.

Lemma 1.8 (Neyman-Pearson) Let $X_1, X_2, \ldots, X_n$ be a sequence of i.i.d. variables distributed according to a probability distribution D, and let $p_0(x)$ and $p_1(x)$ be probability distribution functions for the distributions D_0 and D_1. Consider the decision problem corresponding to the two hypotheses $H_0 : \mathsf{D} = \mathsf{D}_0$ and $H_1 : \mathsf{D} = \mathsf{D}_1$. For some $\tau \geqslant 0$, define a decision region

$$\mathcal{A} \stackrel{\text{def}}{=} \left\{ x_1, x_2, \ldots, x_n \ : \ \frac{p_0(x_1, x_2, \ldots, x_n)}{p_1(x_1, x_2, \ldots, x_n)} > \tau \right\} \tag{1.30}$$

and let $\alpha = p_0\left(\mathcal{A}^C\right)$ and $\beta = p_1(\mathcal{A})$. Let $\mathcal{B}$ be any decision region associated with error probabilities α^* and β^*. If $\alpha^* \leqslant \alpha$, then $\beta^* \geqslant \beta$.

The set $\mathcal{A}$ is called the *acceptance region* and its cardinality depends heavily on τ. Normally, we set $\tau = 1$ to obtain an equal comparison between the probability distributions D_0 and D_1.

Under the assumption that all samples are independent, we can formulate (1.30) as

$$\mathcal{A} \stackrel{\text{def}}{=} \left\{ x_1, x_2, \ldots, x_n \ : \ \sum_{i=1}^{n} \log_2 \frac{p_0(x_i)}{p_1(x_i)} > \log_2 \tau \right\}, \tag{1.31}$$

commonly referred to as a *likelihood-ratio test*.

ERROR ESTIMATION AND SAMPLE REQUIREMENT

We conclude the section about hypothesis testing by discussing the errors made in hypothesis tests and the number of samples required to achieve a certain error level. Although finding exact and analytic expressions for the

error probabilities α and β is normally not possible, estimates can be obtained by asymptotic approximation.

Stein's lemma [CT91] states that for a fixed α, the asymptotic behavior of the error probability β is given by

$$\lim_{n\to\infty} \frac{\log_2 \beta}{n} = -D\left(\mathsf{D}_0 \| \mathsf{D}_1\right). \tag{1.32}$$

The rate of which β decreases is independent of α, and according to Stein's lemma this situation always occurs. By some reformulation of the above expression, we can asymptotically express β as

$$\beta \approx 2^{-nD(\mathsf{D}_0 \| \mathsf{D}_1)}. \tag{1.33}$$

Suppose that we have a set $\mathcal{S}$ of 2^k sequences, where one sequence $\mathbf{x}$ is drawn from the distribution D_0 and the remaining $2^k - 1$ sequences are drawn from the distribution D_1. Moreover, assume that we want to distinguish $\mathbf{x}$ for a fixed α (type I error). What is the required number of samples (the length of each sequence), denoted n, for a certain error probability β? Denote the number of false positives with a random variable Y. Then, we have

$$\begin{aligned} \mathbb{E}\left(Y\right) &= \sum_{\mathbf{y}\in\mathcal{S}\setminus\{\mathbf{x}\}} \mathbb{P}\left(\mathbf{y} \text{ is misclassified}\right) \\ &= \left(2^k - 1\right) \cdot \beta = \left(2^k - 1\right) \cdot 2^{-nD(\mathsf{D}_0 \| \mathsf{D}_1)}. \end{aligned} \tag{1.34}$$

From the Markov bound (1.6), it holds

$$\mathbb{P}\left(Y \geqslant 1\right) \leqslant \mathbb{E}\left(Y\right). \tag{1.35}$$

Hence, we can bound the probability of guessing incorrectly (type II error) with a constant probability $\theta \in (0,1)$ by setting $\mathbb{E}\left(Y\right) = \theta$. Setting (1.34) equal to θ, we can solve for n and we get

$$n = \frac{\log_2 \frac{2^k - 1}{\theta}}{D\left(\mathsf{D}_0 \| \mathsf{D}_1\right)} \approx \frac{-k \cdot \log_2 \theta}{D\left(\mathsf{D}_0 \| \mathsf{D}_1\right)}. \tag{1.36}$$

In cryptanalysis, we often encounter problems where we need to distinguish between a uniform and a Bernoulli distribution. The following proposition will be relevant to us in the sequel.

Proposition 1.9 Let $\mathcal{S}$ be a set containing 2^k sequences, where one sequence $\mathbf{x}$ is drawn from the distribution Ber_ρ and $2^k - 1$ are drawn from the distribution U. Then, the required number of samples n required to distinguish $\mathbf{x}$ from the other samples is given by

$$n \approx \frac{-k \cdot \log_2 \theta}{1 - h_2(\rho)}, \tag{1.37}$$

for an error probability θ.

Proof. Let $X_i \in \mathsf{Ber}_\rho$ for $1 \leqslant i \leqslant n$ be independent random variables. Assume that a sequence $x_1, x_2, \ldots, x_n$ of samples from the distribution D has been observed and consider the hypotheses

$$\begin{aligned} H_0 : \mathsf{D} &= \mathsf{Ber}_\rho, \\ H_1 : \mathsf{D} &= \mathsf{U}. \end{aligned}$$

Then, the relative entropy of the two distributions is given by

$$D\left(\mathsf{Ber}_\rho \| \mathsf{U}\right) = \rho \cdot \log_2 \frac{\rho}{\frac{1}{2}} + (1-\rho) \cdot \log_2 \frac{1-\rho}{\frac{1}{2}} = 1 - h_2(\rho). \tag{1.38}$$

Finally, we know from (1.36) that the number of required samples is approximately

$$n \approx \frac{-k \cdot \log_2 \theta}{D\left(\mathsf{Ber}_\rho \| \mathsf{U}\right)} = \frac{-k \cdot \log_2 \theta}{1 - h_2(\rho)}. \tag{1.39}$$

■

Corollary 1.10 Under the same assumption as in Proposition 1.9, and with $\rho = \frac{1}{2} \cdot (1-\epsilon)$ where ϵ is small, we have that

$$n \approx \frac{-k \cdot \log_2 \theta}{1 - h_2(\rho)} \approx 2 \ln 2 \cdot \frac{-k \cdot \log_2 \theta}{\epsilon^2}. \tag{1.40}$$

Proof. Using the Taylor expansion of $h_2(\rho)$, we have

$$\begin{aligned} 1 - h_2(\rho) &\overset{(1.15)}{=} \frac{1}{2 \ln 2} \cdot \left\{ (1-2\rho)^2 + \mathcal{O}\left((1-2\rho)^4\right) \right\} \\ &= \frac{1}{2 \ln 2} \cdot \left\{ \epsilon^2 + \mathcal{O}(\epsilon^4) \right\}. \end{aligned} \tag{1.41}$$

Since ϵ is small, we may neglect the $\mathcal{O}(\epsilon^4)$ term. ■

1.5.1 THE WALSH TRANSFORM

The *Walsh transform* is a special case of the *discrete Fourier transform* and it has shown to be extremely useful in cryptanalysis. We define the Walsh transform as follows.

Definition 1.6 (Walsh transform) The Walsh transform of an n-ary boolean function $f = f(x_1, x_2, ...x_n)$ is a $\mathbb{F}_2^n \to \mathbb{F}_2^n$ mapping defined by

$$F(\mathbf{y}) = \sum_{\mathbf{x} \in \mathbb{F}_2^n} (-1)^{f(\mathbf{x}) + \langle \mathbf{x}, \mathbf{y} \rangle}, \tag{1.42}$$

for $\mathbf{y} \in \mathbb{F}_2^n$. The inverse is computed according to

$$f(\mathbf{x}) = 2^{-n} \cdot \sum_{\mathbf{y} \in \mathbb{F}_2^n} (-1)^{F(\mathbf{x}) + \langle \mathbf{x}, \mathbf{y} \rangle}. \tag{1.43}$$

The Walsh transform can be computed in time $\mathcal{O}(n \cdot 2^n)$ using a divide-and-conquer technique. This is called the *fast Walsh-Hadamard transform*.

1.6 ALGORITHMS AND COMPLEXITY THEORY

In this section, we introduce some concepts on algorithms and computational complexity theory. For more details, we refer to [Sha98] for a basic introduction and [Mor98] for a more in-depth reading.

Computers are devices which can perform one single thing: carry out computation steps to process information. An algorithm – a series of steps of computation, will be denoted $\mathcal{A}$. Although algorithms are executed in computers, they need not to have anything to do with them. A so-called *classical algorithm*, which can be carried out by a (classical) computer, could also be carried out by hand. If the steps of computation can be simultaneously computed by a set of computational devices, the algorithm is called *parallelizable*.

As opposed to a classical algorithm, a *quantum algorithm* can only be executed using a quantum computer making use of quantum superposition and entanglement to achieve, sometimes exponential speed-up over a classical computer.

The amount of computation needed when carrying out the steps of an algorithm is called the *computational complexity*. In essence, there are three types of complexity: *best case*, *average case* and *worst case*. Naturally, we will focus on average case and worst case, as these two measures will indicate the performance constraints of an algorithm.

One usually derives *upper bounds* and *lower bounds* on the number of computational steps. Let an arbitrary problem be denoted L. A problem is a collection of *instances* (which is the input string) along with a solution for every instance. The upper bound on the complexity of the problem L is determined by the complexity of the best known algorithm that determines the solution to every instance in L, while the lower bound is determined by what is theoretically possible to achieve in terms of algorithmic performance. A problem can be a *decision problem*, for which the answer is yes or no. It may also be a *search problem*[4], where the answer is solution.

Definition 1.7 (Polynomial and negligible) A function f is said to be *polynomial* if there exists a c such that $f(n) \leqslant n^c$ for all n. We denote f being

[4]For instance, »which is the 100th prime?« or »who is the fifth ninja turtle?«.

polynomial as $f(n) = \mathsf{poly}(n)$. A function f is said to be *negligible* if there exists an integer N such that $f(n) \leqslant 1/\mathsf{poly}(n)$ for all $n > N$. Negligible is asymptotically smaller than any inverse polynomial.

From a computational perspective, one is often interested in how efficient an algorithm can perform when the size n of the input is large. If the time $T(n)$ required to run through all steps of the computation is exponential, it is *intractable*. If the running-time on the other hand is polynomial, it is *tractable* and the computation is expected to finish within a »reasonable« amount of time. Particularly interesting are *probabilistic polynomial-time* (PPT) algorithms, which constitute a class of algorithms that are said to be *efficiently computable*.

Definition 1.8 (PPT algorithm) An algorithm (or Turing machine) $\mathcal{A}$ with input x is said to be a *probabilistic polynomial-time* algorithm if it uses randomness (coin flips) and has running time bounded by some polynomial $T(n) = \mathsf{poly}(n)$ in the input size $n = |x|$.

Probabilistic algorithms are so-called *randomized algorithms*, because they use random bits (or coin flips). Typically, we make a distinction between two types of randomized algorithms.

- A *Monte Carlo algorithm* is a randomized algorithm with a deterministic running time that has a non-zero probability of giving an incorrect output.
- A *Las Vegas algorithm* on the other hand, is a randomized algorithm that always produces a correct answer or raises a flag indicating failure (the symbol $\perp$ is commonly used for this purpose). Its running time is not deterministic, but it runs in expected finite time.

A Las Vegas algorithm can always be transformed into a Monte Carlo algorithm by forcing it to terminate at an early stage. A Monte Carlo algorithm on the other hand, can only be transformed into a Las Vegas algorithm if there exists a function that can verify that both negative and positive output produced by the Monte Carlo algorithm indeed is correct. If so, its probability of success can be arbitrarily amplified by running the algorithm several times to obtain a Las Vegas algorithm.

Proposition 1.11 (Las Vegas running time) Consider a Las Vegas algorithm which succeeds to produce a correct answer with probability ρ and outputs $\perp$ with probability $1 - \rho$. Let X be a random variable that represents the running time of the algorithm. Then, its expected running time is $\mathbb{E}(X) = \frac{1}{\rho}$.

Proof. Since we will run the algorithm until we observe a correct answer, X is distributed according to a geometric distribution. More specifically, the probability that we have to run the algorithm i times is given by

$$\mathbb{P}(X = i) = \rho \cdot (1 - \rho)^{i-1}. \tag{1.44}$$

It is a well-known result that a random variable with distribution given by (1.44) has expected value $\mathbb{E}(X) = \frac{1}{\rho}$. ■

Definition 1.9 We say that a randomized algorithm $\mathcal{A}_{\mathsf{L}}$ $(T(n), \theta)$-*solves* problem L if it runs in time at most $T(n)$ and produces a correct solution with probability at least θ.

Algorithms are theoretic constructions, sometimes embodied in lines of executable code, that are designed to solve a computational problem. *Complexity theory* concerns the classification of such computational problems in terms of their inherent difficulty and how to relate the different classes to each other (illustrated, somewhat simplified, in Figure 1.5).

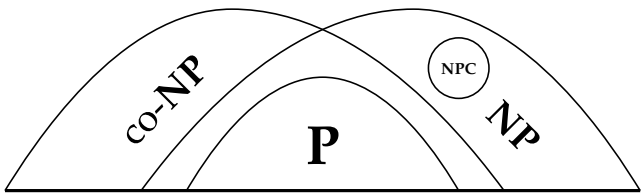


Figure 1.5: The polynomial hierarchy, under the well-established and widely accepted conjectures that **P** $\neq$ **NP** and co-**NP** $\neq$ **NP**.

Informally, **P** is the set of all decision problems solvable by a deterministic Turing machine in polynomial time, and **NP** is the set of all decision problems solvable by a non-deterministic Turing machine in polynomial time[5]. **P** is a subset of **NP**, because problems solvable by a deterministic Turing machine in polynomial time can also be solved by a non-deterministic machine in polynomial time. **NP**-complete problems are the hardest problems in **NP** and share a curious characteristic. A decision problem L is **NP**-complete if:

- L is in **NP**, and
- Every problem in **NP** is reducible to L in polynomial time (reduction is defined later).

[5]Or using algorithms that by magic always guess right.

Note that search problem versions of **NP**-complete problems are **NP**-hard (although, they are equally difficult to solve). We stress that **NP**-complete problems are not known to be solvable by a quantum computer. As previously mentioned, such problems are suitable for post-quantum cryptography.

Definition 1.10 (Reduction, informal) Problem L' is *reducible* to problem L if a (possibly hypothetical) algorithm $\mathcal{A}_{\mathsf{L}}$ that solves L efficiently may be used as sub-routine to solve problem L' efficiently. If the sub-routine is executed a polynomial number of times, it is a *polynomial reduction*, implying that problem L' is no harder than problem L, up to polynomial factors.

In cryptology, we often use *oracles* when building theory, in particular when constructing complexity reductions. An oracle is a black box with »magical« powers capable of solving some problem in a single step of computation. For instance, the oracle represent the hypothetical algorithm $\mathcal{A}_{\mathsf{L}}$ in previous definition. A special kind of oracles, *random oracles*, are capable of providing true randomness. Random oracles are particularly useful in schemes where strong randomness assumptions are required.

1.7 THE BIRTHDAY PARADOX AND COLLISION SEARCH

We will now discuss some variations of a very specific algorithmic problem and how to solve it efficiently. The results we give here serve as basis for many of the algorithms presented in this thesis and will be referenced frequently in subsequent chapters.

The *birthday paradox* is a well-known result from elementary probability theory, which has proven itself invaluable when constructing randomized algorithms for collision search. More specifically, we are concerned with algorithms that store information in memory to be able to execute certain procedures, e.g., matching more efficiently. Algorithms of this kind are often referred to as *time–memory trade-off* (or space–time trade-off) algorithms.

Consider the following. Let $\mathcal{S}$ be a set with cardinality $|\mathcal{S}| = N$. Suppose that items are sampled at random from $\mathcal{S}$. The birthday paradox states that as N becomes larger, the expected number of samples we need to randomly pick some item in $\mathcal{S}$ twice approaches

$$\sqrt{\frac{\pi}{2} \cdot N}. \tag{1.45}$$

The birthday problem can be stated as a general computational problem, if we carefully define the operations associated with the problem. Consider the following:

Problem 1.12 (2-COL) Given lists $\mathcal{L}_1, \mathcal{L}_2$ containing elements from the set $\mathcal{S}$, determine the (possibly empty) list

$$\mathcal{L} = \{(x, y) : x \in \mathcal{L}_1, y \in \mathcal{L}_2 \text{ such that } x \oplus y = 0\} \tag{1.46}$$

under some operation $\oplus$. We define this operation to be $\mathcal{L}_1 \diamond \mathcal{L}_2 = \mathcal{L}$.

FINDING COLLISIONS

If $\mathcal{S}$ is an ordered set, then the elements can be deterministically ordered[6] in some way (to define the concepts of small and large) using binary relations $\preceq$ and $\succeq$. If so, then to solve Problem 1.12, we can employ a *sort-and-match* algorithm, which sorts two lists $\mathcal{L}_1, \mathcal{L}_2$ separately, and then sequentially matches their elements. It is a well-known fact that sorting a list of n elements costs $\mathcal{O}(n \log n)$ operations. To simplify analysis, we will frequently assume that the number of elements in each list is equal. Therefore, the complexity of sorting the two (or a constant number of) lists of length n remains $\mathcal{O}(n \log n)$.

The algorithm begins simultaneously to read at the bottom of each list, comparing their respective elements. Each element is compared according to the following rules:

- If one element is smaller than the other, it is discarded. Since the lists are sorted in descending order, the next element cannot be smaller than the one it discarded. This procedure is iterated until one list is empty or a match is found.

- If a match is found, the elements are included in the target list. Then, we fix the element in one list $\mathcal{L}_1$ and compare it with the element above in the other list $\mathcal{L}_2$. This is also a match, so it is included in the target list. We repeat until there are no matches. Then, we discard the fixed element and return to (a).

The piece of pseudocode given in Algorithm 1 describes the sort-and-match procedure in more rigor.

Every element in a list is potentially a match with n elements in the other list, so if all element are equal we obtain the worst case. The upper bound of matches is the number of pairs we can choose, which is n^2 (if the operation $\diamond$ is commutative, then this number is $\binom{n}{2}$). Therefore, the worst case complexity of Sort-and-match is $\mathcal{O}(n^2 + n \log n) = \mathcal{O}(n^2)$. However, if the length n of lists $\mathcal{L}_1$ and $\mathcal{L}_2$ are chosen according to (1.45), then the average-case complexity is $\mathcal{O}(n \log n)$.

[6] Although the Problem 1.12 is defined for any type of comparable mathematical object, we suggest the reader to think of elements of $\mathbb{Z}$.

Algorithm 1 (Sort-and-match)

Input: Lists $\mathcal{L}_1, \mathcal{L}_2$ with comparable elements.

Output: List $\mathcal{L}$.

```
Sort the lists L1, L2;
n ← |L1| + |L2|;
i ← 0, j ← 0;
while do
    if L1[i] < L2[j] then
        i ← i + 1;
    else if L1[i] > L2[j] then
        j ← j + 1;
    else
        l ← i;
        while L1[l] = L2[j] do
            LR ← (L1[l], L2[j]);
            l ← l + 1;
return L
```

Algorithm 2 (Hash-match)

Input: Lists $\mathcal{L}_1, \mathcal{L}_2$ with comparable elements.

Output: List $\mathcal{L}$.

```
for x ∈ L1 do
    H[x] ← x;
for y ∈ L2 do
    if H[y] has at least one entry then
        for x ∈ H[y] do
            L ← (y, x);
return L
```

Another algorithmic approach is *hash-based matching*, which is inherently more efficient than Sort-and-match and does not require $\mathcal{S}$ to be an ordered set. Instead of matching the elements by sorting, one uses a hash table (a constant-time access associative data structure) to carry out matching of elements in the lists. Problem 1.12 is solved in the following way: all elements are read from the first list in any order and inserted into the hash table. With the elements stored in the hash table, all elements are read from the other list and for each element the algorithm checks in the hash table whether there is a match or not.

Clearly, Hash-match has to read through each list one time which requires

$\mathcal{O}(n)$ time and computation. The worst case is the same as for Sort-and-match, i.e., when all elements are equal in which case we have a complexity of $\mathcal{O}(n^2)$. By choosing the list size n according to (1.45), the expected complexity is $\mathcal{O}(n)$.

Observation 1.13 (2-COL complexity) We can expect to find at least one collision among N distinct elements using $n \sim \sqrt{N}$ randomly drawn samples and the same order of time and computation.

It is clear from above that Algorithm 1 and Algorithm 2 implements the operation $\diamond$, and that both algorithms are sound and complete.

INCREASING THE NUMBER OF LISTS

A natural step is to make a generalization of the collision problem of two lists into a collision problem of k lists. The problem becomes particularly interesting when k is a power of 2. To the best of the author's knowledge, the generalization of the birthday problem was introduced in by Blum, Kalai and Wasserman in [BKW00] and later on extended to other applications by Wagner in [Wag02].

Problem 1.14 (k-COL) Given lists $\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_k$, for some positive integer k, containing elements from the set $\mathcal{S}$, determine the (possibly empty) list

$$\mathcal{L} = \left\{ (x_i)_{i\in\{1,2,\ldots,k\}} : x_i \in \mathcal{L}_i \text{ such that } \bigoplus_{i\in\{1,2,\ldots,k\}} x_i = 0 \right\} \tag{1.47}$$

under some operation $\oplus$.

1.7.1 SEARCHING FOR COLLISIONS IN VECTORSPACES

We will now study the special case when the set $\mathcal{S}$ is a vectorspace over some alphabet $\mathcal{X}$. Consider the following function.

Definition 1.11 (Masking function) Let $\mathcal{M} \subseteq \{1, 2, \ldots, n\}$ with $|\mathcal{M}| = m$ and $\mathbf{x} = (x_i)_{i\in\{1,2,\ldots,n\}} \in \mathcal{X}^n$. If $\phi_{\mathcal{M}}$ is a linear map such that

$$\begin{array}{rccc} \phi_{\mathcal{M}} : & \mathcal{X}^n & \longrightarrow & \mathcal{X}^m \\ & \cup & & \cup \\ & \mathbf{x} & \longmapsto & (x_i)_{i\in\mathcal{M}}, \end{array} \tag{1.48}$$

then $\phi_{\mathcal{M}}$ is called a *masking function* for the *masking set* $\mathcal{M}$. When there is no ambiguity, we usually leave out the subscript and write ϕ for notational simplification.

The masking function ϕ defines an equivalence relation on $\mathcal{X}^m$, and all elements that are equal in the positions $\mathcal{M}$ belong to the same equivalence class.

Example 1.1 *Let $\mathcal{X}$ be an alphabet,*

$$\mathbf{x} = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 & x_5 \end{pmatrix} \in \mathcal{X}^5$$

be a vector and $\mathcal{M} = \{1, 2, 5\}$ be a masking set. Then

$$\phi_{\mathcal{M}}(\mathbf{x}) = \begin{pmatrix} x_1 & x_2 & x_5 \end{pmatrix} \in \mathcal{X}^3.$$

Let us now consider the k-COL problem over a vectorspace $\mathcal{X}^n$ when $k = 4$, i.e., when we have four lists that we assume have equal size. A solution $\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 + \mathbf{x}_4$ must satisfy the constraints

$$\mathbf{x}_i \in \mathcal{L}_i, \quad i \in \{1, 2, \ldots, 4\}, \text{ such that } \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 + \mathbf{x}_4 = \mathbf{0}. \tag{1.49}$$

By exhaustive search, we would need to spend $\prod_{i \in \{1,2,\ldots,4\}} |\mathcal{L}_i|$ operations. Clearly, a square-root gain in time complexity is possible by matching two separate sets $(\mathcal{L}_1 \times \mathcal{L}_2) \diamond (\mathcal{L}_3 \times \mathcal{L}_4)$, at the expense of memory trade-off proportional to the gain, because two of the lists must be put in a data structure.

A different algorithmic approach takes advantage of the masking function ϕ. First, we select a random masking set

$$\mathcal{M} \xleftarrow{\$} \{1, 2, \ldots, n\} \tag{1.50}$$

such that $|\mathcal{M}| = \alpha \cdot n$ for some $\alpha > 0$. Secondly, we merge the lists $\mathcal{L}_1$ and $\mathcal{L}_2$ using $\phi_{\mathcal{M}}$, according to

$$\begin{aligned} \mathcal{L}_{12} = \quad & \phi_{\mathcal{M}}(\mathcal{L}_1) \diamond \phi_{\mathcal{M}}(\mathcal{L}_2) \\ = \quad & \{\mathbf{x} \oplus \mathbf{y} : \mathbf{x} \in \mathcal{L}_1,\ \mathbf{y} \in \mathcal{L}_2,\ \phi_{\mathcal{M}}(\mathbf{x}) \oplus \phi_{\mathcal{M}}(\mathbf{y}) = \mathbf{0}\} \end{aligned} \tag{1.51}$$

and equivalently,

$$\mathcal{L}_{34} = \phi_{\mathcal{M}}(\mathcal{L}_3) \diamond \phi_{\mathcal{M}}(\mathcal{L}_4). \tag{1.52}$$

Finally, we compute $\mathcal{L} = \mathcal{L}_{12} \diamond \mathcal{L}_{34}$. The whole merging tree is illustrated in Figure 1.6.

Let us analyze the above procedure. Every element $\mathbf{x} \in \mathcal{L}$ must be a solution to the k-COL problem, so this algorithm must be sound. However, the algorithm is not complete because a correct solution may not be included in the list, since the operation $\diamond$ in general is not associative, meaning that for lists A, B, and C, there is not necessarily equality between the sets $(A \diamond B) \diamond C$ and $A \diamond (B \diamond C)$. Obviously, this will pose a problem if there is only a single or a few solutions and the probability of finding a solution is very small.

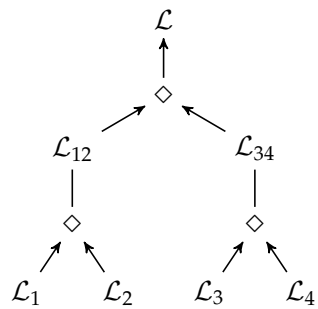


Figure 1.6: Merging tree for 4-COL.

Lemma 1.15 Assume that the elements in the lists $\mathcal{L}_i$ and $\mathcal{L}_j$ are uniformly sampled from $\mathcal{X}^n$. Let $|\mathcal{L}_i| = |\mathcal{L}_j| = |\mathcal{X}|^m$. Then, if $\mathcal{M} \subseteq \{1, 2, \ldots, n\}$ and $|\mathcal{M}| = m$,

$$\mathbb{E}\left(|\mathcal{L}_{ij}|\right) = \mathbb{E}\left(|\phi(\mathcal{L}_i) \diamond \phi(\mathcal{L}_j)|\right) \approx |\mathcal{X}|^m \tag{1.53}$$

is the number of elements in the resulting list.

Proof. For any $\mathbf{x} \in \mathcal{L}_i$ and $\mathbf{y} \in \mathcal{L}_j$, we have that

$$\mathbb{P}\left(\phi(\mathbf{x}) \oplus \phi(\mathbf{y}) = \mathbf{0}\right) = |\mathcal{X}|^{-m}. \tag{1.54}$$

There are $|\mathcal{X}|^m \cdot |\mathcal{X}|^m$ such pairs $(\mathbf{x}, \mathbf{y})$, so the expected list size is

$$\mathbb{E}\left(|\mathcal{L}_{ij}|\right) = \sum_{\mathbf{x} \in \mathcal{L}_i, \mathbf{y} \in \mathcal{L}_j} \mathbb{P}\left(\phi(\mathbf{x}) \oplus \phi(\mathbf{y}) = \mathbf{0}\right) = \sum_{\mathbf{x} \in \mathcal{L}_i, \mathbf{y} \in \mathcal{L}_j} |\mathcal{X}|^{-m} = |\mathcal{X}|^m. \tag{1.55}$$

■

Let us now consider a concrete example. For the case $k = 4$, we may set the $m = \alpha \cdot n = \frac{1}{3} \cdot n$. Then, the expected number of elements in each list is $|\mathcal{L}_{12}| = |\mathcal{X}|^{n/3}$ and $|\mathcal{L}_{34}| = |\mathcal{X}|^{n/3}$. Since the number of remaining positions is $|\{1, 2, \ldots, n\} \setminus \mathcal{M}| = \frac{2}{3} \cdot n$, we have

$$\mathbb{P}\left(\mathbf{x} \oplus \mathbf{y} = \mathbf{0} \mid \phi(\mathbf{x}) = \phi(\mathbf{y}) = \mathbf{0}\right) = |\mathcal{X}|^{-2n/3}. \tag{1.56}$$

There are $|\mathcal{X}|^{2n/3}$ pairs of the form $(\mathbf{x}, \mathbf{y})$ where $\mathbf{x} \in \mathcal{L}_{12}$ and $\mathbf{y} \in \mathcal{L}_{34}$, so the expected size of the resulting list is

$$\mathbb{E}\left(|\mathcal{L}|\right) = \sum_{\mathbf{x} \in \mathcal{L}_i, \mathbf{y} \in \mathcal{L}_j} \mathbb{P}\left(\mathbf{x} \oplus \mathbf{y} = \mathbf{0} \mid \phi(\mathbf{x}) = \phi(\mathbf{y}) = \mathbf{0}\right) = 1. \tag{1.57}$$

Thus, we expect to have at least one solution in the list $\mathcal{L}$ with high probability. To analyze the complexity, we use the results we derived for Hash-match, implying that lists can then be merged under the operation $\diamond$ in (expected) linear time, i.e., proportional to the list size. As we previously established, the size of each list is $|\mathcal{X}|^{n/3}$, so the total complexity is $\mathcal{O}(|\mathcal{X}|^{n/3})$.

A more general result for k being an arbitrary power of two, is the following.

Lemma 1.16 Let $\mathcal{M}_t \xleftarrow{\$} \{1, 2, \ldots, n\}$ at merging level $t \in \{1, 2, \ldots, k\}$ be chosen such that $\mathcal{M}_t \cap \mathcal{M}_s = \varnothing$ for all $s \neq t$ and $|\mathcal{M}_t| = \alpha \cdot n$ with $\alpha = (1 + \log k)^{-1}$. Furthermore, let the elements of the lists be uniformly and independently sampled from the vectorspace $\mathcal{X}^n$. Then with high probability, k-COL can be (partially) solved in $\mathcal{O}(k \cdot |\mathcal{X}|^{n/(1+\log k)})$ time and $\mathcal{O}(|\mathcal{X}|^{n/(1+\log k)})$ memory.

Proof. Set $\beta = \log k$. For every list, we sample $|\mathcal{X}|^{n/(1+\beta)}$ elements uniformly and independent from $\mathcal{X}^n$. We create $\frac{1}{2} \cdot \beta$ disjoint pairs of lists and merge them, as given in Figure 1.7.

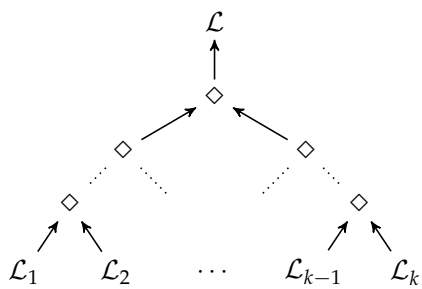


Figure 1.7: Merging tree for k-COL.

By recursively merging levels $t \in \{1, 2, \ldots, \beta - 1\}$ in the above tree, we have

$$\begin{aligned} \left| \{1, 2, \ldots, n\} \setminus \bigcup_{t \in \{1,2,\ldots,\beta-1\}} \mathcal{M}_t \right| &= n - \sum_{t \in \{1,2,\ldots,\beta-1\}} |\mathcal{M}_t| \\ &= n \cdot \left(1 - \frac{\beta - 1}{\beta + 1}\right) = n \cdot \frac{2}{1 + \beta}. \end{aligned} \tag{1.58}$$

If the pair $(\mathbf{x}, \mathbf{y})$ is chosen from the two final lists, then the probability that the two vectors match in the remaining positions is

$$\mathbb{P}\left(\mathbf{x} \oplus \mathbf{y} = \mathbf{0} \mid \phi(\mathbf{x}) = \phi(\mathbf{y}) = \mathbf{0}\right) = |\mathcal{X}|^{-2n/(1+\beta)}, \tag{1.59}$$

Lemma 1.15 states that for any pair of lists $(\mathcal{L}_i, \mathcal{L}_j)$, the expected resulting list size is

$$\mathbb{E}\left(|\mathcal{L}_{ij}|\right) = \mathbb{E}\left(|\phi_{\mathcal{M}_t}(\mathcal{L}_i) \diamond \phi_{\mathcal{M}_t}(\mathcal{L}_j)|\right) \approx |\mathcal{X}|^{n/(1+\beta)}. \tag{1.60}$$

By induction, the above is satisfied at any stage $t \in \{1, 2, \ldots, \beta - 1\}$ and for any pair of lists. The lists that will merge into the final list contains together $|\mathcal{X}|^{2n/(1+\beta)}$ such pairs. Therefore, list final list is expected to be of size

$$\begin{aligned} \mathbb{E}\left(|\mathcal{L}|\right) &= \sum_{\mathbf{x}\in\mathcal{L}_i, \mathbf{y}\in\mathcal{L}_j} \mathbb{P}\left(\mathbf{x} \oplus \mathbf{y} = \mathbf{0} \mid \phi_{\mathcal{M}_{t-1}}(\mathbf{x}) = \phi_{\mathcal{M}_{t-1}}(\mathbf{y}) = \mathbf{0}\right) \\ &= \sum_{\mathbf{x}\in\mathcal{L}_i, \mathbf{y}\in\mathcal{L}_j} |\mathcal{X}|^{-2n/(1+\beta)} = 1. \end{aligned} \tag{1.61}$$

From (1.61) it follows that, with high probability, there is at least one surviving candidate in the final list $\mathcal{L}$.

■

1.8 CRYPTOGRAPHY AND CRYPTANALYSIS

A natural question one might ask is: what characterizes a good measure of the security of a system? Security is a broad concept and the answer is not straightforward. Intuitively, it means that a ciphertext encrypted with a secure cryptosystem should reveal no information about the secret key and message. But this is by no means a rigorous definition.

Different paradigms are used to reinforce the security claims of cryptography. Security of some cryptographic primitives is based on pure heuristic arguments, while the security of others is supported by the simple fact that they have resisted cryptanalysis for a long time. This kind of security is often called *empirical security*. Primitives that are *unconditionally secure* are under no circumstances possible to break; such claims are backed up by information theoretical arguments. No adversary, even equipped with unbounded computational power, can break a cryptographic primitive that achieves unconditional security. Unfortunately, primitives of this kind are in general rather impractical and therefore uncommon in practice. The security of most cryptosystems today rely on *computational hardness assumptions*. Essentially, this means that the security is based on reduction from problems that are widely believed to be hard on average. The security proofs usually involve information theoretical and complexity theoretical arguments.

Cryptanalysis are for obvious reasons directed towards systems with security of empirical or provable nature; a cryptographic attack directed towards an unconditionally secure primitive would be absurd, implying a (partial) collapse of information theory or mathematics as a whole. A successful crypt-

analytic attack is usually characterized by that it uses some, possibly unintentional, structure in the primitive which in some sense reduces the cryptanalytic problem space into something smaller.

During the recent decades, cryptanalysis have shown that even provably secure cryptography can be compromised (e.g., RSA in SSL[7]). In implementations, both software and physical hardware can, and does, in some sense leak information which can be used to exploit, say power leakage in the circuitry, to attack a theoretically secure system. Although it is far out of scope for this thesis, it should not be left without mention. One can argue, as a rule of thumb, that Kerckhoffs's principle should not only apply to the cryptographic primitive, but also to all the levels of the implementation that are, or could be, prone to attacks.

1.8.1 ONE-WAY FUNCTIONS

We will start off by introducing the concept one-way functions. A one-way function is a function that can be efficiently computed on every input, but is computationally hard to invert given an image of a random input. A one-way function is a so-called *key-less primitive*. We define it (adopting notation from [Gol07]) as follows.

Definition 1.12 (One-way function) Let $f : \mathcal{X} \to \mathcal{Y}$ be a function that satisfies the following two conditions:

- *Easy to compute*: $f(x)$ is efficiently computable for all $x \in \mathcal{X}$,
- *Hard to invert*: for a randomly drawn $y \xleftarrow{\$} \mathcal{Y}$, computing $f^{-1}(y)$ is computationally infeasible in the average case. For all sufficiently large n and for every randomized algorithm $\mathcal{A}$ that runs in time polynomial in n
$$\mathbb{P}\left(\mathcal{A}(y) = f^{-1}(y)\right) < \mathsf{negl}(n). \tag{1.62}$$

If so, then we say that f is a one-way function.

In order to be useful, the computation of an efficiently computable function should not lead to unacceptable execution speeds in the cryptographic applications it is intended to be used for. In order to be secure, infeasible must assert that it is not possible to evaluate the function in a reasonably long time period using the best known algorithm.

When constructing one-way functions, one often takes inspiration from intractable problems, such as the class of **NP**-complete[8] problems. Another

[7]See Common Vulnerabilities and Exposures – CVE-2003-0147.

[8]In the introduction, we discussed the possibility of using the knapsack problem for the purpose of building a one-way function.

popular source of computationally hard problems used in construction of one-way functions is number theory; two very familiar problems are factoring and discrete logarithm problems in cyclic groups. Unfortunately, many problems in number theory, including the two previous ones, have shown to be unsuitable for post-quantum cryptography as they can be efficiently computed by a quantum computer.

Whether one-way functions exist or not is still an open question in computer science, and proving their existence would resolve the long-lived open question regarding $\mathbf{P} \neq \mathbf{NP}$, implying that the two classes are not equal [Gol07].

Cryptographic hash functions are attempts to create one-way functions, although their one-wayness is usually a heuristic claim (cf. empirical security); a hash function is in most cases needed to be very fast in practice, while a provably secure function that involves an intractable problem is inherently slow in implementation.

In cryptography, so-called *trapdoor one-way functions* play a very significant role, which will be thoroughly assessed in the sequel. Figure 1.8 illustrates the principle of a trapdoor one-way function.

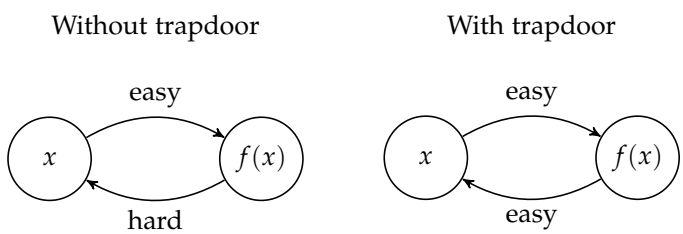


Figure 1.8: An ideal one-way function f.

We define a trapdoor one-way function as follows.

Definition 1.13 (Trapdoor one-way function) We say that a function is a trapdoor one-way function, if it is one-way and has the property that given some non-trivial information, its inverse can be computed efficiently.

1.8.2 SYMMETRIC CRYPTOGRAPHY

Symmetric-key ciphers constitute a class of cryptographic algorithms that allows for secure communication using the same key for both encryption and decryption. There are essentially two (main) subclasses of symmetric-key ciphers: *block ciphers* and *stream ciphers*. Block ciphers encrypt the ciphertext in a block-wise manner, with a typical block size ranging between 64 and 256 bits. Some block ciphers also support a variable length. In contrast, stream ciphers

use pseudorandom generators to obfuscate the message, usually by simply adding the pseudorandom stream, called the keystream, onto the message[9]. The receiver decrypts by subtracting the same keystream from the ciphertext. Stream ciphers accept a continuous stream of message bits making them very suitable for high-demand applications such as cellular networks and digital media streams.

The symmetric-key ciphers, and stream ciphers in particular, are often used in time-constrained environments, such as in mobile communication protocols. Whereas this kind of ciphers are constructed using simple building blocks and operations such as finite field arithmetic and substitution, their simplicity allows them to perform both encryption and decryption at a very high speed.

Apart from applications in encryption schemes, the symmetric ciphers can be used to construct other cryptographic primitives such as *message authentication codes* and *cryptographic hash functions*. We omit the details as this is out of scope for this thesis – for a thorough review, refer to, e.g., [Sma03].

Definition 1.14 (Symmetric-key cryptosystem) A (complete) symmetric-key encryption scheme is a triple $\Delta = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ of efficiently computable functions (algorithms) such that

- $\mathsf{KeyGen}(\lambda)$ outputs key k, given security parameter λ as input,
- $\mathsf{Enc}(m, k)$ outputs ciphertext c, given message m and key k,
- $\mathsf{Dec}(c, k)$ outputs message m, given ciphertext c and key k.

Additionally, it must hold for all λ, all k by KeyGen, all messages m, and all ciphertexts c output by Enc, that $\mathsf{Dec}(c) = m$.

The security parameter λ describes the asymptotic behavior of a scheme. Ideally, the algorithms (KeyGen, Enc, Dec) used in the cryptosystem should run in time polynomial in λ. At the same time, the cryptosystem must not be susceptible to attacks that run in time polynomial in λ, and therefore any polynomial-time attack may succeed with probability at most negligible in λ.

1.8.3 PUBLIC-KEY CRYPTOGRAPHY

In public-key cryptography, a key pair consisting of a public and a secret key is used. The public key is accessible[10] to anyone and can be used solely for encryption. In order to decrypt, the corresponding secret key must be used.

[9] Sometimes, this kind of cipher is called additive stream cipher.

[10] Since the same key is assumed to be used in encryption of several messages, public-key cryptography cannot achieve perfect security.

To achieve public-key cryptography, (conjectured) trapdoor one-way functions are employed. Informally, we could say that the computation $x \to f(x)$ is used for encryption, while the inverse computation $f(x) \to x$ is used for decryption.

Definition 1.15 (Public-key cryptosystem) A (complete) public-key encryption scheme is a triple $\Delta = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ of efficiently computable functions (algorithms) such that

- $\mathsf{KeyGen}(\lambda)$ outputs a key pair with secret and a public key (k_s, k_p), given security parameter λ as input,
- $\mathsf{Enc}(m, k_p)$ outputs ciphertext c, given message m and public key k_p,
- $\mathsf{Dec}(c, k_s)$ outputs message m, given ciphertext c and secret key k_s.

As before, we require that for all λ, all k by KeyGen, all messages m, and all ciphertexts c output by Enc, it holds $\mathsf{Dec}(c) = m$. In some cryptographic schemes, this holds except with exponentially small probability; this is generally sufficient in practice.

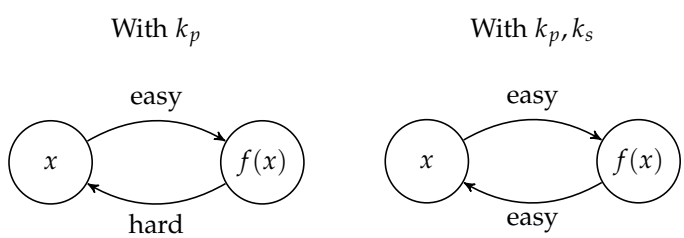


Figure 1.9: An ideal public-key cryptosystem based on a trapdoor one-way function f.

1.8.4 ATTACK MODELS AND SETTINGS

There are essentially three different cryptanalytic attacks applicable on asymmetric cryptography.

- **Key recovery**: In a *key-recovery attack* an adversary will recover the secret key. Resilience against key-recovery attacks is a very weak property, since it does not guarantee that no information leaks from the ciphertext.

- **Message recovery**: Related to the prior attack, in a *message-recovery attack* an adversary will recover one out of possibly many messages. Resilience against key-recovery attacks is a stronger property. Naturally, a successful key-recovery attack implies a message-recovery attack.

- **Distinguishing**: In a *distinguishing attack* an adversary neither recovers the key nor a message. Instead, it classifies whether an intercepted sequence is random or stems from a (fixed) cipher.

Security of a cryptographic primitive must hold in the presence of a malicious adversary. Attack settings depend heavily on different assumptions of what the adversary is capable of doing. A passive adversary can only observe (and intercept communicated information), while an active adversary can interact – sometimes even adaptively.

- **Known ciphertext**: The weakest adversary model is the *known-ciphertext* (or ciphertext-only, COA) attack. Given a set of intercepted ciphertexts c, the adversary tries to reveal the message $m \in \mathcal{M}$, or more preferably, the secret key $k_s \in \mathcal{K}$. For instance, if the key space is very small it allows for an exhaustive search to be carried out, computing the decryption for all keys. If the message is known to be distributed according to some distribution D, then the distribution of each decryption can be tested against D using some distance measure to verify if the guess is correct or not. More generally, if $|\mathcal{M}| \cdot |\mathcal{K}| > |\mathcal{C}|$, then we require more ciphertexts to uniquely determine m or k_s.

- **Chosen plaintext**: Slightly stronger is the *chosen-plaintext* attack model (CPA), in which the adversary is given access to an encryption oracle under some key k_p, which allows for choosing messages arbitrarily and determine their corresponding ciphertexts. For instance, in *differential cryptanalysis* the attacker chooses a set of messages that differ in a way which makes it beneficial to the attacker. In the *batch chosen-plaintext attack*, the adversary chooses all messages in anticipation. On the contrary, in the *adaptive chosen-plaintext attack* the adversary performs a sequence of queries, where subsequent messages can be chosen using information from prior encryptions.

- **Chosen ciphertext**: The strongest, but also least realistic attack model is *chosen-ciphertext* (CCA). The adversary is given access to both an encryption oracle and a decryption oracle and may therefore both encrypt messages and decrypt ciphertexts, but only a polynomial[11] number of

[11] This is the reason why it is sometimes called a lunchtime or midnight attack; the attacker is given access to the machine while the legitimate owner is at lunch or asleep.

them and the challenge ciphertext may not itself be decrypted. In a slightly boosted adversarial setting (CCA2), the attacker can adaptively choose messages and obtain ciphertexts, which in turn may be used to choose ciphertexts that can be decrypted to obtain messages.

BRUTE-FORCE ATTACKS

The simplest cryptanalytic attacks are *brute-force* attacks. As the name suggests, this kind of attack relies on the assumption that the attacker has access to computational power comparable to the claimed security level (or is lucky). If the key space is $|\mathcal{K}|$, then the expected number of tries is $\frac{1}{2} \cdot |\mathcal{K}|$.

Since brute-force attacks in no way exploit underlying properties of the cipher, they can be applied to virtually all cryptography. Therefore, the brute force complexity serves as a very crude upper bound on the security of a cipher.

It might seem that brute force is a simple theoretical and impractical upper bound. However, brute-force attacks are easily parallelizable and the task can be distributed over computer clusters or specialized hardware (e.g. FPGA based circuitry or GPU computation). Bernstein argues in [Ber05] that albeit (at most) linear speed-up is achieved, the cost often is less than serially executed attacks that are considered to break a cryptographic construction.

TIME–MEMORY TRADE-OFF ATTACKS

A time–memory trade-off (as introduced in Section 1.7) attack uses memory to speed up computation. It is divided into a pre-processing phase and a real-time phase. First, during the pre-processing phase, the algorithm makes a series of computational steps collecting information about the cipher, or storing information of frequently occurring computations. The information retrieved or computed is stored in memory, usually in look-up tables or similar. When entering the real-time phase, the algorithm uses the stored information with unknown information retrieved during the real-time phase. Time–memory trade-off techniques are not restricted to cryptanalysis; as an example, dynamic programming is a very general method for solving a problem by dividing it into sub-problems.

The first published time–memory trade-off attack appears in [Hel80], successfully mounting a chosen-plaintext attack on a block cipher. The attack in [Hel80] describes a time-memory trade-off technique commonly called a *meet-in-the-middle* (MITM) attack[12]. When applicable, it is usually used – like

[12]This attack should not be confused with the man-in-the-middle attack.

the brute force attack – as security upper bound (or equivalently, as a complexity lower bound). We will now give a very simple example of such an attack.

Example 1.2 *Assume that an attacker knows a message m and a ciphertext c that was encrypted twice with two unknown keys k_1 and k_2 such that*

$$c = \mathsf{Enc}(\mathsf{Enc}(m, k), k').$$

A brute force attack would require to exhaust all pairs of keys in $\mathcal{K} \times \mathcal{K}$, requiring $\mathcal{O}(|\mathcal{K}|^2)$ time. A meet-in-the-middle attack proceeds as follows.

First, in the pre-processing stage, the adversary computes all ciphertexts $\mathsf{Enc}(m, k)$ for all keys $k \in \mathcal{K}$ and stores the key in a table or list $\mathcal{L}_1$ indexed by the cipher text. Then, the adversary computes for all keys $k' \in \mathcal{K}$ all ciphertexts $\mathsf{Dec}(c, k')$ and stores the keys accordingly in $\mathcal{L}_2$. The adversary then enters the real-time phase and starts computing the list $\mathcal{L} \leftarrow \mathcal{L}_1 \diamond \mathcal{L}_2$. If $\mathcal{L}$ is non-empty, then each entry will be a pair of keys such that m encrypts to c. A simple complexity analysis shows that meet-in-the-middle attack requires $\mathcal{O}(|\mathcal{K}|)$ time and memory, which is a square root improvement over brute force.

1.8.5 DISTINGUISHING ATTACKS

All cryptographic attacks can be turned into decision problems. In the decisional setting, the adversary is given a sequence of symbols and must answer either Cipher if the intercepted sequence originates from a (known and fixed) cryptographic primitive, or Random if the sequence is distributed according to the uniform distribution. An adversary is considered to be successful if it is able to produce a correct answer with probability greater than $\frac{1}{2}$. Of course, a successful attack by no means implies a practical one. In fact, a distinguisher may require immense amounts of time and memory, which in some cases may cause it to be slower than brute force.

The idea of a distinguishing attack is based the assumption that some information leaks from the cryptosystem. For a public-key encryption scheme, *indistinguishability under chosen plaintext attack* (IND-CPA) is defined by a computational game between an adversary $\mathcal{A}$ and a challenger, as given in Game 1. In a cryptographic scheme based on computational security, the adversary $\mathcal{A}$ is a computationally bounded eavesdropper and modeled by a probabilistic polynomial-time algorithm. Since $\mathcal{A}$ is a PPT algorithm, it must output a guess of which message was encrypted, within a polynomial number of computation steps. We say that a public-key cryptosystem $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is secure in terms of indistinguishability if for all probabilistic polynomial-time algorithms $\mathcal{A}$ holds that $|\mathbb{P}(\mathcal{A} \text{ guesses correctly}) - \frac{1}{2}|$ is negligible in security parameter λ.

Game 1 (IND-CPA)

Challenger		**Adversary** $\mathcal{A}$
$\mathsf{KeyGen}(\lambda)$ generates (k_s, k_p);		
	$\xrightarrow{k_p}$	
		$\mathcal{M} \leftarrow \{m_0, m_1\}$;
	$\xleftarrow{\mathcal{M}}$	
Flips coin $b \xleftarrow{\$} \{0,1\}$;		
Computes $c \leftarrow \mathsf{Enc}(m_b, k_p)$;		
	$\xrightarrow{c}$	
		$\mathcal{A}$ makes a guess b';
	$\xleftarrow{b'}$	
if $b = b'$ **then** accept		
else reject;		

1.9 SUMMARY

In this chapter, we have provided a mathematical background to various theoretical concepts from fundamental probability, information theory and complexity theory. We have given the necessary details for hypothesis testing, and in particular Neyman-Pearson's lemma. We have also treated the birthday attack and how to approach it algoritmically. Finally, we have provided some background to cryptography and cryptanalysis, introduced the symmetric and public-key encryption in more detail and outlined different relevant attacks and attack settings for the two.

2

Linear Codes

I just wondered how things were put together.
– *Claude E. Shannon*

In this chapter, we will provide the reader with fundamentals of coding theory necessary for the sequel of this thesis. Essentially, coding theory concerns achieving and studying means of reliable and efficient information transmission over a noisy channel. Error-correcting codes have found direct applications in telecommunication, data storage and data compression, but also more subtle theoretical implications in cryptography, complexity theory, pseudo-randomness and combinatorics.

As a general coding-theoretical model, we assume that a sequence of symbols called message is transmitted over a noisy channel. With a non-zero probability, an arbitrary symbol will be inflicted by an error, and therefore the received message is likely to become corrupted to the receiver. To overcome this problem, the transmitted information will not only contain the message, but also include some redundancy based on the message symbols. For instance, the sender can transmit each symbol in the message several times to include information redundancy. Then the receiver may use a majority decision for each symbol to obtain the message. This simple encoding is called *repetition coding*.

In the realm of linear codes we find a wide range of code families that offer reliable and efficient information transmission. There is typically distinction between two types of codes: *block codes* encode information block by block, while *convolutional codes* encode a continuous stream of information bits.

2.1 GENERAL

We will now introduce some basics on linear codes. A *linear code*, which we denote $\mathcal{C}$, is a linear subspace. The elements of the code are called *codewords*.

Definition 2.1 (Linear code) An $[n,k]_q$ linear code $\mathcal{C}$ is a linear subspace over $\mathbb{F}_q$ of length n and dimension k.

The cardinality of $\mathcal{C}$ is $|\mathcal{C}| = q^k$ and the (information) *rate*, denoted R, of an $[n,k]_q$ linear code is $\frac{k}{n}$. We will sometimes omit the subscript and simply write $[n,k]$ linear code, which implicitly means that the code is binary, i.e., $q = 2$. A highly important property of a code is its *minimum distance*. We define it as follows.

Definition 2.2 (Minimum distance) The minimum distance of a linear code $\mathcal{C}$, denoted $d_{\min}$, is the minimum distance of its codewords

$$\begin{aligned} d_{\min} &= \min_{\mathbf{c},\mathbf{c}' \in \mathcal{C}} d_{\mathrm{H}}\left(\mathbf{c},\mathbf{c}'\right) \\ &= \min_{\mathbf{c},\mathbf{c}' \in \mathcal{C}} w_{\mathrm{H}}\left(\mathbf{c}-\mathbf{c}'\right) \\ &= \min_{\mathbf{c} \in \mathcal{C}, \mathbf{c} \neq \mathbf{0}} w_{\mathrm{H}}\left(\mathbf{c}\right). \end{aligned} \tag{2.1}$$

If $\mathcal{C}$ has minimum distance $d_{\min}$, then we say that $\mathcal{C}$ is an $[n,k,d_{\min}]_q$ linear code over $\mathbb{F}_q$. From the above, we see that the minimum distance is equal to the minimum of codeword weights in $\mathcal{C}$. A linear code $\mathcal{C}$ can be expressed as the image of a matrix, and as the kernel of another matrix:

Definition 2.3 (Generator matrix and parity-check matrix) Let $\mathcal{C} \subseteq \mathbb{F}_q^n$ be a linear code of dimension k. If $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ is a basis matrix of $\mathcal{C}$, i.e.,

$$\mathcal{C} = \left\{\mathbf{u}\mathbf{G} : \mathbf{u} \in \mathbb{F}_q^k\right\}, \tag{2.2}$$

then we say that $\mathbf{G}$ is a *generator matrix* for $\mathcal{C}$. Therefore, $\mathcal{C}$ has an encoding map $f : \mathbb{F}_q^k \to \mathbb{F}_q^n$, which is $\mathbf{u} \mapsto \mathbf{u}\mathbf{G}$.

If $\mathcal{C}$ is the kernel of a matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times k}$, i.e.,

$$\mathcal{C} = \ker(\mathbf{H}) = \left\{\mathbf{v} \in \mathbb{F}_q^n : \mathbf{H}\mathbf{v}^{\mathrm{T}} = \mathbf{0}\right\} \tag{2.3}$$

then we say that $\mathbf{H}$ is a *parity-check matrix* of $\mathcal{C}$. It follows that $\mathbf{G}\mathbf{H}^{\mathrm{T}} = \mathbf{0}$.

A basis matrix for a linear subspace is not unique, so a code $\mathcal{C}$ has many generator matrix representations. In particular, a generator matrix is said to be in *systematic form* if the first k columns of $\mathbf{G}$ form the $k \times k$ identity matrix.

For any systematic generator matrix $\mathbf{G}_{\text{sys}}$, each entry of the message vector appears among the entries of the codeword. Given the generator matrix in systematic form $\mathbf{G}_{\text{sys}} = \begin{pmatrix}\mathbf{I}_k & \mathbf{A}\end{pmatrix}$, the parity-check matrix in *canonical form* is $\mathbf{H} = \begin{pmatrix}-\mathbf{A}^{\mathrm{T}} & \mathbf{I}_{n-k}\end{pmatrix}$, where $\mathbf{I}_i$ is the $i \times i$ identity matrix. To see that this indeed is the case, we can form the product $\mathbf{G}_{\text{sys}}\mathbf{H}^{\mathrm{T}} = \mathbf{I}_k(-\mathbf{A}) + \mathbf{A}\mathbf{I}_{n-k} = \mathbf{0}$.

Definition 2.4 (Information set) Any subset $\mathcal{I} \subset \{1, 2, \ldots, n\}$ of the coordinate positions of a $[n, k]_q$ linear code is called an *information set* if there is a generator matrix for the code that is systematic on the columns in those positions, i.e., if the columns are linearly independent.

More informally, the positions of an information set carry the information, while the remaining positions contribute *redundancy* (sometimes, these symbols are called parity-check symbols). The amount of redundancy of $\mathcal{C}$ is $n - k$.

If a word $\mathbf{v} \in \mathbb{F}_q^n$ does not satisfy $\mathbf{H}\mathbf{v}^{\mathrm{T}} = \mathbf{0}$, then $\mathbf{v}$ is not a codeword in $\mathcal{C}$. By writing $\mathbf{v} = \mathbf{c} + \mathbf{e}$, with $\mathbf{c} \in \mathcal{C}$ and $\mathbf{e} \in \mathbb{F}_q^n$, we obtain

$$\mathbf{H}\mathbf{v}^{\mathrm{T}} = \mathbf{H}(\mathbf{c} + \mathbf{e})^{\mathrm{T}} = \mathbf{H}\mathbf{e}^{\mathrm{T}} \in \mathbb{F}_q^{1\times(n-k)}. \tag{2.4}$$

We call this the *syndrome* of the received word $\mathbf{v}$, being a linear combination of the columns of $\mathbf{H}$ corresponding to the error positions in $\mathbf{e}$. The syndrome, in some sense, is an indicator of error.

Definition 2.5 (Support) The *support* of a codeword $\mathbf{c} \in \mathcal{C}$ is the set of positions in which the non-zero elements appear and denoted $\operatorname{supp}(\mathbf{c})$.

Definition 2.6 (Coset) Let $\mathcal{C} \subseteq \mathbb{F}_q^n$ be a $[n, k]_q$ linear code. Moreover, let $\mathbf{x} \in \mathbb{F}_q^n$. Then, the subset

$$\mathbf{x} + \mathcal{C} = \{\mathbf{x} + \mathbf{c} : \mathbf{c} \in \mathcal{C}\} \tag{2.5}$$

is called a *coset* of $\mathcal{C}$. In particular, if $\mathbf{x} \in \mathcal{C}$, then $\mathbf{x} + \mathcal{C} = \mathcal{C}$.

Definition 2.7 (Dual code) The *dual code* of $\mathcal{C}$, which we will denote $\mathcal{C}^{\perp}$, is a linear subspace containing the words that are perpendicular to all codewords of $\mathcal{C}$, i.e.,

$$\mathcal{C}^{\perp} \stackrel{\text{def}}{=} \left\{\mathbf{v} \in \mathbb{F}_q^n : \langle \mathbf{v}, \mathbf{c} \rangle = 0,\ \forall c \in \mathcal{C}\right\} = \left\{\mathbf{v} \in \mathbb{F}_q^n : \mathbf{G}^{\mathrm{T}}\mathbf{v} = 0\right\}. \tag{2.6}$$

From above, we note that $\mathbf{G}^{\mathrm{T}}$ is a parity-check matrix of the code $\mathcal{C}^{\perp}$. Similarly, $\mathbf{H}^{\mathrm{T}}$ is a generator matrix of the dual code $\mathcal{C}^{\perp}$.

Two codes $\mathcal{C}$ and $\mathcal{C}'$ over $\mathbb{F}_q^n$ are called *permutation equivalent* provided there is a coordinate permutation f, which sends $\mathcal{C}$ to $\mathcal{C}'$, i.e.,

$$\mathcal{C}' = \{f(\mathbf{x}) : \mathbf{x} \in \mathcal{C}\}. \tag{2.7}$$

In particular, if f maps $\mathcal{C}$ to itself, then g is called an (permutation) automorphism. If there exists a permutation between $\mathcal{C}$ and $\mathcal{C}'$ with generator matrices $\mathbf{G}$ and $\mathbf{G}'$, respectively, then there must necessarily exist a non-singular matrix $\mathbf{S}$ and a permutation matrix $\mathbf{P}$ such that

$$\mathbf{G} = \mathbf{S}\mathbf{G}'\mathbf{P}. \tag{2.8}$$

If so, then the two generator matrices encode the same code (up to codeword symbol permutation); the matrix $\mathbf{S}$ causes the information sequence to codeword mapping to change, and the permutation matrix $\mathbf{P}$ permutes coordinates in all codewords.

Most code-based cryptography schemes use the property of code equivalence (and, in some sense, relies on the hardness of it; see e.g. [SS13] for further details on this matter), which usually involves coordinate permutations and linear transformations – among others, the McEliece public-key cryptosystem [McE78] and the CFS signature scheme [CFS01].

2.2 CHANNEL ERROR MODELS

Information transmission is a situation with presence of noise. To be able to understand and to analyze the noise factor and its impact on the transmission system, one usually approximates the noise with some well-defined and mathematically well-behaving model. Obviously, different situations require different models to obtain a suitable approximation of reality. Two such models that we will use is the *binary symmetric channel* and the *fixed-error model*.

Definition 2.8 (Binary symmetric channel) If the channel is discrete and memoryless with a constant crossover (bit error) probability $\rho \in [0, 1]$, we say that is a *binary symmetric channel*, denoted BSC_ρ.

The error of a binary symmetric channel BSC_p is Bernoulli distributed and can be interpreted as an arbitrarily long sequence $(X_i)_{t>0}$, where each random variable is i.i.d. and $X_i \sim \mathsf{Ber}_\rho$. It is commonly used in the setting of a »real-world« scenario, i.e., for transmission of radio signals or on an ethernet network.

In a more uncommon and highly artificial scenario, a constant and known number of errors occurs each time n information symbols are transmitted over a channel. In this thesis, we denote it as the *fixed-error model*. While the positions of the errors occur randomly, the quantity of the error is non-random. We define the fixed-error model as follows.

Definition 2.9 (Fixed-error model) If a n symbols are transmitted over a discrete channel and the total number of errors that occur is exactly $\rho \cdot n$, then we say that the channel is in the fixed-error model.

It is equivalent to say that the error sequence $\mathbf{e}$ is drawn uniformly from a Hamming sphere over $\mathbb{F}_q^n$ with radius $\rho \cdot n$, i.e.,

$$\mathbf{e} \xleftarrow{\$} \mathcal{S}_q(n, \rho \cdot n). \tag{2.9}$$

2.3 TYPES OF DECODING

Decoding comes in many flavors. In this section, we will briefly introduce different types of decoding approaches. The focus will be on unique decoding, i.e., when one desires a unique solution to the decoding problem. In unique decoding, the number of errors that an $[n, k, d_{\min}]_q$ linear code is able to correct is given by the *error-correction capability*

$$t \stackrel{\text{def}}{=} \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor. \tag{2.10}$$

MAXIMUM-LIKELIHOOD DECODING

A *maximum-likelihood decoding* is a central algorithmic error-correction problem in coding theory. Any algorithm that solves the problem is an optimal decoding algorithm.

Given a $[n, k]_q$ code $\mathcal{C}$ and a received word $\mathbf{r} \in \mathbb{F}_q^n$, a maximum-likelihood decoding procedure chooses the most likely codeword $\mathbf{c} \in \mathcal{C}$, i.e., it will find a solution to the maximization problem

$$\arg\max_{\mathbf{c} \in \mathcal{C}} \mathbb{P}\left(\mathbf{r} \text{ received} \mid \mathbf{c} \text{ sent}\right). \tag{2.11}$$

If all codewords are sent according to a uniformly random distribution, it allows for a reformulation of the maximization problem. Using Bayes' rule, we obtain

$$\begin{aligned} \mathbb{P}\left(\mathbf{r} \text{ received} \mid \mathbf{c} \text{ sent}\right) &= \frac{\mathbb{P}\left(\mathbf{r} \text{ received}, \mathbf{c} \text{ sent}\right)}{\mathbb{P}\left(\mathbf{c} \text{ sent}\right)} \\ &= \mathbb{P}\left(\mathbf{c} \text{ sent} \mid \mathbf{r} \text{ received}\right) \cdot \underbrace{\frac{\mathbb{P}\left(\mathbf{r} \text{ received}\right)}{\mathbb{P}\left(\mathbf{c} \text{ sent}\right)}}_{\text{constant}}, \end{aligned} \tag{2.12}$$

which in turn yields the maximization problem

$$\arg\max_{\mathbf{c} \in \mathcal{C}} \mathbb{P}\left(\mathbf{c} \text{ sent} \mid \mathbf{r} \text{ received}\right). \tag{2.13}$$

This reformulation of the maximum-likelihood decoding problem is called *ideal observer decoding*.

MINIMUM-DISTANCE DECODING

Closely akin to maximum-likelihood decoding, we have *minimum-distance decoding*, in which one searches for the nearest codeword in terms of Hamming metric.

A minimum-distance (or nearest neighbor) decoding procedure chooses the codeword $\mathbf{c} \in \mathcal{C}$ closest to the received word $\mathbf{r}$. More specifically, a minimum-distance decoding procedure solves the minimization problem

$$\arg\min_{\mathbf{c}\in\mathcal{C}} d_{\mathrm{H}}(\mathbf{r}, \mathbf{c}). \tag{2.14}$$

When the error model is that of a binary symmetric channel BSC_ρ with $\rho < \frac{1}{2}$, minimum-distance decoding is equivalent to maximum-likelihood decoding. This follows from that the probability-distribution function of the error

$$\mathbb{P}(\mathbf{r} \text{ received} \mid \mathbf{c} \text{ sent}) = (1-\rho)^{n-d} \cdot \rho^d \tag{2.15}$$

obtains its maximum when $d = d_{\mathrm{H}}(\mathbf{r}, \mathbf{c})$ is minimal.

SYNDROME DECODING

Syndrome decoding is an efficient method to solve minimum-distance decoding. Recall the definition of the syndrome. If $\mathbf{H}^{\mathrm{T}}\mathbf{r} = \mathbf{0}$, then $\mathbf{r} \in \mathcal{C}$ and no error within the correction radius t has occurred. On the other hand, if $\mathbf{H}^{\mathrm{T}}\mathbf{r}$ has non-zero weight, then it is compared with a list of pre-computed syndromes (usually stored in a look-up table for constant time access). If there is a match in the list, then that entry will hold the error vector. If no match is found, then the decoder concludes that there was more than t errors.

Given received word $\mathbf{r} = \mathbf{c} + \mathbf{e}$ where $\mathbf{c} \in \mathcal{C}$ and $\mathbf{e}, \mathbf{r} \in \mathbb{F}_q^n$, the syndrome is computed as follows

$$\begin{array}{rccc} f: & \mathbb{F}_q^n & \longrightarrow & \mathbb{F}_q^{n-k} \\ & \cup & & \cup \\ & \mathbf{r} & \longmapsto & \mathbf{H}^{\mathrm{T}}\mathbf{e}. \end{array} \tag{2.16}$$

For special codes, such as Hamming codes (see Subsection 2.6.3), no list of pre-computed syndromes is required. Instead, the error can be directly derived from the syndrome.

OTHER DECODING METHODS

List decoding is a decoding procedure which also solves minimum-distance decoding. Given a code $\mathcal{C}$ with error correction capability t, a received word

$\mathbf{r} \in \mathbb{F}_q^n$ and a parameter

$$m > t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor, \tag{2.17}$$

output a list of codewords from the code $\mathcal{C}$ that are located a distance at most m from the received word $\mathbf{r}$. By using a list decoding procedure, it allows the decoder to go beyond the error correction capability t. This is out of scope of this thesis and we will refrain from going deeper into the details.

In the subsequent chapters, we will consider *fixed-distance decoding*, which is the problem of decoding a block code for a fixed error rate, i.e., in the fixed-error model. This is particularly interesting in code-based cryptography.

2.4 RANDOM-CODING BOUNDS AND COVERING CODES

The probabilistic behavior of random codes will play a significant role in this thesis, in particular when we study the average case complexity of certain algorithms. In this section, we will give some results on random codes that are useful in the analysis. For more details, refer to [MS78] [CHL97].

First, we introduce two relevant problems in coding.

- **Packing problem**: given integers n and r, what is the maximum number of non-intersecting Hamming balls of radius r that can be placed in n-dimensional space?
- **Covering problem**: given integers n and l, what is the minimum number of Hamming balls of radius l that can be placed in to completely cover the n-dimensional space?

The packing problem is the basis of error correction; any two codewords in such a packing are at least distance $d = 2r + 1$ apart, meaning that at least r errors can be corrected. We have illustrated this in Figure 2.1.

Definition 2.10 (Hamming bound) The *Hamming bound* (or *sphere-packing bound*) states that if the code $\mathcal{C}$ is a $[n, k]_q$ linear code with minimum distance at least d, then the number of codewords satisfies

$$|\mathcal{C}| \leqslant q^n \cdot \left| \mathcal{B}_q \left(n, \left\lfloor \frac{d-1}{2} \right\rfloor \right) \right|^{-1}. \tag{2.18}$$

Proof. If the code $\mathcal{C}$ has a minimum distance at least d, then the code can correct up to $r = \lfloor \frac{d-1}{2} \rfloor$ errors. Every pair of Hamming balls are non-intersecting and each ball has size $|\mathcal{B}_q(n, r)|$. Taking the union over all balls, we obtain $|\mathcal{C}| \cdot |\mathcal{B}_q(n, r)| \leqslant q^n$. ■

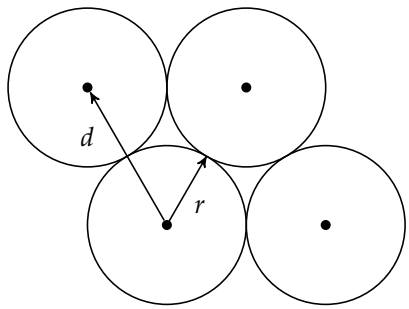


Figure 2.1: Sphere packing (rhombohedral).

The *covering radius* of a code of length n is defined as the smallest integer d_C such that all vectors in space $\mathbb{F}_q^n$ are within Hamming distance d_C of some codeword. The sphere-covering bound states how many such codewords that are needed to cover the whole space.

Theorem 2.1 (Sphere-covering bound) For every integer $q > 1$ and n, k such that $1 \leqslant k \leqslant n$ and covering distance d_C it holds that

$$|\mathcal{C}| \geqslant q^n \cdot |\mathcal{B}_q(n, d_C - 1)|^{-1}. \tag{2.19}$$

Later on, *covering codes* will be a useful tool in linear approximation in cryptanalysis. These codes have the property that any word is within some fixed distance l of a codeword in the code. More formally, we define a covering code as follows.

Definition 2.11 (d_C-covering code) Let $\mathcal{C}$ be a $[n, k]_q$ linear code and an integer $d_C \geqslant 0$, then $\mathcal{C}$ is called an d_C*-covering code* if there for any received word $\mathbf{r} \in \mathbb{F}_q^n$ is a codeword $\mathbf{c} \in \mathcal{C}$ such that $d_{\mathrm{H}}(\mathbf{r}, \mathbf{c}) \leqslant d_C$.

Theorem 2.2 (Gilbert-Varshamov bound) Let n, k and d be positive integers such that

$$|\mathcal{B}_q(n-1, d-2)| < q^{n-k}. \tag{2.20}$$

Then, there exists an $[n, k]$ linear code having minimum distance at least d.

The Gilbert-Varshamov bound ensures the existence of a q-ary linear code with a rate R linear code that has an minimum distance at least d. For instance, a binary linear code with rate $R = \frac{1}{2}$ has the GV bound $d = 0.11n$. This means that there exists at least one such code that is able to correct up to $\frac{1}{2} \cdot 0.11n$ errors. In fact, one can show that a random linear code attains the GV bound with high probability, implying that most random codes meet the GV bound.

2.5 COMPUTATIONAL PROBLEMS IN CODING

The vast majority of the computational problems in coding theory are inclined towards the decoding procedure (which seemingly is the most intractable), i.e., retrieving encoded information from a noisy observation. The *general decoding problem* can be stated as follows: given an encoding function $f : \mathcal{M} \to \mathcal{C}$ and an encoded message $f(m) \in \mathcal{C}$ that is inflicted by random noise, determine the original message $m \in \mathcal{M}$. In general decoding, the encoding function f can be an arbitrary mapping which not necessarily is linear.

In this thesis, we will concern only the linear setting, in which the (linear) encoding function generates a codeword

$$\mathbf{c} = f_{\mathbf{G}}(\mathbf{u}) = \mathbf{u}\mathbf{G}$$

that is perturbed by noise (according to some error model) when sent over a channel, and received on the other end as $\mathbf{r} = \mathbf{c} + \mathbf{e}$. The problem which the receiver will face consists of finding $\mathbf{c}$ under the constraint that $w_{\mathrm{H}}(\mathbf{e})$ is minimized. More formally, this problem is defined as follows.

Problem 2.3 (Minimum Distance Decoding**)** Let $\mathcal{C}$ be an $[n,k]_q$ linear code. Given a received word $\mathbf{r}$ and an integer w, find a codeword $\mathbf{c} \in \mathcal{C}$ such that $d_{\mathrm{H}}(\mathbf{r},\mathbf{c}) \leqslant w$. If no such codeword exists in $\mathcal{C}$, output $\perp$. We denote the problem MDD.

The MDD problem is intractable and inherently hard to solve in general. The decision problem version of MDD was proven **NP**-complete for binary linear codes by Berlekamp, McEliece and van Tilborg in [EBvT78] (and subsequently extended to q-ary linear codes by Barg in [Bar94]). The authors of [EBvT78] achieved the result by reduction to a well-known **NP**-complete problem called Three-Dimensional Matching. Since the groundbreaking work [EBvT78], similar hardness results have been published for many types of codes and settings. Not surprisingly, MDD can be (equivalently) formulated as a problem of syndrome decoding. We define it as follows.

Problem 2.4 (Computational Syndrome Decoding**)** Let $\mathcal{C}$ be an $[n,k]_q$ linear code. Given an $(n-k) \times k$ parity-check matrix $\mathbf{H}$ for $\mathcal{C}$, a syndrome $\mathbf{s} \in \mathbb{F}_q^{n-k}$ and an integer $w > 0$, find a word $\mathbf{e} \in \mathbb{F}_q^n$ such that $\mathbf{H}\mathbf{e}^{\mathrm{T}} = \mathbf{s}$ and $w_{\mathrm{H}}(\mathbf{e}) \leqslant w$. If no such word exists, output $\perp$. We denote the problem CSD.

Consequently, the decision problem version of CSD is also **NP**-complete (and, thus CSD as a search problem is **NP**-hard), and conjectured difficult in the average case [Sen11]. To illustrate an upper bound on the complexity of CSD, we consider a naïve (brute-force) algorithm that solves $\mathsf{CSD}(\mathbf{H}, w)$ by enumerating all possible solutions to $\mathbf{e}$ such that $w_{\mathrm{H}}(\mathbf{e}) \leqslant \alpha \cdot n$. This set of

vectors is contained in $\mathcal{B}_q(n, \alpha \cdot n)$, which according to Lemma 1.7 is of size $\sim q^{nh_q(\alpha)+o(n)}$. Since a brute-force algorithm exhausts the entire solution space, its running time is

$$T_{\mathsf{CSD\text{-}BF}}(n) = \mathcal{O}(q^{nh_q(w/n)+o(n)}), \tag{2.21}$$

and clearly exponential in n for a fixed ratio $\alpha = \frac{w}{n}$.

The best (known) average performance when solving arbitrary[1] instances of MDD and CSD obtained by using *information-set decoding* algorithms (discussed in Chapter 2). This quite narrow class of algorithms has, just like the brute-force algorithm we previously outlined, exponential-time complexity $T(n) = 2^{\mathcal{O}(n)}$ when the rate R is fixed and the error weight is parameterized (typically a linear function) in n.

Tightly related is the problem of finding a codeword in the code $\mathcal{C}$, such that it has a certain Hamming weight w, where w typically is very small (compared to the GV bound).

Problem 2.5 (Minimum Distance Problem[2]**)** Let $\mathcal{C}$ be an $[n, k, d_{\min}]_q$ linear code. Given an integer $w > 0$, find a codeword $\mathbf{c} \in \mathcal{C}$ such that $w_{\mathrm{H}}(\mathbf{c}) = w$. If no such codeword exists, output $\perp$. We denote the problem MDP.

As shown by Canteaut and Chabaud in [CC98], MDD can be reduced to MDP. Let $\mathcal{C}$ be an $[n, k, d_{\min}]_q$ linear code with generator matrix $\mathbf{G}$. Moreover, let $\mathbf{r} = \mathbf{c} + \mathbf{e}$ be the received word, where $\mathbf{c} \in \mathcal{C}$ and $\mathbf{e} \in \mathbb{F}_q^n$. Since a machine solving MDP must be able to terminate with a solution on any input, a new extended code $\mathcal{C}'$ generated by the extended generator matrix

$$\mathbf{G}' = \begin{pmatrix} \mathbf{G} \\ \mathbf{r} \end{pmatrix} \tag{2.22}$$

can be constructed. By definition, in order for the codeword $\mathbf{c}$ to be uniquely decodable, the error weight must satisfy $w_{\mathrm{H}}(\mathbf{e}) \leqslant t < \frac{1}{2} \cdot d_{\min}$, where t is the error-correction capability of the code $\mathcal{C}$. Hence,

$$\mathcal{C}' = \mathcal{C} \cup \{\mathbf{r} + \mathcal{C}\} \tag{2.23}$$

has minimum distance $w_{\mathrm{H}}(\mathbf{e})$ and therefore the minimum-weight codeword is $\mathbf{e}$. Solving MDP on inputs $(\mathcal{C}', j)$ for $0 \leqslant j \leqslant t$ will necessarily give the

[1] Assuming that the codeword weight or the error weight is below the Gilbert-Varhamov bound.

[2] In literature, this problem is sometimes referred to as Low-Weight Codeword or Subspace Weight.

error vector $\mathbf{e}$, after which the contribution from the error can be reverted by forming $\mathbf{r} - \mathbf{e}$.

A similar argument holds for the other direction in reducing MDP to MDD. Let $\mathcal{C}$ be an $[n, k]_q$ linear code generated by $\mathbf{G}$. Suppose that we want to find a codeword of weight t. Remove the ith row $\mathbf{g}_i$ from $\mathbf{G}$ forming $\mathbf{G}_i$. $\mathbf{G}_i$ generates the $[n, k-1]_q$ linear code $\mathcal{C}_i$. We can then solve MDD on input $(\mathcal{C}_i, \mathbf{g}_i)$. If the output $\mathbf{c}$ has weight t, then terminate. If not (the algorithm outputs $\perp$), choose another $0 < i \leqslant k$.

2.6 EFFICIENTLY DECODABLE FAMILIES

A family of codes is said to be *efficiently decodable* if there exists a PPT machine that can solve the MDD problem for all given instances in that particular encoding. Ever since the introduction of error-correcting codes in [Sha48], constructing efficiently decodable (and encodable) codes with good error-correction properties has been one of the major problems in coding theory. A lot of progress has been made in this area. For instance, it is known how to construct asymptotically good codes that are able to correct a constant fraction of errors, and for which decoding can be performed in polynomial or even linear time using expander constructions as in [Spi96] [GI01].

2.6.1 CONVOLUTIONAL CODES

Convolutional codes are a widely used family of error-correcting codes that allow for efficient decoding [JZ99]. This family differs from block codes in that the encoder has memory containing a number of previous symbols.

Encoding is done by a *sliding window* (this is the reason for the name convolutional codes) to calculate parity symbols by linear combinations of message symbols in the window. The length of the sliding window is called the *constraint length* or *memory* m of the code. Naturally, the larger the constraint length is, the more influence a single message symbol has on the parity bits, yielding better error-correction capability. However, (optimal) decoding time complexity scales exponentially with the constraint length, making it a trade-off between correction properties and decoding complexity.

Let b and c be positive integers. In encoding of a rate $R = \frac{b}{c}$ convolutional code $\mathcal{C}$, b-bit information symbols are transformed into c-bit symbols. The code rate R can be modified using *symbol puncturing*, i.e., excluding some parity symbols in the codeword[3]. In contrast to block codes, a convolutional code may be of infinite length.

[3]One could interpret code puncturing as a masking function ϕ being applied on every codeword such that $C' = \{\phi(\mathbf{c}) : \mathbf{c} \in \mathcal{C}\}$.

Definition 2.12 (Time-invariant convolutional code) A *time-invariant* or *fixed* convolutional code $\mathcal{C}$ has a generator matrix $\mathbf{G}$ such that

$$\mathbf{G} = \begin{pmatrix} \mathbf{G}_1 & \mathbf{G}_2 & \cdots & \mathbf{G}_m & & \\ & \mathbf{G}_1 & \mathbf{G}_2 & \cdots & \mathbf{G}_m & \\ & & \ddots & \ddots & & \ddots \end{pmatrix},$$

where $\mathbf{G}_i$ is a $b \times c$ matrix. The empty spaces of $\mathbf{G}$ are all-zero.

The output at any time is a linear combination of the current information symbol and the $m-1$ previous ones. The integer m is called the *memory* of the code. The sub-matrices $\mathbf{G}_i$ of each row are the same as the previous row, but shifted c steps. If the sub-matrices in any sense are different from previous row it is called a *time-variant* code, which we define as follows.

Definition 2.13 (Time-variant convolutional codes) A *time-variant* convolutional code $\mathcal{C}$ has a generator matrix $\mathbf{G}$ such that

$$\mathbf{G} = \begin{pmatrix} \mathbf{G}_{1,1} & \mathbf{G}_{1,2} & \cdots & \mathbf{G}_{1,m} & & \\ & \mathbf{G}_{2,1} & \mathbf{G}_{2,2} & \cdots & \mathbf{G}_{2,m} & \\ & & \ddots & \ddots & & \ddots \end{pmatrix}, \tag{2.24}$$

where all the $\mathbf{G}_{ij}$ matrices may be different.

Convolutional codes also allow for finite constructions by terminating the code in some way. One such terminated construction is a *tail-biting* code, i.e., the encoder will start in the same state as it will stop after encoding all information blocks.

Definition 2.14 (Tail-biting convolutional code) A *tail-biting* convolutional code $\mathcal{C}$ has a generator matrix $\mathbf{G}$ such that

$$\mathbf{G} = \begin{pmatrix} \mathbf{G}_{1,1} & \mathbf{G}_{1,2} & \cdots & \mathbf{G}_{1,m} & & \\ & \mathbf{G}_{2,1} & \ddots & & \ddots & \\ & & \ddots & & & \mathbf{G}_{i,m} \\ \mathbf{G}_{i+1,m} & & & \ddots & & \vdots \\ \vdots & \ddots & & & \ddots & \vdots \\ \mathbf{G}_{L,2} & \cdots & \mathbf{G}_{L,m} & & & \mathbf{G}_{L,1} \end{pmatrix}. \tag{2.25}$$

Optimal decoding of convolutional codes can be carried out using the Viterbi algorithm [Vit67] in time complexity $\mathcal{O}(2^{mb} \cdot k)$, where k is the number of information symbols being transmitted. Alternatively, more efficient but sub-optimal decoding can be achieved by *sequential decoding*, e.g., by the stack algorithm [JZ99].

2.6.2 CYCLIC CODES

Cyclic codes constitute an important family of linear block codes, first explored by Prange in [Pra57]. In particular, they are attractive since they achieve good error-correction capability. Many code constructions fall into the family of cyclic codes, thereamong BCH codes, Reed-Solomon codes, Euclidian geometry codes and quadratic residue codes. Refer to [LC04] for an excellent coverage of these code families.

Definition 2.15 (Cyclic code) Let $\mathcal{C}$ be a $[n, k, d_{\min}]_q$ linear code. If every circular shift of a codeword $\mathbf{c} = \begin{pmatrix} c_1 & c_2 & \cdots & c_n \end{pmatrix} \in \mathcal{C}$ again is a codeword $\mathbf{c}' = \begin{pmatrix} c_n & c_1 & \cdots & c_{n-1} \end{pmatrix} \in \mathcal{C}$, then we say that $\mathcal{C}$ is a cyclic code.

The description of cyclic codes may become more understandable if we regard them as polynomials, i.e.,

$$\begin{pmatrix} c_1 & c_2 & \cdots & c_n \end{pmatrix} \mapsto c_1 + c_2 x + \cdots c_n x^{n-1} \in \mathbb{F}_q[x]/\langle x^n - 1 \rangle. \tag{2.26}$$

If so, then every right-circular shift in codeword space corresponds to a multiplication by x in the polynomial counterpart, i.e.,

$$\begin{aligned} \begin{pmatrix} c_n & c_1 & \cdots & c_{n-1} \end{pmatrix} &\mapsto c_n + c_1 x + \cdots c_{n-1} x^{n-1} \\ &= x \cdot (c_1 + c_2 x + \cdots c_n x^{n-1}). \end{aligned}$$

While cyclic codes are cyclically symmetric [LC04], meaning that they behave exactly like polynomials, there are linear block codes that partially possess this property, called *quasi-cyclic codes*. We define a quasi-cyclic code as follows.

Definition 2.16 (Quasi-cyclic code) Let $\mathcal{C}$ be a $[n, k, d_{\min}]_q$ linear code. If each circular shift by $n_0 > 1$ coordinates of a codeword $\mathbf{c} = \begin{pmatrix} c_1 & c_2 & \cdots & c_n \end{pmatrix} \in \mathcal{C}$ again is a codeword $\mathbf{c}' = \begin{pmatrix} c_{n-n_0+1} & c_{n-n_0+2} & \cdots & c_{n-n_0} \end{pmatrix} \in \mathcal{C}$, then we say that $\mathcal{C}$ is a quasi-cyclic code of order n_0.

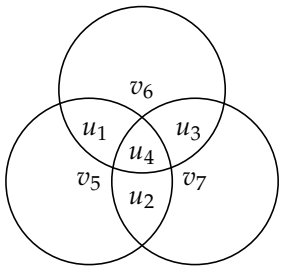
The integer n_0 is commonly called the *shifting constant*.

2.6.3 HAMMING CODES

The Hamming codes are a family of (binary) linear error-correcting (block) codes that are able to correct one error and able to detect up to two errors. Hamming codes are special in the sense that they are *perfect*[4], meaning that they attain the Hamming bound and thus achieve the highest possible rate for a certain block length and minimum distance $d_{\min} = 3$. For any integer $r > 1$, there is at least one Hamming code of length $n = 2^r - 1$ and dimension $k = 2^r - r - 1$.

[4]Hamming codes are, along with the trivial perfect codes and Golay codes, the only binary linear codes that attain the Hamming bound.

Example 2.1 ([7, 4] Hamming code) *A very simple way to construct a* $[7,4]$ *Hamming code is to use a Venn diagram that maps the information bits into parity bits.*



$$\begin{pmatrix} u_1 & u_2 & u_3 & u_4 \end{pmatrix} \mapsto \begin{pmatrix} u_1 & u_2 & u_3 & u_4 & v_5 & v_6 & v_7 \end{pmatrix}$$

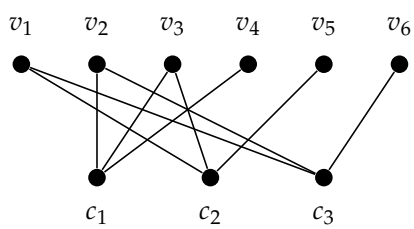
$$\mathbf{G} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

2.6.4 LDPC AND MDPC CODES

Low-density parity-check (LDPC) codes (also called Gallager codes) are a family of codes having very sparse parity-check matrices. LDPC codes are one of few codes that are capacity-approaching, which means they allow for the noise threshold close to the theoretical maximum for a symmetric memoryless channel.

Decoding of LDPC codes is achieved via iterative belief propagation on the *Tanner graph* representation of the code. A Tanner graph is a bipartite graphical representation of the code which contains a very small number of edges (each edge corresponds to a non-zero element in $\mathbf{H}$). Notably, probabilistic inference methods (such as iterative belief propagation) on sparse graphs is known to have linear complexity in the code length [Pea88].

Example 2.2 (Tanner graph) *A shortened* $[6,3]$ *Hamming code having parity-check matrix and corresponding Tanner graph representation as follows.*



$$\mathbf{H} = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Let us now consider the following: if an LDPC code is constructed with »slightly« relaxed weight constraints, one obtains a code which admits denser parity-check matrix than a normal LDPC code, but still sparser than those of classical block codes. This type of code is commonly referred to as *moderate density parity-check* (MDPC) codes. The performance degrades with the increased weight, but algorithms based on e.g. iterative belief propagation yield acceptable decoding performance.

Note: Both LDPC and MDPC codes may be cyclic or quasi-cyclic. In Chapter 5, we will explore some properties of quasi-cylic MDPC codes.

2.7 SUMMARY

In this chapter, we have given the basics of linear codes and provided a sufficient coding-theoretical foundation for the remaining chapters of this thesis. In particular, we have introduced three related computational problems in decoding, which are very relevant for the remaining part of the thesis. We have also introduced a few common code families that occur in the subsequent chapters.

3

Information-Set Decoding

All information look like noise until you break the code.
– *Neal Stephenson*

Decoding random linear codes is a fundamental problem in coding and complexity theory, but also in cryptography. In the previous chapter, we established that decoding is a very hard problem and it has been conjectured to be hard on average. We have seen a few examples of code families that have efficient decoding procedures, but such code constructions are very rare – most encodings are not known to have an efficient decoding procedure.

There are several known techniques for decoding random linear codes. The best[1] algorithms that we know of rely on information-set decoding (ISD), originally proposed by McEliece in [McE78] and inspired by the work [Pra62] of Prange. In short, the idea behind information-set decoding is to pick a sufficiently large set of error-free coordinates in a sent codeword such that the corresponding columns in the generator matrix form an invertible sub-matrix. Then, the information sequence can easily be obtained by linear algebra.

Since its introduction in [McE78], numerous different algorithmic techniques have been explored to improve complexity of information-set decoding. We will briefly outline some of the proposals in this chapter.

3.1 PLAIN ISD AND LEE-BRICKELL'S ALGORITHM

A *plain information-set decoding* algorithm randomly selects an information set and then computes the information by solving linear equations. Consider the

[1] When the error weight is below the Gilbert-Varshamov bound.

Algorithm 3 (Plain-ISD)

Input: Generator matrix $\mathbf{G} \in \mathbb{F}_2^{k \times n}$, received vector $\mathbf{r} \in \mathbb{F}_2^n$ and algorithm parameter $w \in \mathbb{N}$.

Output: Information vector $\mathbf{u} \in \mathbb{F}_2^k$.

```
while do
    Choose a subset I ←$ {1, 2, ..., n} and define φ_I(G) = (g_i^T)_{i∈I};
    Compute the information vector x ← φ_I(r)φ_I(G)^{-1};
    if w_H(r + xG) ⩽ w then
        return x
```

following. Let $\mathbf{u} \in \mathbb{F}_2^k$ be an unknown information sequence and

$$\mathbf{r} = \begin{pmatrix} r_1 & r_2 & \cdots & r_n \end{pmatrix} = \mathbf{u}\mathbf{G} + \mathbf{e} \in \mathbb{F}_2^n \tag{3.1}$$

a received codeword encoded by an $[n, k]$ linear code with generator matrix

$$\mathbf{G} = \begin{pmatrix} \mathbf{g}_1^{\mathrm{T}} & \mathbf{g}_2^{\mathrm{T}} & \cdots & \mathbf{g}_n^{\mathrm{T}} \end{pmatrix}. \tag{3.2}$$

Suppose we pick an information set $\mathcal{I} \subset \{1, 2, \ldots, n\}$ of size k. Recall that $\mathcal{I}$ is an information set if and only if the vectors $(\mathbf{g}_i)_{i \in \mathcal{I}}$ are all linearly independent. Now, define a masking function ϕ (see Definition 1.11) such that $\phi(\mathbf{G}) = (\mathbf{g}_i^{\mathrm{T}})_{i \in \mathcal{I}}$ and, thus,

$$\phi(\mathbf{r}) = (r_i)_{i \in \mathcal{I}} = (\mathbf{u}\mathbf{G} + \mathbf{e})_{i \in \mathcal{I}} = (\mathbf{u}\mathbf{G})_{i \in \mathcal{I}} + (\mathbf{e})_{i \in \mathcal{I}}. \tag{3.3}$$

As noted before, there is non-zero probability of symbols in the information set being perturbed by error, and therefore the procedure usually needs to be iterated with different information sets until it finds one that is entirely error free. Whenever the selected information set $\mathcal{I}$ is non-corrupted, i.e., $(\mathbf{e})_{i \in \mathcal{I}} = \mathbf{0} \iff \mathcal{I} \cap \operatorname{supp}(\mathbf{e}) = \varnothing$, then the information symbols can be obtained by computing

$$\phi(\mathbf{r})\phi(\mathbf{G})^{-1} = \phi(\mathbf{u}\mathbf{G})\phi(\mathbf{G})^{-1} = \mathbf{u}. \tag{3.4}$$

If, on the other hand, the information set contains a non-zero number of errors, then consequently $\phi(\mathbf{r})\phi(\mathbf{G})^{-1} \neq \mathbf{u}$, and therefore another different $\mathcal{I}$ must be chosen, until a solution satisfying the weight constraint is found. We summarize the procedure as given in Algorithm 3.

Let us now move on to analyze iteration success probability and algorithm complexity. Algorithm 3 succeeds to produce correct answer when it chooses an information set $\mathcal{I}$ disjoint from the support of $\mathbf{e}$.

Denote the iteration success probability $P_{\text{Plain-ISD}}$. Then,

$$\begin{aligned} P_{\text{Plain-ISD}}(n,k,w) &\stackrel{\text{def}}{=} \mathbb{P}\left(\mathcal{I} \cap \operatorname{supp}(\mathbf{e}) = \varnothing\right) \\ &= \frac{\mathcal{S}_q(n-k,w)}{\mathcal{S}_q(n,w)} = \binom{n-k}{w}\binom{n}{w}^{-1}, \end{aligned} \tag{3.5}$$

by simple counting argument; there are $\binom{n-k}{w}$ different patterns of the error positions, given that errors are absent in the $|\mathcal{I}| = k$ positions corresponding to the information set. Again, with no restriction, the number of error patterns is $\binom{n}{w}$.

The algorithm performs a series of independent iterations, where each iteration consists of random choices. Therefore, by Proposition 1.11, the expected running time is $P^{-1}_{\text{Plain-ISD}}$ iterations until Algorithm 3 succeeds with finding an error-free information set.

To obtain an expression for the complexity in terms of binary operations, we also need to include the amount of binary operations used in each iteration. The steps in Algorithm 3 are quite simple, and it suffices to compute the inverse of a matrix by Gaussian elimination. We denote the complexity of Gaussian elimination as $C_{\text{Gauss}}(n,k)$. Hence, the expected complexity of Plain-ISD, denoted $C_{\text{Plain-ISD}}(n,k,w)$, is

$$C_{\text{Plain-ISD}}(n,k,w) \stackrel{\text{def}}{=} C_{\text{Gauss}}(n,k) \cdot P^{-1}_{\text{Plain-ISD}}(n,k,w). \tag{3.6}$$

Note: By Lemma 1.4, choosing a uniform random matrix out of all binary matrices $\mathbb{F}_2^{k\times k}$ gives a non-singular matrix with probability at least $\frac{1}{4}$ (this probability is around 0.288). Although the probability remains constant with increasing k, it will add a constant factor around 4 in the random case to the overall complexity. This might not seem to be a problem – however, for certain classes of codes the columns may be highly biased, yielding a much larger additional factor. Therefore, Stern suggested in [Ste89] to choose the information set adaptively by picking the columns one at the time and subsequently performing row reduction before choosing the next column.

Note: Common practice is to set

$$C_{\text{Gauss}}(n,k) = \frac{(n-k)\cdot k^2}{2}, \tag{3.7}$$

although significant improvements are known. To remain consistent with literature, we will assume complexity according to (3.7) throughout this chapter.

In [LB88], Lee and Brickell explored a generalization of plain information-set decoding, allowing a set of $p \in \{0, 1, \ldots, w\}$ errors in the information. The parameter p is typically chosen to be a small number, and Peters proved in [Pet11] that $p = 2$ in fact is optimal in the binary case, improving the complexity by a factor $\Theta(n \cdot \log n)$ over plain information-set decoding ($p = 0$).

The Lee-Brickell algorithm enumerates the vectors in the Hamming sphere centered in $\mathbf{r}$, i.e., $\mathcal{S}_2(n, p) + \mathbf{r}$, and then checks if any of them is a codeword. In practice, this is accomplished by first computing $\mathbf{y} \leftarrow \phi(\mathbf{r})\phi(\mathbf{G})^{-1}\mathbf{G}$, and then for each size-p pattern $\mathcal{S} \subseteq \mathcal{I}$ compute $\mathbf{e} \leftarrow \mathbf{y} + \sum_{i \in \mathcal{S}} \mathsf{Row}(\mathbf{G}, i)$, i.e., $\mathbf{y}$ summed up with rows in positions indexed by $\mathcal{S}$. If $w_{\mathrm{H}}(\mathbf{e}) = w$, the correct codeword has been found. Assuming that the error $\mathbf{e}$ is uniformly random, the probability of having exactly p errors in the k-sized information set is

$$\mathbb{P}\left(|\mathcal{I} \cap \mathrm{supp}(\mathbf{e})| = p\right) = \binom{n-k}{w-p}\binom{k}{p}\binom{n}{w}^{-1}, \tag{3.8}$$

which also gives the probability of success in one iteration of the Lee-Brickell algorithm.

We will leave out asymptotic analysis of information-set decoding. For a thorough asymptotic complexity analysis of most algorithms presented in this chapter, see e.g. [Pet11] and [Meu12].

3.2 STERN'S ALGORITHM

Most modern information-set decoding algorithms of today are based on Stern's algorithm, which incorporates birthday attack methods to speed-up decoding (cf. 2-COL). This idea was first proposed by Stern in [Ste89]. A similar idea was proposed by Dumer in [Dum91], but we will refer to it as Stern's algorithm. The algorithm finds a weight-w codeword in a $[n, k]$ linear code $\mathcal{C}$, generated by $\mathbf{G} \in \mathbb{F}_2^{k \times n}$. Stern's algorithm allows, like the Lee-Brickell algorithm, a certain number of errors in the information set. Apart from the generator matrix $\mathbf{G}$, the algorithm takes as input parameters $p \in \{0, 1, \ldots, w\}$ and $l \in \{0, 1, \ldots, n-k\}$.

We will now go through the steps of the algorithm. First, two disjoint and equally sized subsets $\mathcal{I}_1$ and $\mathcal{I}_2$ are chosen at random, such that $\mathcal{I} = \mathcal{I}_1 \cup \mathcal{I}_2$ is an information set. We assume that $|\mathcal{I}_1| = |\mathcal{I}_2| = \frac{k}{2}$. The systematic generator matrix $\mathbf{G}_{\mathrm{sys}}$ is computed from $\mathbf{G}$ with respect to $\mathcal{I}$. The algorithm proceeds by picking a masking function ϕ defined from a randomly chosen set $\mathcal{Q} \subset \{1, 2, \ldots, n\} \setminus \mathcal{I}$ of size l.

The above steps are achieved by randomly choosing a column permutation

$$\pi : \mathbb{F}_2^{k \times n} \to \mathbb{F}_2^{k \times n} \tag{3.9}$$

and computing $\pi(\mathbf{G})$. Gaussian elimination are then performed on $\pi(\mathbf{G})$, to obtain $\mathbf{G}_{\text{sys}}$. $\mathcal{Q}$ is random, but since the columns already are randomly permuted, we can use a fixed masking function ϕ with the same result:

$$\phi(\mathbf{x}) = \begin{pmatrix} x_{k+1} & x_{k+2} & \cdots & x_{k+l} \end{pmatrix}. \tag{3.10}$$

In systematic form, the permuted generator matrix $\pi(\mathbf{G})$ is of the form

$$\mathbf{G}_{\text{sys}} = \begin{pmatrix} \mathbf{I} & \mathbf{Q} & \mathbf{J} \end{pmatrix}, \tag{3.11}$$

where $\mathbf{I}$ is the $k \times k$ identity matrix corresponding to the information set, $\mathbf{Q}$ is a $k \times l$ matrix corresponding to $\mathcal{Q}$, and $\mathbf{J}$ is a $k \times n-k-l$ matrix.

The algorithm proceeds by computing two lists $\mathcal{L}_1$ and $\mathcal{L}_2$, after which $\mathbf{u}$ runs through all weight-p vectors of length $\frac{k}{2}$, i.e., the Hamming sphere $\mathcal{S}_2(\frac{k}{2}, p)$ is computed. Then all vectors $\mathbf{x} = (\mathbf{u} \| \mathbf{0})\mathbf{G}_{\text{sys}}$ (for each $\mathbf{u}$) are stored in $\mathcal{L}_1$, sorted according to $\phi(\mathbf{x})$. Equivalently, a list $\mathcal{L}_2$ is constructed, sorted according to $\phi(\mathbf{x})$ and containing all vectors $\mathbf{x} = (\mathbf{0} \| \mathbf{u})\mathbf{G}_{\text{sys}}$ where again $\mathbf{u}$ runs through all weight-p vectors.

Finally, all pairwise sums of vectors $\mathbf{x} \in \mathcal{L}_1$ and $\mathbf{x}' \in \mathcal{L}_2$ that match in coordinates indexed by $\mathcal{Q}$ are computed, yielding the list

$$\mathcal{L} = \mathcal{L}_1 \diamond \mathcal{L}_2. \tag{3.12}$$

If there exists at least one codeword in $\mathcal{L}$ with weight at most w, the algorithm outputs the first one it finds (or a list of all) and terminates. The full procedure is described in Algorithm 4.

We will move on to analyze the complexity of Stern's algorithm. First, we analyze the success probability within one iteration. Stern's algorithm will produce a correct answer in the event that there are exactly p errors in $\mathcal{I}_1$ and p errors in $\mathcal{I}_2$, i.e., $|\mathcal{I}_1 \cap \text{supp}(\mathbf{e})| = p$ and $|\mathcal{I}_2 \cap \text{supp}(\mathbf{e})| = p$. Moreover, it must hold that $\mathcal{Q} \cap \text{supp}(\mathbf{e}) = \varnothing$. Then, the errors will be arranged according to Figure 3.1.

As before, we can sketch an expression for the probability of such an arrangement by a counting argument. There are $\binom{k/2}{p}$ different patterns in each half of the information set. In the l-symbol collision window there is only one pattern: the all-zero one. Finally, in the last $n-k-l$ we have $\binom{n-k-l}{w-2p}$ different patterns. Hence, based on the above argument, we can express the probability of success in one single iteration as

$$P_{\text{Stern-ISD}}(n,k,w,p,l) = \binom{k/2}{p}^2 \binom{n-k-l}{w-2p} \binom{n}{w}^{-1}, \tag{3.13}$$

Algorithm 4 (Stern-ISD)

Input: Generator matrix $\mathbf{G} \in \mathbb{F}_2^{k \times n}$ and algorithm parameters $w, p, l \in \mathbb{N}$.

Output: Codeword $\mathbf{c} \in \mathcal{C}$ of weight $w \in \mathbb{N}$.

1 **repeat**
2 Choose a random column permutation and form $\pi(\mathbf{G})$;
3 Bring the generator matrix $\pi(\mathbf{G})$ to systematic form:

$$\mathbf{G}_{\text{sys}} = \begin{pmatrix} \mathbf{I} & \mathbf{Q} & \mathbf{J} \end{pmatrix},$$

where $\mathbf{I}$ is the $k \times k$ identity matrix, $\mathbf{Q}$ is a $k \times l$ matrix and $\mathbf{J}$ is a $k \times n - k - l$ matrix;
4 Let $\mathbf{u}$ run through all weight-p vectors of length $\frac{k}{2}$. Store all vectors $\mathbf{x} = (\mathbf{u} \| \mathbf{0})\mathbf{G}_{\text{sys}}$ in a sorted list $\mathcal{L}_1$, sorted according to $\phi(\mathbf{x})$;
5 Then construct a list $\mathcal{L}_2$ sorted according to $\phi(\mathbf{x})$, containing all vectors $\mathbf{x} = (\mathbf{0} \| \mathbf{u})\mathbf{G}_{\text{sys}}$ where $\mathbf{u}$ runs through all weight-p vectors. Add all pairs of vectors $\mathbf{x} \in \mathcal{L}_1$ and $\mathbf{x}' \in \mathcal{L}_2$ for which $\phi(\mathbf{x}) = \phi(\mathbf{x}')$ and put in a list $\mathcal{L}$;
6 **until** *there is an* $\mathbf{x} \in \mathcal{L}$ *such that* $w_H(\mathbf{x}) = w - 2p$
7 **return** *the codeword* $\mathbf{c} \in \mathcal{C}$ *corresponding to* $\mathbf{x}$

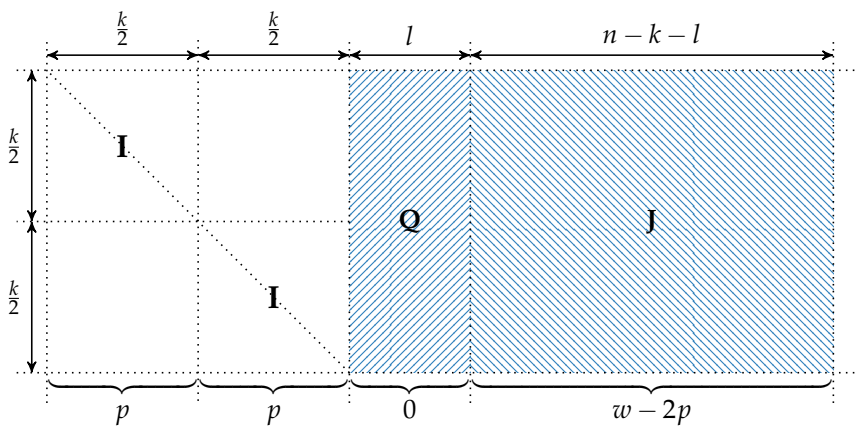


Figure 3.1: The different fields used in Stern's algorithm and their error distributions.

provided that $\mathcal{I}_1$, $\mathcal{I}_2$ and $\mathcal{Q}$ are picked at random. The iteration cost in terms of binary operations of Stern's algorithm, denoted $W_{\text{Stern-ISD}}(n, k, w, p, l)$, is

given by

$$C_{\mathsf{Gauss}}(n,k) + (n-k)\cdot(|\mathcal{L}_1| + |\mathcal{L}_2|) + (n-k-l)\cdot \mathbb{E}\left(|\mathcal{L}_1 \diamond \mathcal{L}_2|\right). \tag{3.14}$$

The size of the lists $\mathcal{L}_1$ and $\mathcal{L}_2$ is determined by code dimension k and algorithm parameter p, i.e.,

$$|\mathcal{L}_1| = |\mathcal{L}_2| = \binom{k/2}{p/2}, \tag{3.15}$$

and the expected size of the resulting merged list is given by

$$\mathbb{E}\left(|\mathcal{L}_1 \diamond \mathcal{L}_2|\right) = |\mathcal{L}_1| \cdot |\mathcal{L}_2| \cdot 2^{-l}. \tag{3.16}$$

according to Lemma 1.15. Again, by Proposition 1.11, the expected time complexity of running Stern's algorithm on instance parameters (n,k,w), denoted $C_{\mathsf{Stern\text{-}ISD}}(n,k,w)$, can be estimated to be

$$\min_{p,l}\left\{W_{\mathsf{Stern\text{-}ISD}}(n,k,w,p,l)\cdot P^{-1}_{\mathsf{Stern\text{-}ISD}}(n,k,w,p,l)\right\}. \tag{3.17}$$

Typically, the optimal value for the algorithm parameter l is

$$l = \log_2|\mathcal{L}_1| = \log_2|\mathcal{L}_2|. \tag{3.18}$$

Then, the expected list size is

$$\mathbb{E}\left(|\mathcal{L}_1 \diamond \mathcal{L}_2|\right) \approx |\mathcal{L}_1| = |\mathcal{L}_2|, \tag{3.19}$$

and therefore, the complexity of the final merging step will match the complexity of creating the initial lists $\mathcal{L}_1$ and $\mathcal{L}_2$.

CANTEAUT-CHABUAD

In [CC98], Canteaut and Chabaud investigated how to improve the Gaussian elimination step by re-using previous computations – a technique commonly referred to as *bit-swapping*. Initially, this approach was intended for use in Stern's algorithm, but applies to all information-set decoding algorithms.

Instead of picking a completely new information set at random in the beginning of each iteration of the information-set decoding algorithm, the authors of [CC98] suggest to only permute one or a few positions. As a result, the work performed in the Gaussian elimination step is reduced (roughly, by a factor k). However, iterations are not independent, leading to a different probability distribution. The probability mass function is rather involved, but can be estimated by the limiting distribution of Markov chains (see [CC98] for more details).

A generalization of the bit-swapping technique appears in [BLP11] and further investigation of the method appears in [Pet11].

FINIASZ-SENDRIER AND BALL-COLLISION DECODING

Instead of a fixed collision window, Finiasz and Sendrier explored in [FS09] the possibility of allowing the support of the error vector to spread across both the information set and the collision window, thereby increasing the success probability in one iteration. This is accomplished by computing the Hamming sphere over the coordinates in the set $\mathcal{I} \cup \mathcal{Q}$, rather than only $\mathcal{I}$.

In its original form, the Finiasz-Sendrier algorithm operates on the parity-check matrix $\mathbf{H}$, essentially meaning that it solves the CSD problem. Stern's algorithm, on the other hand, operates on the generator matrix $\mathbf{G}$ and solves the MDP problem. In principle, there is no difference in the two approaches – it is easy to transform Stern's algorithm into an algorithm solving the MDP problem. Consider the following. Let $\mathbf{G}_{\text{sys}}$ be a systematic and column-permuted generator matrix. Then, by operating on the rows of the matrix

$$\mathbf{G}' \stackrel{\text{def}}{=} \begin{pmatrix} & \mathbf{G}_{\text{sys}} & \\ \mathbf{0} & \mathbf{I}_l & \mathbf{0} \end{pmatrix} = \begin{pmatrix} \mathbf{I}_k & \mathbf{Q} & \mathbf{J} \\ \mathbf{0} & \mathbf{I}_l & \mathbf{0} \end{pmatrix} \in \mathbb{F}_2^{(k+l)\times n}, \tag{3.20}$$

we can solve MDP as efficiently as [FS09] solves CSD. The success probability in one iteration of Finiasz-Sendrier algorithm is

$$P_{\text{FS-ISD}}(n,k,w,p,l) = \binom{(k+l)/2}{p}^2 \binom{n-k-l}{w-2p} \binom{n}{w}^{-1}. \tag{3.21}$$

In the spirit of the Finiasz-Sendrier algorithm, Bernstein, Lange and Peters proposed a generalization of collision decoding, which they called *ball-collision decoding* [BLP11]. The essential idea behind ball-collision decoding is to allow a certain number of errors in the the collision fields. It was shown in [MMT11] that Finiasz-Sendrier algorithm is asymptotically at least as fast as ball-collision decoding.

3.3 A NEW ALGORITHM

We will now describe a new ISD algorithm that we proposed in [JL11]. Our algorithm extends Stern's algorithm beyond the birthday collision algorithm (which solves 2-COL), increasing the number of lists from two to several. In the original description, four lists were used, transforming the problem into 4-COL, which is solvable by the generalized birthday attack [Wag02].

A nearly identical algorithm was independently proposed by May, Meurer and Thomae in [MMT11][2]. Analogous to the algorithm by Finiasz and Sendrier

[2] After a correspondence with the authors of [MMT11], we decided not to publish our report beyond the scope of this thesis.

[FS09], the MMT algorithm solves CSD and, therefore, is applied on the canonical parity-check matrix $\mathbf{H}$, rather than on the generator matrix $\mathbf{G}$. The MMT algorithm takes advantage of the idea presented by Finiasz and Sendrier [FS09], allowing errors in the collision fields.

We briefly summarize the operation of our new algorithm as follows. Being an information-set decoding algorithm, it initially chooses an information set $\mathcal{I}$ of size k, and two disjoint sets $\mathcal{Q}_1$ and $\mathcal{Q}_2$ such that $\mathcal{Q}_1, \mathcal{Q}_2 \subset \{1, 2, \ldots, n\} \setminus \mathcal{I}$. The input generator matrix $\mathbf{G}$ is brought to systematic form with respect to $\mathcal{I}$, yielding $\mathbf{G}_{\text{sys}}$. Then four identical lists which we denote $\mathcal{L}$, each containing linear combinations of $\frac{p}{2}$ rows of $\mathbf{G}_{\text{sys}}$, are computed. Since every list is the same, it suffices to compute one of them and merge it by itself via a masking function $\phi_{\mathcal{Q}_1}$ to obtain the first level in the collision tree. To avoid triple subscripts, we simply denote this operation by $\diamond$. We write the merging of the lists as

$$\mathcal{L}_2 = \mathcal{L}_1 \diamond \mathcal{L}_1.$$

In the next step, again, we merge $\mathcal{L}_2$ by itself via the masking function $\phi_{\mathcal{Q}_2}$. To keep the notations of the two operations distinct, we denote it $\blacklozenge$, and write the last merging step as

$$\mathcal{L} = \mathcal{L}_2 \blacklozenge \mathcal{L}_2.$$

Finally, we examine the weight of the entries in $\mathcal{Q}_2$. If we find one that has weight w, we output that entry.

Again, without loss of generality, we can assume that the columns corresponding to the information set are permuted into the first k positions, followed by the columns indexed by $\mathcal{Q}_1$ and $\mathcal{Q}_2$. We then obtain a generator matrix $\mathbf{G}_{\text{sys}}$ in the following systematic form:

$$\mathbf{G}_{\text{sys}} = \begin{pmatrix} \mathbf{I} & \mathbf{Q}_1 & \mathbf{Q}_2 & \mathbf{J} \end{pmatrix}, \tag{3.22}$$

where $\mathbf{I}$ is the $k \times k$ identity matrix, $\mathbf{Q}_1$ is a $k \times z$ matrix, $\mathbf{Q}_2$ is a $k \times l$ matrix and $\mathbf{J}$ is a $k \times n - k - z - l$ matrix. Let $\phi_1(\mathbf{x}) \in \mathbb{F}_2^z$ be the value of $\mathbf{x}$ in the positions of $\mathbf{Q}_1$, i.e.,

$$\phi_1(\mathbf{x}) = \begin{pmatrix} x_{k+1} & x_{k+2} & \cdots & x_{k+z} \end{pmatrix}. \tag{3.23}$$

Similarly, let $\phi_2(\mathbf{x}) \in \mathbb{F}_2^l$ be the value of $\mathbf{x}$ in positions in $\mathbf{L}$, i.e.,

$$\phi_2(\mathbf{x}) = \begin{pmatrix} x_{k+z+1} & x_{k+z+2} & \cdots & x_{k+z+l} \end{pmatrix}. \tag{3.24}$$

We have illustrated the different fields used in the algorithm and their corresponding expected error weight in Figure 3.2.

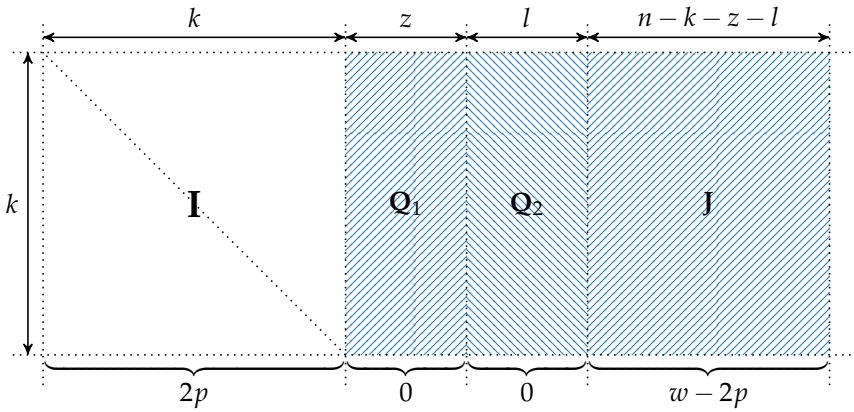


Figure 3.2: The different fields used in the new algorithm and their error distributions.

3.3.1 EXPLAINING THE IDEAS BEHIND THE ALGORITHM

First we note that the algorithm is similar to Stern's algorithm in its steps. Being a Las Vegas type algorithm, it makes a series of independent iterations where each iteration can be successful, i.e., succeed in finding the weight-w codeword.

Each iteration starts with selecting a random permutation of the columns in the generator matrix, and bringing it to systematic form. As for Stern's algorithm, this means that the non-zero entries in the low-weight codeword appear in randomly selected positions in the codeword. In an iteration, we require that in the first k positions (systematic part) of the desired codeword has weight $2p$, and in the following $z + l$ positions has weight 0. We can hope for a successful iteration, if this condition is fulfilled. Stern's algorithm has a similar (but not the same) condition.

The main improvement comes from introducing a second condition, which needs to hold for an iteration to be successful. A second necessary condition lowers the probability of a successful iteration, which is not to our benefit. However, it also lowers the number of active candidates in each iteration, hence reducing the computational complexity of each iteration.

The second condition can be explained as follows. Assuming that the first condition is valid, our desired codeword $\mathbf{c}_w$ of weight w has weight $2p$ in the first k positions, and in the following $z + l$ positions has weight 0. Then we can see that we will successfully find the codeword $\mathbf{c}_w$ in the iteration if and only if there are two words $\mathbf{x}, \mathbf{x}' \in \mathcal{L}_2$ such that $\mathbf{c}_w = \mathbf{x} + \mathbf{x}'$ and $\phi_1(\mathbf{x}) = \begin{pmatrix} 0 & 0 & \cdots & 0 \end{pmatrix}$. Since $\phi_1(\mathbf{c}_w) = \begin{pmatrix} 0 & 0 & \cdots & 0 \end{pmatrix}$ from our first condition, we

Algorithm 5 (JL-ISD)

Input: Generator matrix $\mathbf{G} \in \mathbb{F}_2^{k \times n}$ and algorithm parameters $w, p, z, l \in \mathbb{N}$.

Output: Codeword $\mathbf{c} \in \mathcal{C}$ of weight $w \in \mathbb{N}$.

1 **repeat**

2 Choose a column permutation and form $\pi(\mathbf{G})$, where π is a random column permutation and $\mathbf{G}$ is the given generator matrix;

3 Bring the generator matrix $\pi(\mathbf{G})$ to systematic form:

$$\mathbf{G}_{\text{sys}} = \begin{pmatrix} \mathbf{I} & \mathbf{Q}_1 & \mathbf{Q}_2 & \mathbf{J} \end{pmatrix},$$

where $\mathbf{I}$ is the $k \times k$ identity matrix, $\mathbf{Q}_1$ is a $k \times z$ matrix, $\mathbf{Q}_2$ is a $k \times l$ matrix and $\mathbf{J}$ is a $k \times n - z - l$;

4 Let $\mathbf{u}$ run through all weight-p vectors of length k. Store all vectors $\mathbf{x} = \mathbf{u}\mathbf{G}_{\text{sys}}$ such that $\phi_1(\mathbf{x}) = \begin{pmatrix} 0 & 0 & \cdots & 0 \end{pmatrix}$ in a sorted list $\mathcal{L}_2$, sorted according to $\phi_2(\mathbf{x})$. This is done by constructing a list $\mathcal{L}_1$ containing all vectors $\mathbf{x} = \mathbf{u}\mathbf{G}'$ where $\mathbf{u}$ runs through all vectors of weight $\frac{p}{2}$. Then add all pairs of vectors $\mathbf{x}, \mathbf{x}' \in \mathcal{L}_1$ in the list with $\phi_1(\mathbf{x}) = \phi_1(\mathbf{x}')$ and such that the largest index of the non-zero entries in $\mathbf{x}$ is smaller than the smallest index of non-zero entries in $\mathbf{x}'$;

5 As above, combine the list $\mathcal{L}_2$ with itself to receive a new list $\mathcal{L}$ of all codewords $\mathbf{x} = \mathbf{u}\mathbf{G}_{\text{sys}}$ with $\mathbf{u}$ of weight $2p$, such that $\phi_2(\mathbf{x}) = \begin{pmatrix} 0 & 0 & \cdots & 0 \end{pmatrix}$;

6 **until** *there is an* $\mathbf{x} \in \mathcal{L}$ *such that* $w_H(\mathbf{x}) = w - 2p$

7 **return** *the codeword* $\mathbf{c} \in \mathcal{C}$ *corresponding to* $\mathbf{x}$

have $\phi_1(\mathbf{x}') = \begin{pmatrix} 0 & 0 & \cdots & 0 \end{pmatrix}$. Hence, if $\mathbf{x}$ remains in the list $\mathcal{L}_2$, then so must $\mathbf{x}'$. With this in mind, we may now proceed with the complexity analysis of our new algorithm.

3.3.2 COMPLEXITY ANALYSIS

We derive formulas for the computational complexity similar to existing complexity formulas for the Stern's algorithm. Note that we consider the basic approach where some implementation-oriented improvements have not been made.

Since the proposed algorithm has a structure similar to Stern's algorithm, we can use the same approach to evaluate its computational complexity. The algorithm runs a number of iterations and a first requirement is that the error vector after the random permutation is such that there are $2p$ errors in the first

k positions, followed by zero errors in the $z+l$ following bits. We derive the probability, denoted P_1, for this event using a standard counting argument. This yields

$$P_1 \stackrel{\text{def}}{=} \binom{k}{2p}\binom{n-k-z-l}{w-2p}\binom{n}{w}^{-1}. \tag{3.25}$$

Let us now analyze amount of computation performed in the individual steps of the algorithm.

1. In the first step of an iteration, we apply a random permutation and transform the generator to systematic form. The complexity for this step, denoted C_1, is the same as for Stern's algorithm and is as in previous work set to
$$C_1 \stackrel{\text{def}}{=} C_{\mathsf{Gauss}}(n,k). \tag{3.26}$$

2. Next, we construct all codewords with weight $\frac{p}{2}$ in the information set. They are put in a list $\mathcal{L}_1$ sorted according to $\phi_1(\mathbf{x})$. The complexity of this step, denoted C_2 is
$$C_2 \stackrel{\text{def}}{=} |\mathcal{L}_1| \cdot c_1 = \binom{k}{p/2} \cdot c_1, \tag{3.27}$$
where c_1 denotes the actual work that has to be done to find $\phi_1(\mathbf{x})$ and then possibly storing more. The standard approach would set $c_1 = \frac{p}{2} \cdot z$ as we need to sum $\frac{p}{2}$ vectors over z bits to find out $\phi_1(\mathbf{x})$.

3. We now come to the first collision step. We build a new list $\mathcal{L}_2$ of all codewords with weight p on the information set such that $\phi_1(\mathbf{x}) = \mathbf{0}$, by adding pairwise all weight-$\frac{p}{2}$ vectors with the same $\phi_1(\mathbf{x})$. Assume that we only add weight-$\frac{p}{2}$ vectors that result in weight p in the first k bits. As every pairwise addition then gives a weight p vector such that $\phi_1(\mathbf{x}) = \mathbf{0}$ and we assume that only a fraction 2^{-z} of all weight-p vectors will have $\phi_1(\mathbf{x}) = \mathbf{0}$.

   Observation 3.1 There exist several pairs of vectors of weight $\frac{p}{2}$ that sum up to the same weight-p vector.

   In fact, we will have $\binom{p}{p/2}$ pairs that sum up to the same vector. Since we desire to pick only one among these, we add a condition which holds for one and only one pair. Suppose that for $\mathbf{x}_{i_1}, \mathbf{x}_{i_2}, \mathbf{x}_{i_3}, \mathbf{x}_{i_4}$, it holds that
$$\mathbf{x}_{i_1} + \mathbf{x}_{i_2} = \mathbf{x}_{i_3} + \mathbf{x}_{i_4}. \tag{3.28}$$

There are $\binom{4}{2}$ sums involving these four vectors, but matching them under the condition

$$i_1 < i_2 < i_3 < i_4, \tag{3.29}$$

there is only one such sum. Of course, this can be generalized to any p. In total, the collision step gives rise to complexity

$$C_3 \stackrel{\text{def}}{=} \mathbb{E}\left(|\mathcal{L}_1 \diamond \mathcal{L}_1|\right) \cdot c_2 = \binom{k}{p} \cdot 2^{-z} \cdot c_2, \tag{3.30}$$

where c_2 is the cost of computing the value of $\phi_2(\mathbf{x})$. The standard approach would set $c_2 = p \cdot l$ as we need to sum p vectors over l bits to find out $\phi_2(\mathbf{x})$.

4. The final step is the generation of a subset of all weight $2p$ vectors $\mathbf{x}$ such that $\phi_2(\mathbf{x}) = \mathbf{0}$. The array contains about $\binom{k}{p} \cdot 2^{-z}$ elements, and we sum together any two of them, $\mathbf{x}, \mathbf{x}'$, with the same $\phi_2(\mathbf{x}) = \phi_2(\mathbf{x}')$. As is argued in the analysis of Stern's algorithm, the total number of weight $2p$ vectors $\mathbf{x}'' = \mathbf{x} + \mathbf{x}'$ such that $\phi_2(\mathbf{x}'') = \mathbf{0}$ is then $(\binom{k}{p} \cdot 2^{-z})^2 \cdot 2^{-l}$. For each of them we need to check if its weight is w and we get the complexity for this part as

$$C_4 \stackrel{\text{def}}{=} \mathbb{E}\left(|\mathcal{L}_2 \blacklozenge \mathcal{L}_2|\right) \cdot c_2 \approx \left[\binom{k}{p} \cdot 2^{-z}\right]^2 \cdot 2^{-l} \cdot c_3, \tag{3.31}$$

where c_3 is the cost of deciding if a vector is the desired vector. Following previous work, the standard value for c_3 would be $c_3 = 2p \cdot (n-k)$.

The final issue is the following. Assuming that the desired low-weight codeword has $2p$ ones in the first k positions and zero ones in the next $z+l$ positions, what is the probability that this codeword is actually in the subset of the remaining codewords of weight $2p$ in the information set. In the process of building this subset there is one filtering stage. Namely, we keep only vectors with p ones in the first k positions such that $\phi_1(\mathbf{x}) = \mathbf{0}$.

If the desired codeword $\mathbf{x}''$ can be written as $\mathbf{x}'' = \mathbf{x} + \mathbf{x}'$ where $\phi_1(\mathbf{x}) = \mathbf{0}$, then it must hold that $\phi_1(\mathbf{x}') = \mathbf{0}$. This means that the desired codeword $\mathbf{x}''$ remains in the subset if there is an x such that $\phi_1(\mathbf{x}) = \mathbf{0}$.

As $\mathbf{x}''$ has $2p$ ones in first k positions corresponding to the information set, there are $\frac{1}{2} \cdot \binom{2p}{p}$ different ways to choose two vectors $\mathbf{x}, \mathbf{x}'$ of weight p in the first k positions. Note that the vectors in this step are not restricted by the condition in (3.29). This means that the probability that there exists at least one pair of vectors $\mathbf{x}, \mathbf{x}'$ with $\phi_1(\mathbf{x}) = \phi_1(\mathbf{x}') = \mathbf{0}$, denoted here by P_2, is

$$P_2 \stackrel{\text{def}}{=} 1 - \left(1 - 2^{-z}\right)^{\binom{2p}{p}/2}. \tag{3.32}$$

We may now put all the expressions together to get the overall average complexity for finding the low-weight codeword.

Theorem 3.2 The total complexity of Algorithm 5 is given by

$$\frac{C_{\mathsf{Gauss}}(n,k) + \binom{k}{p/2} \cdot pz/2 + \binom{k}{p} 2^{-z} \cdot pl + \left(\binom{k}{p} 2^{-z}\right)^2 2^{-l} \cdot 2p(n-k)}{\left(\binom{k}{2p}\binom{n-k-z-l}{w-2p}\binom{n}{w}^{-1}\right) \cdot \left(1 - (1-2^{-z})^{\binom{2p}{p}/2}\right)}. \tag{3.33}$$

Proof. Putting together the parts we obtained, in conjunction with the result in Proposition 1.11, we get

$$\mathbb{E}\,(\#\text{operations}) = \frac{C_1 + C_2 + C_3 + C_4}{P_1 \cdot P_2}, \tag{3.34}$$

which yields (3.33). ■

3.3.3 PARAMETERS AND SECURITY

Various examples of parameters were given in [BLP11] to obtain certain security levels. New bit-security levels of our new algorithm applied the given parameters are presented in Table 3.1.

As for the case $(6624, 5129, 117)$, that is recommended for 256-bit security, we solve it with on average $2^{247.29}$ operations with $p = 12$, $z = 52$ and $l = 74$. Moreover, for the 1000-bit security level with parameters $(30332, 22968, 494)$, we obtain a much better complexity of $2^{976.55}$ operations using our algorithm with $p = 28$, $z = 53$ and $l = 264$.

Instance			Complexity ($\log_2$)		
n	k	w	Stern	Algorithm 5	Ball-collision*
1632	1269	34	82.23	78.91	81.33
2960	2288	57	129.84	125.06	127.89
6624	5129	117	258.61	247.29	254.15
30332	22968	494	1007.4	976.55	996.22

Table 3.1: Choice of parameters and their security with corresponding algorithm. Entries marked with * are taken from [BLP11].

Becker *et al.* introduced a refined information-set decoding technique in [BJMM12], further lowering the upper bound on asymptotic complexity. We will not give the details here, but we will provide a complexity comparison

between different algorithms. In [HS13], Hamdaoui and Sendrier investigated the non-asymptotic complexity of the algorithms presented in [MMT11] and [BJMM12]. In contrast to our bit-oriented complexity analysis, the authors of [HS13] measured complexity in terms of column operations. Converting the complexity expressions into row-oriented ones (loosing approximately a factor $z + l$), we obtain figures comparable with those of [HS13]. The results are given in Table 3.2.

Instance			Complexity ($\log_2$)				
n	k	w	LB	Stern	Alg. 5	MMT*	BJMM*
1024	524	50	62.51	58.46	55.71	54.29	52.50
2048	1696	32	91.07	84.37	79.87	79.32	75.78
4096	3844	21	93.73	84.42	78.33	78.11	74.34
4096	3616	40	134.20	124.67	117.68	118.05	114.62
8192	7945	19	105.33	92.71	85.19	85.87	82.58
8192	7815	29	138.70	125.85	116.46	118.67	115.65
9600	4800	84	98.75	91.65	90.17	87.75	85.82
22272	14848	85	149.87	140.61	138.71	137.07	134.78

Table 3.2: Complexities of different algorithms measured in column (row) operations. The entries marked with * are taken from [HS13] and measured in column operations.

For some of the instances in Table 3.2, Algorithm 5 performs seemingly better than MMT. We believe that this is due to the assumptions made by the authors of [HS13] to simplify analysis. It is reasonable to believe that MMT should be slightly more efficient, using the technique introduced in [FS09]. Of course, this technique can be used in the presented algorithm, spreading the p errors along both collision fields. For instance, we can construct a matrix such that

$$\mathbf{G}' \stackrel{\text{def}}{=} \begin{pmatrix} \mathbf{I}_k & \mathbf{Q}_1 & \mathbf{Q}_2 & \mathbf{J} \\ \mathbf{0} & \mathbf{I}_z & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_l & \mathbf{0} \end{pmatrix} \in \mathbb{F}_2^{(k+z+l)\times n}, \tag{3.35}$$

and apply Algorithm 5 on $\mathbf{G}'$ to find a low-weight codeword (and at the same time make sure that at least one row of the first k rows are present in the linear combination). There are obvious implementation improvements that can be done to this approach, but it is out of scope for this thesis.

As a final remark, we note that the complexity expressions given in this chapter are valid when there is only one solution. When several codewords of weight w are be present, and provided that any of the low-weight vectors

is an acceptable solution, we can use the following lemma to obtain a more accurate expression for algorithmic complexity.

Lemma 3.3 (Codeword multiplicity factor) Let $\mathcal{C}$ be an $[n, k]$ random code with a subset $\mathcal{C}_w \subseteq \mathcal{C}$ of weight-w codewords. Then, an arbitrary ISD algorithm solves $\mathsf{MDP}(\mathcal{C}, w)$ with complexity $C_{\mathsf{ISD}} \cdot |\mathcal{C}_w|^{-1}$.

Proof. Since $\mathcal{C}$ is random, all codewords in $\mathcal{C}_w$ are independent. Assume that the ISD algorithm finds a solution with probability ξ. It suffices to find a single solution, so the probability of succeeding is that of finding at least one out of $|\mathcal{C}_w|$ codewords is

$$1 - (1 - \xi)^{|\mathcal{C}_w|} \approx |\mathcal{C}_w| \cdot \xi,$$

since all codewords are equally likely to be found. The complexity $C_{\mathsf{ISD}}(, n, k, w)$ is $\mathcal{O}\left(\xi^{-1}\right)$ and therefore $C_{\mathsf{ISD}} \cdot |\mathcal{C}_w|^{-1}$.

■

3.4 SUMMARY

In this chapter, we treated different information-set decoding algorithm used for fixed-distance decoding. Then, we have presented one of the contributions of this thesis – a new information-set decoding algorithm. We have shown that generalized birthday techniques are also applicable with good results in information-set decoding, also for instances with a single solution (when the weight w is below the Gilbert-Varshamov bound). We have provided results showing that the proposed algorithm is more efficient than the state-of-the-art algorithms at the time.

4

Code-based cryptography

Door: "Why it's simply impassible!"
Alice: "Why, don't you mean impossible?"
Door: "No, I do mean impassible. Nothing's impossible!"
– *Lewis Carroll, Alice's Adventures in Wonderland*

In public-key cryptography there are several different cryptosystems based on different hard problems. Integer factorization and discrete logarithms are among the most common and most trusted problems used. But many other problems, like knapsack problems, hard coding theory problems, lattice reduction problems to name a few, are also intensively studied. One motivation for this is that a large future quantum computer, if ever constructed, will be able to solve factorization and discrete logarithms in polynomial time [Sho94], whereas other problems might remain more difficult. For a more detailed discussion on this, see [Ber09]. A second motivation is that in constrained environments we might see a sufficient performance gain when we use some alternative cryptosystem compared to common ones (like RSA, El Gamal) based on factorization or discrete logarithms.

4.1 MCELIECE CRYPTOSYSTEM

The original McEliece construction as proposed by McEliece in [McE78] is an asymmetric encryption scheme using a family of error correcting codes. McEliece proposed a construction based on Goppa codes, and this original construction remains unbroken today. A few years later, Niederreiter [Nie86] proposed a different scheme and proposed to use generalized Reed-Solomon codes. It can be shown that if one uses Goppa codes then Neiderreiter PKC

is equivalent to McEliece PKC. It was also shown by Sidelnikov and Shestakov [SS92] that the choice of generalized Reed-Solomon codes is insecure (in both cases). There has been many proposals of modified schemes, mostly by replacing the Goppa codes by another family of codes, e.g., LDPC codes [MRS09] or codes for the rank metric [Del78]. Interestingly, most of these proposals have turned out to be insecure and the choice of Goppa codes is still the standard solution.

A motivating factor for studying McEliece PKC is that the cryptosystem is a candidate for post-quantum cryptography, as it is not known to be susceptible to attacks using quantum computers. There have also been modifications of McEliece PKC with proved formal security (CCA2-secure), some of which are presented in [EOS07]. Attempts on improving the rate of the system by using a subset of the message bits as error vectors have been done. Some approaches appear in [Sun98]. We should also note that there have been many attempts to build other public-key primitives based on coding theory, e.g., the signature scheme CFS proposed by Courtois, Finiasz and Sendrier in [CFS01] and the hash function FSB proposed by Augot, Finiasz and Sendrier in [AFS05].

Several versions of McEliece, for example, using quasi-cyclic or quasi-dyadic codes have been attacked in structural attacks. Faugère *et al.* [FOPT10] give the basic setting of structural attacks using *algebraic attacks*. In this approach, the problem of reconstructing the secret matrix is formulated as the problem of solving a specific over-defined algebraic system, and it applies to any alternant code (among them Goppa codes). Even though the attack is currently not successful against Goppa codes, as the system is too difficult to solve, we do not know what improvements will come in the future.

Definition 4.1 (McEliece cryptosystem) The McEliece cryptosystem is a triple $\Delta = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ such that

1. Let $\mathcal{C}$ be randomly chosen from a family of efficiently decodable linear codes. Let $\mathbf{G}$ be a generator matrix for $\mathcal{C}$. Furthermore, let $\mathbf{S}$ be a random and non-singular matrix and $\mathbf{P}$ a permutation matrix. Key-generation is

$$(k_p, k_s, w) = (\mathbf{SGP}, \mathbf{G}, w) \leftarrow \mathsf{KeyGen}(\lambda),$$

   where $k_s \leftarrow \mathbf{G}$ with $\mathbf{G}$ a generator matrix for an efficiently decodable code $\mathcal{C}$ with error-correction capability $t \geqslant w$.

2. The encryption function $\mathbf{c} \leftarrow \mathsf{Enc}(\mathbf{m}, \mathbf{SGP}) = \mathbf{m}(\mathbf{SGP}) + \mathbf{e}$ models information transmission under a fixed-error model, so $\mathbf{e} \xleftarrow{\$} \mathcal{B}_w(n, q)$ for some integer w depending on λ.

3. An efficient decoding algorithm solves the decoding problem instance $(\mathbf{G}, \mathbf{uG} + \mathbf{e})$ in polynomial time as guaranteed by the code family.

4. $\mathsf{Dec}(\mathbf{c}, k_s)$ outputs the message $\mathbf{m}$.

The security of McEliece PKC is based on the hardness of decoding a random linear code in the fixed-error model. We established in Section 2.5 that MDP is an **NP**-hard and, thus, intractable problem. The core idea is to transform a special and polynomially solvable instance into something that looks like a random instance. Not knowing how to perform the inverse transformation, the attacker will face a presumably hard problem:

$$\mathbf{c} = \overbrace{\mathbf{m}\hat{\mathbf{G}} + \mathbf{e}}^{\text{hard}} = \mathbf{m}(\mathbf{SGP}) + \mathbf{e} = (\mathbf{mS})\mathbf{GP} + \mathbf{e}. \tag{4.1}$$

Knowing the trapdoor, i.e., $\mathbf{S}$ and $\mathbf{P}$, the presumably hard instance can be turned into one that is efficiently decodable:

$$\mathbf{cP}^{-1} = ((\mathbf{mS})\mathbf{GP} + \mathbf{e})\mathbf{P}^{-1} = (\mathbf{mS})\mathbf{G} + \mathbf{eP}^{-1} = \underbrace{\hat{\mathbf{m}}\mathbf{G} + \hat{\mathbf{e}}}_{\text{easy}}. \tag{4.2}$$

The hardness of inverting the encryption function Enc is based on an assumption that it maps the set of messages into a set of instances of the intractable problem that are hard on average. Alas, it has shown that many proposed code constructions used in McEliece admit an unavoidable structural characteristic, which in the end has lead to a partial or total breakdown of security.

4.1.1 THE ORIGINAL MCELIECE CONSTRUCTION

Let us start by giving a short overview of the original McEliece construction of a public-key encryption scheme.

Let $\mathbf{G}$ be a $k \times n$ generator matrix for a code $\mathcal{C}$ capable of correcting e errors, P a $n \times n$ random permutation matrix and S a $k \times k$ non-singular matrix. We require an efficient decoding algorithm associated with $\mathcal{C}$. The sizes of these matrices are public parameters, but $\mathbf{G}$, $\mathbf{P}$ and $\mathbf{S}$ are randomly generated secret parameters. Furthermore, $\mathbf{G}$ is selected from a large set of possible generator matrices, say $\mathbf{G} \in \mathcal{G}$, where the generator matrices in $\mathcal{G}$ all allow a very computationally efficient decoding procedure. Also, $\mathbf{P}$ is selected among all permutations and $\mathbf{S}$ is selected among all non-singular matrices. Then, pre-processing, encryption and decryption can be described in the following three steps.

A key issue is the selection of a set of easily decodable codes, from which we select $\mathbf{G}$. The original suggestion was to use all binary Goppa codes for

some fixed code parameters. This is a large enough set of possible codes and they can all be decoded assuming a fixed number of errors.

In the paper by McEliece [McE78], parameters $(n, k, e) = (1024, 524, 50)$ were proposed. These parameters, however, do not attain the promised security level in [McE78] due to advances in attacks.

4.2 ATTACKS IN CODE-BASED CRYPTOGRAPHY

We consider two types of attacks in code-based cryptography:

- **Message recovery**: Recovery of the message from a ciphertext is best done (if no structure can be exploited) by information-set decoding algorithms (see Chapter 3), by finding a minimum-weight codeword in the coset $\mathbf{c} + \mathcal{C}$. As we mentioned in Section 2.5, this can be accomplished by constructing an extended code with the generator matrix

$$\mathbf{G}' = \begin{pmatrix} \mathbf{G} \\ \mathbf{c} \end{pmatrix}.$$

  Then, an information-set decoding algorithm may be applied to find a minimum-weight codeword.

- **Key recovery**: The purpose of a key recovery (structural attack) is to recover the private key from the public key. Many codes used in information transmission have a structure that allows for a very efficient decoding procedure. In that context, having a structured code does not pose a problem. However, in code-based cryptography such as McEliece, the security of the cryptosystem relies on the assumption that the public-key generator matrix is indistinguishable from a random one. So, when structured codes are used, the security will suffer because of the unavoidable structure of the public key. One such example of applied and subsequently broken codes is the family of Reed-Solomon codes, which is a family of algebraic codes that have a very well-defined structure. This was first noted in [SS92].

4.3 NEW VARIANTS OF MCELIECE CRYPTOSYSTEM

A problem with the original McEliece construction is the size of the keys. McEliece proposed to use the generator matrix of a linear code as public key. The public key for the originally proposed parameters is roughly 500 Kbits and roughly 300 Kbits for the private key. Although this is very small in perspective to today's storage capacities, it has motivated various attempts to

decrease key size. For instance, it is in many ways tempting to adopt LDPC codes [MRS09], where the parity-check matrix is very sparse. In terms of error correction, this property is desirable because it allows a very efficient decoding and small keys. However, the low-weight parities can be easily found using standard techniques, exposing the cryptosystem to total break attacks. To resolve this problem, [BCGM07] proposed to replace the permutation matrix **P** by an invertible transformation matrix **Q**. As a consequence, the weights of the parity checks in **G** are increased, making the problem of finding low-weight parities much harder.

4.3.1 MCELIECE BASED ON CONVOLUTIONAL CODES

This section presents a new version of the McEliece PKC that uses convolutional codes, originally proposed in [LJ12]. The motivation for this work was a project given to the author in which we should investigate the possibility of using convolutional codes in constructing a McEliece cryptosystem or similar.

The first construction is based on tail-biting time-varying convolutional codes. The second construction uses a block code to set the starting state for the convolutional code. A large part of the code is constructed by randomly generating parity checks for a systematic convolutional code. The new proposal allows for flexible parameters and efficient decoding. The drawback is that we have a non-zero probability of not being successful in decoding, in which case we need to ask for a retransmission. In opposite to the choice of Goppa codes, the first proposed construction uses large parts of randomly generated parity checks, and presumably, this makes structured attacks more difficult. This is the main contribution of the construction. All parity checks are randomly generated and of low weight, but not too low. Algebraic attacks applied to our system are unlikely to be successful, as well as attacks using Sendrier's support splitting algorithm [Sen00].

THE NEW CONSTRUCTION

Our basic idea is that we would like to replace the Goppa code used in McEliece by a convolutional code that can be efficiently decoded. However, this basic approach suffers from two problems that we need to deal with.

- A usual convolutional code as used for error correcting purposes has a somewhat limited memory. This is necessary if we would like to have optimal (maximum-likelihood) decoding through, e.g., the Viterbi algorithm (see e.g. [JZ99]). As an example, a rate $R = \frac{1}{2}$ convolutional code with a memory of 20 would mean that we will have parity checks of low weight. This would be a security problem, as one can try to

recover the low-weight parity checks and through this also recover the structure of the secret code.

The solution we propose here is to use convolutional codes with large memory, but to use a sequential decoding procedure instead of Viterbi decoding. Using, e.g., the stack algorithm we can use codes with much larger memory. This will remove the very low-weight parities in the code. Still leaving low-weight parity checks, but if the weight is not very small, the complexity of finding them will be too large.

- A convolutional code usually starts in the all-zero state. This means that the first code symbols that are produced will again allow very low-weight parity checks, giving security problems. As we cannot allow the identification of low-weight parity checks, we need to modify this. A similar problem may occur when we terminate the code.

  The solution that we propose is to use tail-biting to terminate the code, giving tail-biting trellises in decoding. This would work quite straightforward if we had a small trellis size and were using Viterbi decoding. But for a large memory size and sequential decoding it is not immediate how tail-biting could be used, as one has to go through all possible states at some starting point in the circular trellis. Another approach to solve this problem is to use a block code to set the starting state. We have also examined this approach.

THE MCELIECE PKC BASED ON A TAIL-BITING CONVOLUTIONAL CODE

The scheme works as usual McEliece, with a different way of generating the secret matrix $\mathbf{G}$ as well as a different decoding procedure for the decryption process. The secret generator matrix will have the characteristics of the one appearing in (2.25), i.e., it has a cyclic diagonal structure with m matrices $\mathbf{G}_{i,j}$ in each column and row. We also set the generator matrix $\mathbf{G}$ to be systematic, see Algorithm 6 for a detailed description. We have illustrated the code structure in Figure 4.1.

Assuming that we receive a word $\mathbf{c}$ with e errors, how do we correct them, knowing $\mathbf{G}$? A problem when decoding this structure is that we have to guess a starting state at some time instance and then perform the sequential decoding. With many possible starting states this increases the complexity of decoding. We assume that we use the stack algorithm as decoding algorithm and put a number of possible starting states on the stack in the beginning. An advantage is that this part of the decoding can be easily parallelized by starting decoding at different time instances.

We now describe a basic approach for decoding. Compute $\hat{\mathbf{c}} = \mathbf{c}\mathbf{P}^{-1}$. Start the decoding at any time t. On the stack we then put a set $\mathcal{V}$ of possible

Algorithm 6 (KeyGen)

Input: Algorithm parameters $n, m, b, c, l \in \mathbb{N}$.

Output: Public-key matrix $\hat{\mathbf{G}} \in \mathbb{F}_2^{(nb/c)\times n}$.

1 Write up a generator matrix as (2.25) in systematic form, i.e., choose $\mathbf{G}_{i,1} = \begin{pmatrix}\mathbf{I} & \mathbf{J}_{i,1}\end{pmatrix}$ and $\mathbf{G}_{i,j} = \begin{pmatrix}\mathbf{0} & \mathbf{J}_{i,j}\end{pmatrix}$, for $2 \leqslant j \leqslant m$, where $\mathbf{J}_{i,1}$ is chosen to give the code spanned by $\mathbf{G}_{i,1}$ maximal minimum distance and $\mathbf{J}_{i,j}$, for $2 \leqslant j \leqslant m$, is chosen randomly;

2 Run a test to verify that all parity check positions, or sums of them, have weight at least l. Also check that every information symbol appears in at least l parity checks. If this is not the case, go to Step 2;

3 We have now created the secret matrix $\mathbf{G}$. Create the public-key matrix as usual, i.e., randomly choose $(\mathbf{S}, \mathbf{P})$, $\mathbf{P}$ permutation and $\mathbf{S}$ non-singular, and compute $\mathbf{SGP} = \hat{\mathbf{G}}$;

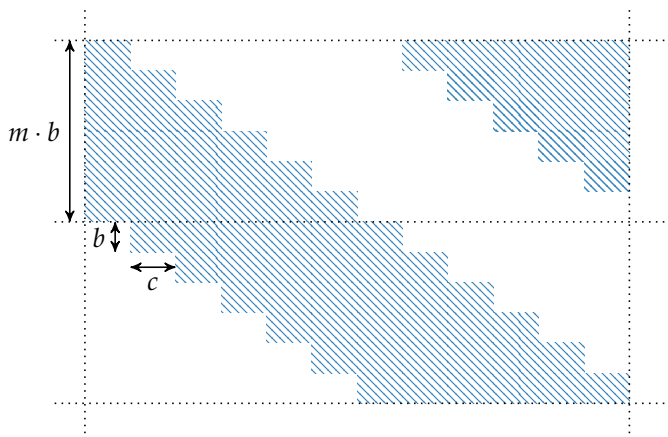


Figure 4.1: The tail-biting construction.

states, which would be formed from the $m \cdot b$ received information symbols in $\hat{\mathbf{c}}$ just before time t. Since these symbols may be perturbed by the error vector, we put all states reachable by e' errors on the stack, where e' is fixed by the decoding algorithm. The expected number of errors in an interval of $m \cdot b$ symbols is

$$\frac{e}{n} \cdot m \cdot b. \tag{4.3}$$

If decoding is unsuccessful we can try a different starting point t. If the

Algorithm 7 (Decoding in tail-biting construction)

Input: Ciphertext sequence $\hat{\mathbf{c}} \in \mathbb{F}_2^n$.

Output: Plaintext sequence $\mathbf{u} \in \mathbb{F}_2^{(nb/c)}$.

1. Write $\hat{\mathbf{c}} = (\mathbf{c}_1, \mathbf{c}_2, \ldots \mathbf{c}_{n/c})$, where $\mathbf{c}_i = (\mathbf{u}_i, \mathbf{p}_i)$ is a c bit vector and $\mathbf{u}_i$ is the systematic part;
2. Choose a random starting point t and put $(\mathbf{u}_{t-m}, \mathbf{u}_{t-m+1}, \ldots, \mathbf{u}_{t-1})$ (index modulo $\frac{n}{c}$), together with all other vectors with at most distance e' from it as starting states in $\mathcal{V}$. Run the stack algorithm until time t is reached again. If decoding was successful, return the decoded sequence. Otherwise, do this item again;
3. If many starting points t have been tried without success, ask for retransmission;

correct state has been established, and decoding is successful, we will get the correct weight of the error vector. As the weight is assumed to be known, this is a way of knowing that decoding was successful. A description of a basic decoding procedure is given in Algorithm 7.

Finding the correct starting state is the major complexity cost in decoding. To be able to decode correctly, we need the correct starting state in $\mathcal{V}$, but we also do not want $\mathcal{V}$ to be too large as this gives a high decoding complexity. As mentioned, we can use different starting points t, and hope that one of them will have a set $\mathcal{V}$ including the correct starting state. We can even put sets $\mathcal{V}$ from many different starting points on the same stack.

In order to decrease the size of $\mathcal{V}$, we propose to use not only $m \cdot b$ bits to form the set $\mathcal{V}$, but to use $m \cdot b + m' \cdot c$ consecutive bits from $\mathbf{c}$, where we now have $(m + m') \cdot b$ information bits but can also use $m' \cdot (c - b)$ parity checks. Deciding that we include all such length $m \cdot b + m' \cdot c$ with at most e' errors as possible starting states, this corresponds exactly to finding all codewords in a $[m \cdot b + m' \cdot c, (m + m') \cdot b]$ linear code with weight at most e'.

A HYBRID CONSTRUCTION

Our construction based on tail-biting convolutional codes has a simple description and a good portion of randomness, which both are desirable properties. One problem however, is the first part of the decoding, i.e., finding the correct starting state before starting the efficient stack algorithm. For constructions with security level around 2^{80} ($\lambda \approx 80$), this can be done with reasonable complexity. But the complexity of this process grows exponentially with the

security level measured in bits.

In order to solve this problem we propose a modified construction, using only polynomial complexity to get the starting state. Our second construction uses a small block code for setting a starting state and then a larger convolutional code for the remaining part of the code. The original code is generated by the generator matrix of the form given in Figure 4.2, where $\mathbf{G}_B$ is a generator matrix of a block code (we propose a Goppa code to be used), $\mathbf{G}_C$ is a generator matrix of a random time-varying systematic convolutional code as in the first tail-biting construction with memory $m = k$ and white areas represent all zeroes.

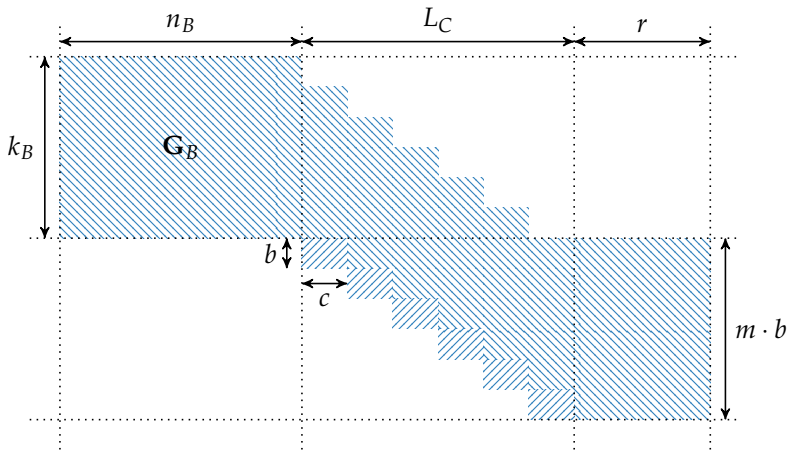


Figure 4.2: McEliece with the combined Goppa and convolutional code construction.

The block code $\mathbf{G}_B$ is an $[n_B, k_B, t_B]$ linear code with efficient decoding of up to t_B errors. We suggest it to be a classical Goppa code. Since the number of errors in the first n_B positions can be higher than the block code can correct, we note that a decoding error might occur here. It is given by

$$\mathbb{P}\left(\text{more than } t_B \text{ errors in block code}\right) = \sum_{x>t_B}^{e} \frac{\binom{n_B}{x}\binom{n_C}{e-x}}{\binom{n}{e}}. \tag{4.4}$$

With suitable parameters this probability can be made very low. An important issue is that we need to have a dimension k_B of the code larger than the number of bits in memory $m \cdot b$. This again comes from the fact that we cannot allow low-weight codewords in the dual code. As the block code generally will have more parity-check symbols per information symbol that leads to an expected existence of codewords with weight lower than in the convolutional

code. We require that no codewords of weight less than l exist in the dual code of $\mathbf{G}_B$.

We then use, as before, a rate $\frac{b}{c}$ convolutional code $\mathbf{G}_C$. Since the leftmost k_B information bits are assumed to be already known (if the block decoding is successful), we do not need to decode these using the sequential decoding procedure.

Finally, we note that the last r positions are needed to protect the last bits with sufficiently many parity checks. As noted by Landais and Tillich in [LT13b], r must be set sufficiently large to avoid low-weight codewords in the code generated by $\mathbf{G}$. The sub-code spanned by the last b rows of $\mathbf{G}$ has about 2^{b-1} codewords of weight $\frac{c+r}{2}$ or less. If r is small, a key-recovery attack can (roughly) be mounted as follows:

1. An arbitrary ISD algorithm can be applied on $\mathbf{G}$ to find a codeword of weight $w \leqslant \frac{c+r}{2}$.

2. With high probability, the support of a found codeword correspond to a subset of the last $c + r$ columns. This could be repeated to obtain all columns.

3. Puncture out the columns obtained in previous step. Repeat the first step with weight $w \leqslant \frac{c}{2}$ until only the columns of the Goppa code $\mathbf{G}_B$ remains.

ANALYSIS AND PARAMETERS

We consider the two standard approaches to attack McEliece PKC: recovery of the message from a received ciphertext and recovery of the key, i.e., determining structure of the secret generator matrix.

Recall that in a structural attack, we try to recover the structure of the code. In our case the only deviation from a random code is the convolutional code structure in terms of low-weight parity checks. In fact, in pre-computation we specified the weight of parity checks to be no less than l. We expect that a structural attack would need to recover parity checks of weight l, and sketch the difficulty of this problem. It is well-known that all parity checks form codewords of the dual code $\mathcal{C}^{\perp}$.

One should also observe that we have a large number of low-weight parity checks. This decreases the complexity of finding one of them when running Stern's algorithm, as the probability of being successful in one iteration increases with the number of existing low-weight vectors. In our case, parity checks are created from $m \cdot b$ bits in memory and c additional bits. For example, there are then at least 2^{c-b-1} parity checks of weight $\frac{1}{2} \cdot (m \cdot b + c)$ or less. However, in the pre-computation creating the generator matrix $\mathbf{G}$, we

can keep track of how many low-weight vectors we have and make sure that the complexity of finding any of them is high enough.

- **Tail-biting** $\lambda \approx 80$: For security level around 2^{80}, we propose the following set of parameters. Use parameters

$$(n, k, e, m) = (1800, 1200, 45, 12)$$

with rate $R = \frac{20}{30}$. We will have a security of 2^{81} against decoding attacks, measured with complexity expressions for Stern's algorithm. We get a security level of $2^{78.4}$ with Algorithm 5. In the construction phase, we set $l = 125$, i.e., every parity check should have a weight no less than 125. This is achievable as every parity check has $240 + 30$ positions where it can be non-zero. The complexity of finding a weight 125 vector in the dual code using Stern's algorithm is roughly 2^{87}. As this complexity is decreased by the number of low-weight vectors, we need to keep track of the number of low-weight vectors in the code to guarantee a complexity of about 2^{80} for finding such a low-weight parity check.

For the decoding, we keep $m \cdot b = 240$ bits as our starting state. Fixing a starting time t, the expected number of errors in these 240 information symbols is 6. We include another 40 information symbols from just before time $t - m$ and the corresponding 20 parity checks that is computed from these $240 + 40$ information symbols. We also split the 240 symbols in two halves. Our assumption is now that we have at most 1 error in the added 60 symbols and at most 3 errors in each of the halves of the state.

We then generate two tables of all parity check syndromes that up to 3 errors can generate, in total less than 2^{19} entries in each table. One table is expanded with adding to every entry the 61 syndromes that the additional 60 information symbols created, giving 2^{25} entries. Merging the two tables, knowing that the syndrome must be zero, gives 2^{24} surviving states. Now, taking the next 30 information symbols and starting sequential decoding, quickly takes the correct state to the top of the stack. If we are unsuccessful, pick a new t.

The probability that our assumption about errors is correct is about $\frac{1}{4}$, so the expected total decoding complexity is about

$$\left(\frac{1}{4}\right)^{-1} \cdot 2^{25} = 2^{27}.$$

By assuming fewer errors, the expected decoding complexity can be further decreased.

- **Hybrid** $\lambda \approx 80$: Pick as $\mathbf{G}_B$ a Goppa code of length $n_B = 1020$, dimension $k_B = 660$ and with capability of correcting 36 errors. Let the convolutional code have rate $R = \frac{20}{30}$ and run 25 information blocks. In [LT13b], it was suggested to use at least $r = 140$ (it might be necessary to increase it even more for 80-bit security). We should add that m must be very large to avoid low-weight codewords in the dual code due to the increased r. This gives rise to a full code (at least) of length $n = 1020 + 25 \cdot 30 + 140 = 1910$ and dimension $k = 660 + 25 \cdot 20 = 1160$.

  The decoding step first decodes the Goppa code. For a $n_B = 1020$ code we expect that algorithms with complexity $\mathcal{O}(n^2)$ are still the best choice. There are algorithms with better asymptotic performance, but they are useful only for excessively large n_B. So, we expect (say) 2^{20} steps for this part. Decoding the convolutional code is then done in much less time than 2^{20} steps. Overall, this gives a faster decoding than standard McEliece, requiring close to 2^{22} steps. With 53 inserted errors, the probability that we get a decoding error can be found to be around $2^{-6.6}$, see (4.4), which might be problematic in practical applications. We may conclude that the hybrid construction is hard to repair, as we were unable to find parameters that at the same time attain good security, acceptable efficiency and small key-size.

4.3.2 MCELIECE BASED ON MDPC CODES

Quite recently, a promising McEliece variant was proposed by Misoczki *et al.* in [MTSB13], based on MDPC codes. The MDPC-McEliece construction deviates slightly from the original proposal in the sense that the column permutation of $\mathbf{G}$ is omitted. In other words, KeyGen differs from the what was described earlier.

1. Select an $[n, r]$ linear code $\mathcal{C}$ in the QC-MDPC family, which has a parity-check matrix $\mathbf{H} \in \mathbb{F}_2^{r \times n}$, $n = n_0 \cdot r$, such that

$$\mathbf{H} = \begin{pmatrix} \mathbf{H}_0 & \mathbf{H}_1 & \cdots & \mathbf{H}_{n_0-1} \end{pmatrix}, \tag{4.5}$$

   where each $\mathbf{H}_i$ is a circulant $r \times r$ matrix with weight w_i in each row and with total row weight $w = \sum w_i$.

2. Generate the public key $\mathbf{G} \in \mathbb{F}_2^{(n-r) \times n}$ from $\mathbf{H}$ in the following way,

$$\mathbf{G} = \begin{pmatrix} \mathbf{I} & \mathbf{P} \end{pmatrix}, \tag{4.6}$$

where

$$\mathbf{P} = \begin{pmatrix} \mathbf{P}_0 \\ \mathbf{P}_1 \\ \vdots \\ \mathbf{P}_{n_0-2} \end{pmatrix} = \begin{pmatrix} \left(\mathbf{H}_{n_0-1}^{-1}\mathbf{H}_0\right)^{\mathrm{T}} \\ \left(\mathbf{H}_{n_0-1}^{-1}\mathbf{H}_1\right)^{\mathrm{T}} \\ \vdots \\ \left(\mathbf{H}_{n_0-1}^{-1}\mathbf{H}_{n_0-2}\right)^{\mathrm{T}} \end{pmatrix}. \tag{4.7}$$

Note: The parity-check matrix $\mathbf{P}$ here should not be confused with the permutation matrix in the original McEliece construction. The QC-MDPC construction has no need for a permutation matrix.

- **Encryption**: Let $\mathbf{m} \in \mathbb{F}_2^{(n-r)}$ be the plaintext. Multiply $\mathbf{m}$ with the public key $\mathbf{G}$ and add noise within the correction radius t of the code, i.e., $\mathbf{c} = \mathbf{mG} + \mathbf{e}$, where $w_{\mathrm{H}}(\mathbf{e}) \leqslant t$. Here t is determined by the error-correction capability of the decoding algorithm for the MDPC code [MTSB13].

- **Decryption**: Let $\mathbf{c} \in \mathbb{F}_2^n$ be the ciphertext. Given the secret low-weight parity check matrix $\mathbf{H}$, a low-complexity decoding procedure is used to obtain the plaintext $\mathbf{m}$.

  The decoding procedure of MDPC codes is a variant of Gallager's bit-flipping algorithm. The latter, an iterative decoding algorithm commonly used with LDPC codes. LDPC codes are efficient because of the low parity-check weights – error-correction capability decreases linearly with the weight of the parity checks. Since MDPC codes have slightly higher parity-check weights than LDPC codes, one should expect to have worse error-correction capability. This is not a big problem, because error-correcting capability is not »very« important in a cryptographic context. We will leave out the details here, a more thorough description can be found in [MTSB13].

PROPOSED PARAMETERS

The authors of [MTSB13] initially proposed the parameters found in Table 4.1 for a McEliece PKC QC-MDPC scheme with 80-bit, 128-bit and 256-bit security level.

An actual implementation for the QC-MDPC scheme was recently published in [HvMG13]. They demonstrated excellent efficiency in terms of computational complexity and key sizes for encryption and decryption on an FPGA using a key size of $r = 4800$. In [MTSB13], the designers of the MDPC scheme subsequently changed all circulant block sizes r to be odd, which currently avoids the attack described in this thesis.

Parameters						
n	r	w	t	n_0	Key-size	Security
9600	4800	90	84	2	4800	80
27200	6800	340	68	4	20400	128
65536	32768	274	264	2	32768	256

Table 4.1: A subset of the proposed QC-MDPC instances with corresponding security and key-size.

4.4 THE TCHO CIPHER

We have seen that in public-key cryptography there are several different cryptosystems based on different hard problems such as integer factorization, discrete logarithm, MDD and CSD problems. In a similar direction, Finiasz and Vaudenay proposed a public-key cryptosystem called TCHo[1] [FV06], based on the problem of finding low-weight multiples of a given binary polynomial. The original proposal was later refined by Aumasson, Finiasz, Meier and Vaudenay in [FV07]. Herrmann and Leander demonstrated in [HL09] a chosen-ciphertext attack on the basic form of TCHo, implying that the use of TCHo with a hybrid encryption framework as originally proposed in [FV06] is the way to use TCHo.

TCHo is a public-key cryptosystem based on the problem of finding a low-weight polynomial multiple, see 6 for more details. The operation of all variants of the public-key cryptosystem TCHo is non-deterministic and can be thought of as transmitting data over a noisy channel, very much like in the McEliece public-key cryptosystem. A high level description is as follows. A linear code encodes the message into a codeword, while an LFSR with a random starting state produces a bitstream that together with a low-weight error vector constitutes the noise on the channel. The connection polynomial of the LFSR is the only component of the public key.

The trapdoor of the cipher works in the following way: the secret low-weight polynomial of fixed degree, which is the only component of the secret key, is divisible by the characteristic polynomial used by the LFSR. Therefore, by »multiplying« the ciphertext with the secret polynomial, the contribution from the LFSR sequence diminishes, leaving only a low-weight error vector as noise. This sequence can then be decoded using conventional decoding algorithms. One implementation of TCHo is as follows.

[1] Acronym for Trapdoor Cipher, Hardware Oriented. The word tchô is also a French slang word originating from *Tchô, monde cruel*, created by the Swiss cartoonist Philippe 'Zep' Chappuis in 1997.

- **Encryption**: Let $\mathbf{G}_{\text{rep}}$ be a generator matrix of a repetition code of length l. The message $\mathbf{m} \in \mathbb{F}_2^{128}$ is repeated and the result is truncated to fit a length l. These two steps can be represented by multiplication with $\mathbf{G}_{\text{rep}}$. In encoding, let $\mathbf{r} = \begin{pmatrix} r_0 & r_1 & \cdots & r_{l-1} \end{pmatrix}$ be a random string of l independent bits with bias

$$\mathbb{P}\left(r_i = 0\right) = \frac{1}{2} \cdot (1 + \gamma). \tag{4.8}$$

Here, every element of $\mathbf{r}$ is chosen according to Ber_γ, such that $\mathbf{r}$ is highly biased and has a lot more zeroes than ones. Using an LFSR with characteristic polynomial $P(x)$ and a randomly chosen starting state, the LFSR sequence $\mathbf{p}$ is generated and truncated to l bits. The encryption is done by adding the three vectors to form the ciphertext $\mathbf{c} \in \mathbb{F}_2^l$, so

$$\mathbf{c} = \mathbf{m}\mathbf{G}_{\text{rep}} + \mathbf{r} + \mathbf{p}. \tag{4.9}$$

A block representation of the encryption mechanism in TCHo is given in Figure 4.3.

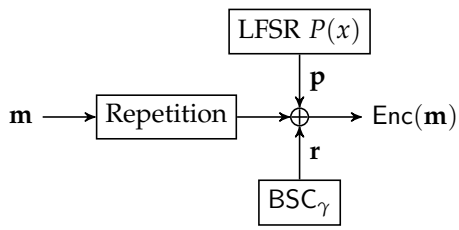


Figure 4.3: Encryption in the TCHo cryptosystem.

- **Decryption**: Given the secret low-weight polynomial $K(x)$ of fixed degree, we can construct the matrix

$$\mathbf{M} = \begin{pmatrix} k_0 & k_1 & \cdots & k_d & & & \\ & k_0 & k_1 & \cdots & k_d & & \\ & & \ddots & \ddots & & \ddots & \\ & & & k_0 & k_1 & \cdots & k_d \end{pmatrix}. \tag{4.10}$$

For the decoding step, consider the product $\mathbf{t} = \mathbf{c}\mathbf{M}^{\mathrm{T}}$. From the encryption step we have that

$$\begin{aligned} \mathbf{t} =& (\mathbf{m}\mathbf{G}_{\text{rep}} + \mathbf{r} + \mathbf{p})\mathbf{M}^{\mathrm{T}} \\ =& \mathbf{m}\mathbf{G}_{\text{rep}}\mathbf{M}^{\mathrm{T}} + \mathbf{r}\mathbf{M}^{\mathrm{T}} + \mathbf{p}\mathbf{M}^{\mathrm{T}} \\ =& \mathbf{m}\mathbf{G}_{\text{rep}}\mathbf{M}^{\mathrm{T}} + \mathbf{r}\mathbf{M}^{\mathrm{T}}. \end{aligned} \tag{4.11}$$

Since $P(x)$ divides $K(x)$, we have that $\mathbf{p}\mathbf{M}^{\mathrm{T}} = \mathbf{0}$. Recall that each bit element in $\mathbf{r}$ was γ-biased. $K(x)$ has weight w and consequently, each element in $\mathbf{r}$ will be a sum of w variables that each has bias γ. Therefore, each element in $\mathbf{r}\mathbf{M}^{\mathrm{T}}$ will be γ^w-biased (by Lemma 1.3). Here, majority-decision decoding can be used to decode

$$\mathbf{t} = \mathbf{m}\left(\mathbf{G}_{\mathrm{rep}}\mathbf{M}^{\mathrm{T}}\right) + \mathbf{r}\mathbf{M}^{\mathrm{T}}, \tag{4.12}$$

i.e., to find a solution $\mathbf{m}$ such that the residual $\mathbf{r}\mathbf{M}^{\mathrm{T}}$ is minimized.

4.4.1 PROPOSED PARAMETERS

The authors of [FV07] suggested the parameters in Table 4.2 for 80-bit security against key-recovery attacks.

d	d_P	w	γ
25820	7000	45	0.981
24730	12470	67	0.981
44677	4433	25	$1 - \frac{3}{64}$
24500	8000	51	0.98
17600	8795	81	$1 - \frac{3}{128}$
31500	13200	65	$1 - \frac{1}{64}$

Table 4.2: Parameters for 80-bit security.

4.5 SUMMARY

In this chapter, we have introduced different coding-based cryptosystems. First, we have given an overview of the original McEliece construction. Secondly, we have proposed a new variant of McEliece based on tail-biting convolutional codes and a hybrid using Goppa codes and convolutional codes in conjunction, and given parameters for 80-bit security. Although it is possible to design such McEliece cryptosystems, we found that the proposed constructions might not be the best choices. However, the idea is still interesting and further investigation might provide more useful constructions. Finally, we have given descriptions of an MDPC-variant of McEliece and the public-key cryptosystem TCHo.

5

A Squaring Attack on QC-MDPC-based McEliece

When in doubt, use brute force.
– *Ken Thompson*

In this chapter, we present an attack on the quasi-cyclic MDPC variant of the McEliece public-key cryptosystem described in Section 4.3.2. The attack was originally proposed in [LJS+15].

To this date, several variants of McEliece based on LDPC codes have been proposed – [BCGM07] [BBC08] [BBC13] to cite a few. In these constructions, the permutation matrix **P** used in the original McEliece construction (see Definition 4.1) is replaced by a non-singular matrix **Q** that »scales up« the weight of the low-weight parity checks in the LDPC code to avoid ISD-based attacks.

Taking a different course, the authors of [MTSB13] proposed to deal with the problem of low-weight codewords by moderately increasing the parity-check weight of **G** used in the LDPC construction, yielding what we described before as the MDPC-code construction (see Subsection 2.6.4). As a result, it is possible to avoid the use of the **Q** matrix. Unfortunately, the increased parity-check weight causes a degradation of error-correction performance and therefore the weight of the error vector must be chosen smaller than that of the LDPC-based variants. In [MTSB13], the authors show that the constrained error weight remains sufficient to prevent standard message-recovery attacks. The QC-MDPC proposal is yet unbroken and it is particularly interesting because of its beautiful simplicity. In [HvMG13], the authors implemented QC-MDPC McEliece PKC in FPGA hardware using the same parameters as the presented attack targets.

As a final note, we stress the universality of the attack we present – it can be applied to all quasi-cyclic codes having circulant sub-matrices of even dimension, e.g., the McEliece PKC variants using QC-LDPC codes.

5.1 A KEY-RECOVERY ATTACK

We will now describe our new attack, which we mount as a key-recovery attack and later on extend to message recovery as well.

Recall the key-recovery attack on a McEliece cryptosystem, targeting to find the secret generator matrix. In the context of using QC-MDPC codes, we aim to find any matrix $\mathbf{H}_i$ given only the public-key matrix $\mathbf{P}$. From $\mathbf{H}_i$, the remaining part of $\mathbf{H}$ can easily be recovered using basic linear algebra. Without loss of generality, we assume that the rate of the code is $R = \frac{1}{2}$, and thus, $n_0 = 2$. In fact, $R = \frac{1}{2}$ constitues the worst case as we shall see later on.

Problem 5.1 (QC-MDPC Key Recovery**)** Given the public-key matrix

$$\mathbf{G} = \begin{pmatrix} \mathbf{I} & \mathbf{P} \end{pmatrix} \in \mathbb{F}_2^{(n-r)\times n}, \tag{5.1}$$

recover the secret-key parity-check matrices $\mathbf{H}_0, \mathbf{H}_1, ..., \mathbf{H}_{n_0-1}$. We denote the problem QC-MDPC.

A natural strategy to solve QC-MDPC apart from exhaustive search is to approach the problem as an instance of MDP by applying an information-set decoding algorithm on the instance $(\mathbf{G}, w)$. The code generated by $\mathbf{G}$ includes quasi-cyclic shifts of, what we can assume to be, the only weight w codeword $\begin{pmatrix} \mathbf{h}_1 & \mathbf{h}_0 \end{pmatrix}$. Here, $\begin{pmatrix} \mathbf{h}_1 & \mathbf{h}_0 \end{pmatrix}$ is any row of $\begin{pmatrix} \mathbf{H}_1 & \mathbf{H}_0 \end{pmatrix}$. Undoubtedly, an oracle for MDP with the instance ($\mathbf{G}$, w) as input can solve QC-MDPC with only polynomial overhead.

The attack we present exploits the quasi-cyclic structure by employing a novel squaring technique, making the attack significantly more efficient than solving MDP using standard techniques. From the construction given in [MTSB13], we have that

$$\mathbf{P}_0 = \left(\mathbf{H}_1^{-1}\mathbf{H}_0\right)^{\mathrm{T}} \iff \mathbf{H}_1\mathbf{P}_0^{\mathrm{T}} = \mathbf{H}_0. \tag{5.2}$$

Since the matrices $\mathbf{H}_0$, $\mathbf{H}_1$ and $\mathbf{P}_0$ are all circulant, the problem is equivalent to solving an equation over the polynomial ring $\mathbb{F}_2[x]/(x^r + 1)$. More specifically, there is a map between the circulant matrices and $\mathbb{F}_2[x]/(x^r + 1)$, as summarized in the following proposition.

Proposition 5.2 Let $\mathfrak{C}_r$ be the set of all $r \times r$ circulant matrices over $\mathbb{F}_2$. Then, there exists an isomorphism between the rings $(\mathfrak{C}_r, +, \cdot)$ and $(\mathbb{F}_2[x]/(x^r + 1), +, \cdot)$.

For notational convenience, let $h_1(x)$ represent $\mathbf{H}_1$ and $h_0(x)$ represent $\mathbf{H}_0$. From (5.2), with $\mathbf{P}_0^{\mathrm{T}}$ represented with $P(x)$, the following equation can be formed,

$$h_1(x)P(x) \equiv h_0(x) \bmod (x^r + 1). \tag{5.3}$$

The key idea in our new attack to be described is a squaring technique applied on the polynomials in (5.3). Squaring both sides gives

$$[h_1(x)P(x)]^2 \equiv h_0(x)^2 \bmod (x^r + 1). \tag{5.4}$$

If the dimension r is even, then the degree of each non-zero term of $h_1(x)^2$ and $h_0(x)^2$ is a multiple of 2. More generally, when squared d times, the degree of any non-zero term will be a multiple of 2^d if and only if $2^d|r$. If so, we can omit all positions that are not a multiple of 2^d, and thus, the dimension decreases by a factor 2^d. As a consequence of the dimension reduction, collisions between non-zero elements can occur and, if so, the weight of the sought codeword will be lowered. The weight reduction is the key to our attack.

Before we delve into the deeper concepts of the attack, we need to establish some terminology and develop a basic theoretical foundation. We begin by developing the concept of polynomial square roots and how to compute them efficiently. Once established, we proceed by giving a detailed description of the attack. Lastly, we unravel the analogy between key recovery and message recovery.

5.1.1 POLYNOMIAL SQUARE ROOTS

In this section, we will focus on the problem of finding polynomial square roots. That is, given a polynomial denoted $a(x)^2 \in \mathbb{F}_2[x]/(x^r + 1)$, determine possible values for the polynomial $a(x)$.

Naturally, there are a lot of square roots of $a(x)^2$, but only one or a few will satisfy the constraints, i.e., the low-weight requirement on the polynomial. The polynomials in our setting will come in linearly dependent pairs $a(x)$ and $b(x)$, where the weight of both polynomials is constrained by some integer w. More formally, we state our problem as follows.

Problem 5.3 (Bounded-Weight Polynomial Square Root) Given the polynomials $a(x)^2, b(x)^2, P(x) \in \mathbb{F}_2[x]/(x^r + 1)$ fulfilling $b(x)^2 = a(x)^2 P(x)^2$, find $a(x), b(x)$ such that

$$\begin{cases} w_{\mathrm{H}}(a(x)) & \leqslant w, \\ w_{\mathrm{H}}(b(x)) & = w_{\mathrm{H}}(a(x)P(x)) \leqslant w. \end{cases} \tag{5.5}$$

We denote the problem BWPSR.

Clearly, the problem is trivial when r is odd, because then no collisions can occur. We therefore can assume that r is even. First, we focus on enumerating the possible solutions to the square root problem. Assume that we have a given polynomial written as $a(x)^2$, and we would like to enumerate the $2^{r/2}$ different square roots $a(x) \in \mathbb{F}_2[x]/(x^r + 1)$.

The two terms x^i and $x^{i+r/2}$ will result in the same term when squared. Define Γ to be the set of all such pairs of terms, i.e.,

$$\Gamma \stackrel{\text{def}}{=} \left\{ (x^i, x^{i+r/2}) : 0 \leqslant i < \frac{r}{2} \right\}. \tag{5.6}$$

We introduce the following definition:

Definition 5.1 Let $\mathcal{R}_{a(x)}$ be the set of pairs in Γ giving rise to a term present in the expression of $a(x)^2$. Furthermore, let $\mathcal{Q}_{a(x)}$ be the set of pairs in Γ giving rise to a term not present in the expression of $a(x)^2$.

The set $\mathcal{R}_{a(x)}$ is called the root set and their elements are called root pairs. Similarly, the set $\mathcal{Q}_{a(x)}$ is called the collision set and the elements are called collision pairs.

$$\begin{array}{ccccccccccc}
a(x)^2 & = & x^{i_1} & + & x^{i_2} & + & \cdots & + & 0 \\
 & & \diagup\ \diagdown & & \diagup\ \diagdown & & & & \diagup\ \diagdown \\
a(x) & = & \underbrace{\{x^{j_1},\ x^{r/2+j_1}\} + \{x^{j_2},\ x^{r/2+j_2}\}}_{\text{Root set } \mathcal{R}_{a(x)}} & + & \cdots & + & \cdots & + & \underbrace{\{x^{j_{r/2}},\ x^{r/2+j_{r/2}}\}}_{\text{Collision set } \mathcal{Q}_{a(x)}}
\end{array}$$

The notation above essentially means that for pairs of terms in the root set, exactly one of the two terms is present in $a(x)$. For pairs from the collision set, either none or both of the terms in a pair are present in $a(x)$. This is illustrated above and in the following example.

Example 5.1 *The polynomial square root* $\sqrt{x^2+1} \bmod (x^6+1)$ *has several solutions. Among these we find* $x+1$ *and* x^5+x^2+x+1*, since they both are equal to* x^2+1 *when squared. In the latter polynomial,* x^5 *and* x^2 *has collided into a zero element because* $(x^5+x^2)^2 \equiv 0 \bmod (x^6+1)$*. The pair* (x^5, x^2) *belongs to the collision set and the pairs* (x, x^4) *and* $(1, x^3)$ *belong to the root set because* $(x)^2 \equiv (x^4)^2 \bmod (x^6+1)$ *and* $(1)^2 \equiv (x^3)^2 \bmod (x^6+1)$*, repectively.*

Let us return to Problem 5.3. Our problem can be formulated using linear algebra. By Proposition 5.2, we can express the polynomials $a(x)^2, b(x)^2$ and $P(x)^2$ in a circulant matrix form, i.e., $\mathbf{A}^2\mathbf{P}^2 = \mathbf{B}^2$. It is sufficient to represent the circulant matrices $\mathbf{A}^2$ and $\mathbf{B}^2$ as vectors denoted $\mathbf{a}^2$ and $\mathbf{b}^2$, repectively. So, if $\mathbf{a}^2$ and $\mathbf{b}^2$ are taken from the first row in $\mathbf{A}^2$ and $\mathbf{B}^2$, then

$$a(x)^2 P(x)^2 = b(x)^2 \iff \mathbf{a}^2\mathbf{P}^2 = \mathbf{b}^2. \tag{5.7}$$

Note that $\mathbf{a}^2$ and $\mathbf{b}^2$ each have at most $\frac{r}{2}$ non-zero entries (the odd ones are zero due to squaring).

The entries of a square root $\mathbf{a} = \begin{pmatrix} a_0 & a_1 & \dots & a_{r-1} \end{pmatrix}$ will fulfill the linear equation

$$a_{r/2+i} + a_i = 1, \quad 0 \leqslant i < \frac{r}{2}, \tag{5.8}$$

if and only if $(x^i, x^{r/2+i}) \in \mathcal{R}_{a(x)}$. Analogously, the entries will fulfill the linear equation

$$a_{r/2+i} + a_i = 0, \quad 0 \leqslant i < \frac{r}{2}, \tag{5.9}$$

if and only if $(x^i, x^{r/2+i}) \in \mathcal{Q}_{a(x)}$. Naturally, the same properties apply to $\mathbf{b}$.

Now we can describe a very efficient reconstruction procedure for our problem. First we identify all collisions as follows. From (5.7) we use that the polynomial pair $a(x)$ and $b(x)$ is mapped to a low-weight codeword $\mathbf{u}$ in the code $\mathcal{C}$ generated by $\mathbf{G} = \begin{pmatrix} \mathbf{I} & \mathbf{P} \end{pmatrix}$. The weight of $\mathbf{u}$ is $w_{\mathrm{H}}(a(x)) + w_{\mathrm{H}}(b(x))$. Similarly, the polynomial pair $a(x)^2$ and $b(x)^2$ is a codeword $\mathbf{u}^2$ in the code $\mathcal{C}^2$ generated by $\mathbf{G}^+ = \begin{pmatrix} \mathbf{I} & \mathbf{P}^2 \end{pmatrix}$, and the weight of $\mathbf{u}^2$ is $w_{\mathrm{H}}\left(a(x)^2\right) + w_{\mathrm{H}}\left(b(x)^2\right)$. The collisions are now found using the following results.

Lemma 5.4 Let $\mathcal{C}$ and $\mathcal{C}^2$ be the codes as described above. If $\mathbf{u} = (\mathbf{a} \| \mathbf{b}) \in \mathcal{C}$ has weight w_1 and $\mathbf{u}^2 = (\mathbf{a}^2 \| \mathbf{b}^2) \in \mathcal{C}^2$ has weight w_2, then there exists a map

$$\begin{array}{rccc} \sigma: & \mathcal{C} & \longrightarrow & \mathcal{C}' \\ & \rotatebox{90}{$\in$} & & \rotatebox{90}{$\in$} \\ & \mathbf{u} & \longmapsto & \mathbf{u}', \end{array} \tag{5.10}$$

such that $\mathcal{C}'$ is a (non-cyclic) $[2r - 2w_2, r]$ linear code containing the codeword $\mathbf{u}'$ of weight $w_1 - w_2$.

Proof. By puncturing the codeword symbols corresponding to the root sets $\mathcal{R}_{a(x)}$ and $\mathcal{R}_{b(x)}$, we remove all the non-zero contributions from $\mathbf{u}^2$. In total, we remove $2w_2$ symbols and decrease the weight by w_2. The only remaining non-zero contributions are those from collisions. So the punctured codeword will have weight $w_1 - w_2$. ■

The weight $w_1 - w_2$ is typically very small and one can compute the solution using an ISD algorithm very efficiently. We can even improve this result by slightly altering the mapping σ. For example, from the linear relations (5.8) the dimension of the code $\mathcal{C}'$ can be lowered.

After having established all collisions as above, we have established the coefficient value for all terms in the collision sets. The second and remaining step is to recover the coefficient value for all terms in the root sets. Given

Algorithm 8 (Polynomial square root)

Input: Polynomials $a(x)^2, b(x)^2, P(x) \in \mathbb{F}_2[x]/(x^r+1)$ and algorithm parameter $w \in \mathbb{N}$.

Output: Polynomials $a(x), b(x) \in \mathbb{F}_2[x]/(x^r+1)$.

1 Using $a(x)^2$ and $b(x)^2$, determine their root sets $\mathcal{R}_{a(x)}$ and $\mathcal{R}_{b(x)}$, and collision sets $\mathcal{Q}_{a(x)}$ and $\mathcal{Q}_{b(x)}$;
2 Construct the matrix $\begin{pmatrix}\mathbf{I} & \mathbf{P}\end{pmatrix}$ from $P(x)$;
3 Construct the mapping σ and apply it to the code generated by $\begin{pmatrix}\mathbf{I} & \mathbf{P}\end{pmatrix}$ to obtain the punctured code with generator matrix $\mathbf{G}'$;
4 $\delta \leftarrow w - w_{\mathrm{H}}\left(a(x)^2\right) - w_{\mathrm{H}}\left(b(x)^2\right)$;
5 $\mathbf{u} \leftarrow \mathsf{ISD}(\mathbf{G}', \delta)$;
6 Solve $\mathbf{aP} = \mathbf{b}$ to obtain the remaining unknowns;

the linear equations (5.8), the system of linear equations $\mathbf{aP} = \mathbf{b}$ and the knowledge of the collisions, we solve for the remaining unknowns.

Based on our arguments, we formulate an algorithm to solve BWPSR for an arbitrary instance $(a(x)^2, b(x)^2, P(x), w)$ with a constrained weight w in Algorithm 8.

In conclusion, we have observed that the polynomial reconstruction can be very efficiently done, by first using a puncturing step and then solve linear equations for the remaining unknowns.

5.1.2 THE ATTACK

In this subsection, we will present the squaring attack on quasi-cyclic codes of even dimension, based on the techniques outlined. Let us recall that when squaring a polynomial in a polynomial ring $\mathbb{F}_2[x]/(x^r+1)$, the dimension decreases if and only if r is even.

As noted, the attack is very general, but for illustrative purposes we choose to apply it on the even-dimension instances of QC-MDPC McEliece, i.e., when $2^d | r$ holds for the dimension r for some positive integer d. We divide the attack into three separate parts.

SQUARING

First, we choose an integer q such that $0 < q \leqslant d$. By repeatedly squaring $P(x)$ for q times, we can obtain the polynomial $P(x)^{2^q}$ with the relation from (5.3),

$$[h_1(x)P(x)]^{2^q} \equiv h_0(x)^{2^q} \bmod (x^r+1), \tag{5.11}$$

where each involved polynomial has dimension $r/2^q$. Moreover, the weights of $h_0(x)^{2^q}$ and $h_1(x)^{2^q}$ will have decreased with some non-zero probability depending on the parameter q. As an example, Figure 5.1 shows the simulated polynomial weight distribution of $a(x)^{2^q}$, when $a(x)$ is chosen uniformly over all polynomials of weight $w = 45$ in $\mathbb{F}_2[x]/(x^{4800}+1)$.

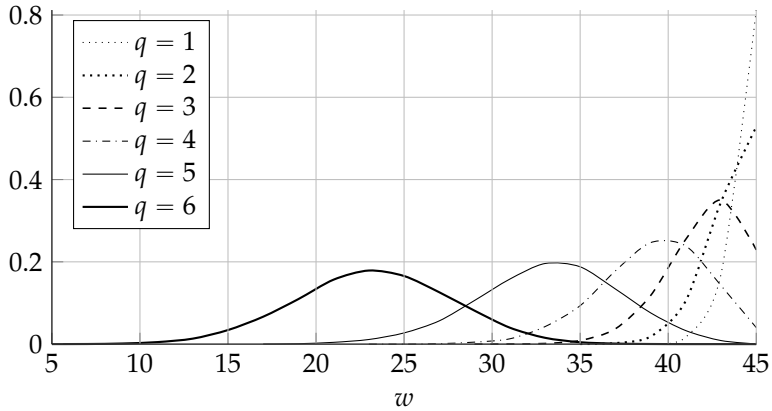


Figure 5.1: The simulated polynomial weight distribution $\mathbb{P}\left(w_{\mathrm{H}}\left(a(x)^{2^q}\right)\right)$ with initial polynomial weight $w = 45$.

FINDING A LOW-WEIGHT POLYNOMIAL

The second step consists in finding a low-weight polynomial in the smaller code. By Proposition 5.2, there is no difference in finding the low-weight polynomial $a(x)$ and finding a low-weight vector $\mathbf{a}$ such that $\mathbf{a}\mathbf{P}^{2^q}$ also has low weight. Therefore, the problem can be transformed into an instance of MDP in which we try to find a codeword of weight at most $2w_q$ in the $[2r/2^q, r/2^q]$ linear code generated by $\begin{pmatrix}\mathbf{I} & \mathbf{P}^{2^q}\end{pmatrix}$, for some selected parameter w_q; for this problem, we can use an arbitrary ISD algorithm, such as Algorithm 4.

It is necessary to point out that with a non-zero probability, there exists a random polynomial $a'(x) \in \mathbb{F}_2[x]/(x^r+1)$, which is not a circular shift of $h_0(x)$ and still satisfies

$$\begin{cases} w_{\mathrm{H}}\left((a'(x))^{2^q}\right) & = w_q, \\ w_{\mathrm{H}}\left((a'(x)P(x))^{2^q}\right) & = w_q. \end{cases} \tag{5.12}$$

In order to make any reasonable claims about the complexity of reconstruction, we first need to determine the expected number of such false positives.

In Section 5.3, we give an expression stating the expected number of false positives (Lemma 5.6).

RECONSTRUCTING THE POLYNOMIALS

Once a solution $h_0(x)^{2^q}$ and $h_1(x)^{2^q}$ has been found, we execute Algorithm 8 to reconstruct its square roots. We consider two main approaches to the attack. They are outlined as follows.

- **Immediate reconstruction**: We reconstruct the final solutions $h_0(x)$ and $h_1(x)$ immediately from $h_0(x)^{2^q}$ and $h_1(x)^{2^q}$. The following mapping

$$\begin{array}{ccc} \mathbb{F}_2^r & \xrightarrow{\text{Squaring } q \text{ times}} & \mathbb{F}_2^{r/2^q} \\ & & \downarrow \quad \text{ISD} \\ \mathbb{F}_2^r & \xleftarrow{\text{Polynomial reconstruction}} & \mathbb{F}_2^{r/2^q} \end{array}$$

  illustrates the attack procedure and how the problem dimension will vary. Every non-zero symbol in $\mathbb{F}_2^{r/2^q}$ will give rise to 2^q possible root-set symbols in $\mathbb{F}_2^r$. From the mapping point of view, this yields that for each non-zero symbol, 2^q symbols are punctured (according to Lemma 5.4). Thus, to reconstruct, we recover a codeword of weight $w - w_2$ in a $[2r - 2^q w_2, r]$ linear code.

- **Russian-doll reconstruction**: If there are a lot of possible square root solutions to consider (mainly when q is large), it may be beneficial to recover the solutions in a stepwise manner. The mapping

$$\begin{array}{ccccccc} \mathbb{F}_2^r & & \longrightarrow & & & & \mathbb{F}_2^{r/2^q} \\ & & & & & & \downarrow \\ \mathbb{F}_2^r & \leftarrow & \mathbb{F}_2^{r/2} & \leftarrow \cdots \leftarrow & \mathbb{F}_2^{r/2^{q-1}} & \leftarrow & \mathbb{F}_2^{r/2^q} \end{array}$$

  gives an idea how the problem dimension will vary in the different steps. By using a stepwise square root reconstruction, we will reconstruct different collided elements in different steps. For an arbitrary step $\mathbb{F}_2^{r/2^i} \leftarrow \mathbb{F}_2^{r/2^{i+1}}$, where $0 \leqslant i < q$, we find a low-weight codeword in a $[2r/q^i - 2w_i, r/q^i]$ linear code, according to Lemma 5.4. The (even) integer w_i is a reconstruction parameter, where $w_i/2$ will be the maximum number of collisions that we can reconstruct in a particular step. It is not known how many collisions have occurred in the squaring $\mathbb{F}_2^{r/2^i} \rightarrow \mathbb{F}_2^{r/2^{i+1}}$. So, the best strategy is to successively increase w_i; first we assume no collisions, then one collision and so on and so forth.

Algorithm 9 (Key recovery)

Input: Public-key matrix $\mathbf{P} \in \mathbb{F}_2^{r \times 2r}$ and algorithm parameters $q, w_1, w_2, ...w_q \in \mathbb{N}$, where $0 < q \leqslant d$.

Output: Matrices $\mathbf{A}, \mathbf{B} \in \mathbb{F}_2^{r \times r}$.

1 Calculate $P(x), P(x)^2, ..., P(x)^{2^q}$ from $\mathbf{P}$;
2 From $P(x)^{2^q}$, construct the code $\mathcal{C}^{2^q}$ with generator matrix $\mathbf{G}'$;
3 $\mathbf{u} \leftarrow \mathsf{ISD}(\mathbf{G}', w')$;
4 Construct $h_0(x)^{2^q}$ and $h_1(x)^{2^q}$ from $\mathbf{u}$;
5 $a(x) \leftarrow h_1(x)^{2^q}$, $b(x) \leftarrow h_0(x)^{2^q}$;
6 **for** $i = q$ **to** 1 **do**
7 $\quad a(x), b(x) \leftarrow$ Polynomial square root$(a(x), b(x), P^{2^{i-1}}(x), w_i)$;
8 Create secret key parity-check matrices $\mathbf{A}$ and $\mathbf{B}$ by cyclically shifting $\mathbf{a}$ and $\mathbf{b}$, respectively;

Once the dimension has reached r in the reconstruction process, the private-key polynomials $h_0(x)$ and $h_1(x)$, or its corresponding matrices $\mathbf{H}_0$ and $\mathbf{H}_1$ have been found. The described attack can be summarized as in Algorithm 9.

5.2 A MESSAGE-RECOVERY ATTACK

In this section, we will present a message-recovery attack under the ciphertext-only attack (COA) model, i.e., recovering one out of many intercepted messages, knowing only the public key $\mathbf{G}$ and received ciphertexts. The message-recovery problem can be stated as follows.

Problem 5.5 (Message recovery) Given the public-key matrix $\mathbf{G} \in \mathbb{F}_2^{(n-r) \times n}$ and ciphertexts from the oracle $\Pi^{\mathbf{G}}_{\text{McEliece}}$, find a secret message $\mathbf{m}$.

A fundamental difference from a key-recovery attack is that we can allow a significantly lower success probability when mounting a message-recovery attack, since in this setting we assume that we have access to an abundance of intercepted messages to work with.

Definition 5.2 (McEliece oracle) A McEliece oracle $\Pi^{\mathbf{G}}_{\text{McEliece}}$ returns binary vectors of the form $\mathbf{c} \leftarrow \mathbf{mG} + \mathbf{e}$, where $\mathbf{m}, \mathbf{e} \xleftarrow{\$} \{0,1\}^k$ and with $w_{\text{H}}(\mathbf{e}) \leqslant t$.

Describing the circumstances of message recovery in other words – if we fail for a certain message, we may query the oracle $\Pi^{\mathbf{G}}_{\text{McEliece}}$ for new messages

until we find one that satisfies the constraints required to succeed with the attack. Apart from this difference, the attack settings in key- and message recovery are essentially the same.

A ciphertext $\mathbf{c}$ is always within a distance t from at least one codeword in $\mathcal{C}$ generated by $\mathbf{G}$. A standard method used in message-recovery of McEliece-type cryptosystems consists in slightly altering the generator matrix by appending the intercepted message as an additional row in $\mathbf{G}$. The resulting code is $\mathcal{C}' = \mathcal{C} \cup \{\mathcal{C} + \mathbf{c}\} = \mathcal{C} \cup \{\mathcal{C} + \mathbf{e}\}$. Hence, $\mathcal{C}'$ is also generated by

$$\mathbf{G}' = \begin{pmatrix} \mathbf{I} & \mathbf{P} \\ \mathbf{e}_0 & \mathbf{e}_1 \end{pmatrix} \in \mathbb{F}_2^{(r+1)\times 2r}, \tag{5.13}$$

where $\mathbf{e} = \begin{pmatrix} \mathbf{e}_0 & \mathbf{e}_1 \end{pmatrix}$. The code $\mathcal{C}'$ has a minimum distance at most t. Thus, if we can find the minimum-weight codeword $\mathbf{e}$, then we can easily determine the secret message by computing $\mathbf{m} = (\mathbf{c} + \mathbf{e})\mathbf{G}^{-1}$.

We will now show that the techniques for recovering the key can be used for recovering a message, with only a few modifications. We query the oracle $\Pi^{\mathbf{G}}_{\text{McEliece}}$ for an encrypted message $\mathbf{c}$, i.e., a message $\mathbf{m}$ encrypted with the public-key matrix $\mathbf{G}$. Expressing the encryption as a function over a polynomial field, we have that

$$c(x) = m(x)G(x) + e(x) \bmod (x^r + 1). \tag{5.14}$$

By squaring the ciphertext polynomial $c(x)$ for q times, we get

$$c(x)^{2^q} = [m(x)G(x)]^{2^q} + e(x)^{2^q} \bmod (x^r + 1), \tag{5.15}$$

where $e(x)^{2^q}$ is likely to reduce in weight if q is sufficiently high. We map the squared polynomials back to their vectorial form $P(x)^{2^q} \mapsto \mathbf{P}^{2^q}$ and $e(x)^{2^q} \mapsto \begin{pmatrix} \mathbf{e}_0^{2^q} & \mathbf{e}_1^{2^q} \end{pmatrix}$ and put them into the context of (5.13). By doing so, a new such relation emerges from the generator matrix

$$\mathbf{G}'' = \begin{pmatrix} \mathbf{I} & \mathbf{P}^{2^q} \\ \mathbf{e}_0^{2^q} & \mathbf{e}_1^{2^q} \end{pmatrix}. \tag{5.16}$$

If the weight of $e(x)^{2^q}$ has decreased according to our assumptions, finding a low-weight codeword in $\mathbf{G}''$ from (5.16) should be considerably easier than in $\mathbf{G}'$.

Unfortunately, the extended code construction $\mathbf{G}''$ (and $\mathbf{G}'$) used in message recovery suffers from a drawback. There exists only one shift of the codeword $\begin{pmatrix} \mathbf{e}_0^{2^q} & \mathbf{e}_1^{2^q} \end{pmatrix}$ in the code, as opposed to key recovery where $r/2^q$ cyclic shifts of the low-weight vector are present. On the whole, the absence of codeword multiples results in a complexity increase that is roughly a factor $\mathcal{O}\left(r/2^q\right)$ compared to the key-recovery attack.

As the next step in the attack, we should perform reconstruction of the polynomial square roots. The ciphertext $\mathbf{c}$ can be expressed as

$$\begin{pmatrix}\mathbf{c}_0 & \mathbf{c}_1\end{pmatrix} = \mathbf{mG} + \mathbf{e} = \begin{pmatrix}\mathbf{m} + \mathbf{e}_0 & \mathbf{mP} + \mathbf{e}_1\end{pmatrix}. \tag{5.17}$$

We define a vector $\mathbf{z}$ as

$$\mathbf{z} \stackrel{\text{def}}{=} \mathbf{c}_1 + \mathbf{c}_0\mathbf{P} = \mathbf{e}_1 + \mathbf{e}_0\mathbf{P}, \tag{5.18}$$

which gives a relation that is similar to (5.3). Suppose that we are given $\mathbf{e}_1^2 + \mathbf{z}^2 = \mathbf{e}_0^2\mathbf{P}^2$, where $\mathbf{e}_0^2$ and $\mathbf{e}_1^2$ are of low weight, and we want to reconstruct the polynomial roots $\mathbf{e}_0$ and $\mathbf{e}_1$. Since $\mathbf{c}_0, \mathbf{c}_1$ are known, we can compute the vector $\mathbf{z}$. We then construct a code generated by

$$\tilde{\mathbf{G}} = \begin{pmatrix}\mathbf{I} & \mathbf{P} \\ \mathbf{0} & \mathbf{z}\end{pmatrix}. \tag{5.19}$$

Using the vectorial representations $\mathbf{e}_0^2$ and $\mathbf{e}_1^2$, we can determine the root-set symbols and create a map σ and apply to the code generated by $\tilde{\mathbf{G}}$, according to Lemma 5.4. By finding the low-weight codeword in the resulting code $\mathcal{C}'$, the value of each collision-set symbol of $e(x)$ can be computed, which in turn provides us with $e(x)$.

5.3 ANALYSIS

In this section, we give an analysis of the complexity of the key-recovery and message-recovery attacks presented in Section 5.1 and 5.2. We measure all complexities in binary operations. For the notation, we assume that w is the weight of the codeword given by $h_0(x)$ and $h_1(x)$, and where the weight of each polynomial is $w' = w/2$.

The codewords in our $[2r/2^q, r/2^q]$ linear code that are not a shift of the desired codeword $\begin{pmatrix}\mathbf{h}_0^{2^q} & \mathbf{h}_1^{2^q}\end{pmatrix}$, but still have low weight $2w_q$ are what we call false positives. They all need to be tested through a reconstruction step before they can be discarded. The number of false positives we will find, grows as the dimension decreases (or q increases). To get an estimate on how many false positives we can expect to find, we formulate the following lemma.

Lemma 5.6 The expected number of false positives of weight s in dimension $[n, k] = [2r/2^q, r/2^q]$ in step 3 of Algorithm 9 is

$$\mu^*(r, q, s) \stackrel{\text{def}}{=} 2^{-r/2^q+1} \cdot \binom{r/2^q}{s/2}^2, \tag{5.20}$$

assuming that $P(x)$ is uniformly random.

Proof. Let X be an integer-valued and discrete random variable for the number of positives. For a randomly chosen polynomial $a(x)^{2^q}$, we have that

$$\mathbb{P}\left(w_{\mathrm{H}}\left((a(x)P(x))^{2^q}\right) = s/2\right) = 2^{-r/2^q+1} \cdot \binom{r/2^q}{s/2}, \tag{5.21}$$

since the total number polynomials having weight that has the same parity as $s/2$ is $2^{r/2^q}/2$ and the number of polynomials having weight $s/2$ is $\binom{r/2^q}{s/2}$. Moreover, the number of possible choices for $a(x)^{2^q}$ is $\binom{r/2^q}{s/2}$. By assuming independent events, the expected number of codewords having $s/2 + s/2$ nonzero elements is

$$\begin{aligned} \mathbb{E}\left(X\right) &= \sum_{a(x)\in\mathcal{A}} \mathbb{P}\left(w_{\mathrm{H}}\left((a(x)P(x))^{2^q}\right) = s/2\right) \\ &= 2^{-r/2^q+1} \cdot \binom{r/2^q}{s/2}^2, \end{aligned} \tag{5.22}$$

where $\mathcal{A} \stackrel{\text{def}}{=} \left\{a(x) \in \mathbb{F}_2[x]/(x^r+1) : w_{\mathrm{H}}\left(a(x)^{2^q}\right) = s/2\right\}$. ■

To determine the number of instances required to find one that satisfies the weight constraints when squared q times, we determine the probability of success for a randomly drawn instance.

Proposition 5.7 Let $a(x) \in \mathbb{F}_2[x]/(x^r-1)$ be arbitrary and $w_{\mathrm{H}}\left(a(x)\right) = s$. Then, the probability of m collisions when squaring $a(x)$ once is

$$\psi(r,s,m) \stackrel{\text{def}}{=} 2^{s-2m} \cdot \binom{r/2}{s-2m}\binom{r/2-s+2m}{m}\binom{r}{s}^{-1}. \tag{5.23}$$

Proof. We use a combinatorial counting argument. There are $\binom{r/2}{s-2m}$ ways to choose $a^2(x)$ (and thus the root set) and 2^{s-2m} ways to pick a polynomial root $a(x)$ satisfying a particular root set. The remaining $r/2-s+2m$ positions must be in the collision set, which gives $\binom{r/2-s+2m}{m}$ possibilities. The total number of possible ways to pick the polynomial $a(x)$ such that it has weight s is $\binom{r}{s}$. ■

The result of Proposition 5.7 only applies to when a polynomial is squared once, i.e., $q = 1$. For q being an arbitrary number of squarings, the expression becomes a bit more complicated.

Lemma 5.8 The probability of m collisions in q squarings of a polynomial of weight s is given by

$$\Psi(r,s,m,q) \stackrel{\text{def}}{=} \sum_{\substack{i_1,i_2,\ldots,i_q \geqslant 0 \\ i_1+i_2+\cdots+i_q=m}} f(r,s,q,i_1,\ldots,i_q), \tag{5.24}$$

where $f(r,s,q,i_1,\ldots,i_q)$ is defined as

$$\psi(r,s,i_1)\cdot\psi(r/2,s-2i_1,i_2)\cdots\psi(r/2^{q-1},s-2(i_1+\cdots+i_{q-1}),i_q). \tag{5.25}$$

Proof. Let the integers $i_1,i_2,\ldots,i_q$ denote the number of collisions occurring in each squaring; we have i_1 collisions in the first squaring, i_2 collisions in the second and so forth. If the integers are chosen such that $i_1+i_2+\cdots+i_q=m$, then that path gives rise to m collisions. The probability of a particular path is given by $f(r,s,q,i_1,\ldots,i_q)$. In order to calculate the total probability $\Psi(r,s,m,q)$ of all paths leading to exactly m collisions, we enumerate all the possible paths to m collisions. Since all the paths (events) are mutually exclusive, we can perform summation over all probabilities to obtain $\Psi(r,s,m,q)$. ■

We can now give an estimate on the expected number of binary operations required for the attack to be successful. It assumes we have a number of instances to attack and success in attacking one of them is enough.

Lemma 5.9 (Attack complexity) The complexity of the attack described, denoted $C^*(r,w)$, is bounded by

$$\begin{aligned}\min_{q,m}\Bigg\{&\frac{1}{\Psi(r,w,m,q)^2}\Big(C_{\mathsf{ISD}}(2r/2^q,r/2^q,w-4m)\\ &+C_{\mathsf{ISD}}(2r-2^q(w-4m),r,4m)\cdot(\Psi(r,w,m,q)^2+\mu^*(r,q,2m))\Big)\Bigg\}.\end{aligned} \tag{5.26}$$

Proof. In the expression, q corresponds to the number of squarings and m corresponds to an assumed number of collisions in each of the two polynomials. The expected number of instances that we have to exhaust until we find a pair of polynomials that forms a codeword which has weight $w-4m$ is

$$\left(\sum_{k+l=2m}\Psi(r,w,k,q)\cdot\Psi(r,w,l,q)\right)^{-1} \leqslant (\Psi(r,w,m,q))^{-2}. \tag{5.27}$$

Whenever an instance is attacked, we spend $C_{\mathsf{ISD}}(2r/2^q,r/2^q,w-4m))$ bit operations on recovering a codeword of weight $w-4m$.

Assuming that the algorithm succeeds to recover one or several codewords of weight $w-4m$, we expect to perform $\mu^*(r,q,w-4m)$ such reconstructions

to full (or a higher) dimension. Reconstruction of a single candidate costs $C_{\mathsf{ISD}}(2r - 2^q(w - 4m), r, 4m)$ bit operations, with an additional but negligible cost for solving the linear system of equations. By putting it all together, we obtain the complexity result stated in the theorem. ■

5.4 RESULTS

In this section, we give a detailed complexity analysis for the proposed parameters found in Table 4.1. The complexity for finding a codeword corresponding to squared the squared polynomials $h_0(x)^{2^q}$ and $h_1(x)^{2^q}$ is denoted C_1, while C_2 denotes the reconstruction complexity. When calculating the complexity of the given attack, we use simulated values Ψ_{sim} that verify the success probability Ψ.

Instance							Complexity ($\log_2$)		
r	w	λ	q	m	Ψ^2_{sim}	μ^*	C_1	C_2	C^*
4800	90	80	4	6	$2^{-8.108}$	0.01	72.58	51.87	80.69
				7	$2^{-11.32}$	$2^{-18.77}$	68.10	55.51	79.42
				8	$2^{-15.30}$	$2^{-31.36}$	63.72	59.18	79.02
			5	13	$2^{-25.41}$	$2^{-8.57}$	45.96	69.63	72.75†
9856	142	128	4	8	$2^{-11.87}$	$2^{-88.51}$	115.8	61.83	127.7
32768	274	256	5	13	$2^{-11.43}$	$2^{-18.38}$	236.6	51.87	248.0

Table 5.1: Attack complexity in bit operations targeting a QC-MDPC McEliece scheme. The complexity marked with † refers to message-recovery.

- **Security level** $\lambda = 80$: A standard attack using Stern's algorithm requires $2^{94.35}$ bit operations in complexity, using parameters $p = 4$ and $j = 49$. In Table 5.1, we have listed some possible choices of parameters when $q = 4$. With $m = 6$ (probability distribution given in Figure 5.2), the attack has a relatively low complexity (see Figure 5.3), while retaining the expected number of public keys required to a few ($2^{8.108} \approx 276$), making it suitable for a key-recovery attack. The complexity $C_1 = 2^{72.58}$ is that of Stern's algorithm, with parameters $p = 4$ and $j = 28$. Since the expected number bit operations spent on reconstructing false candidates are negligible ($\mu^* \cdot C_2 \ll C_1$), the total com-

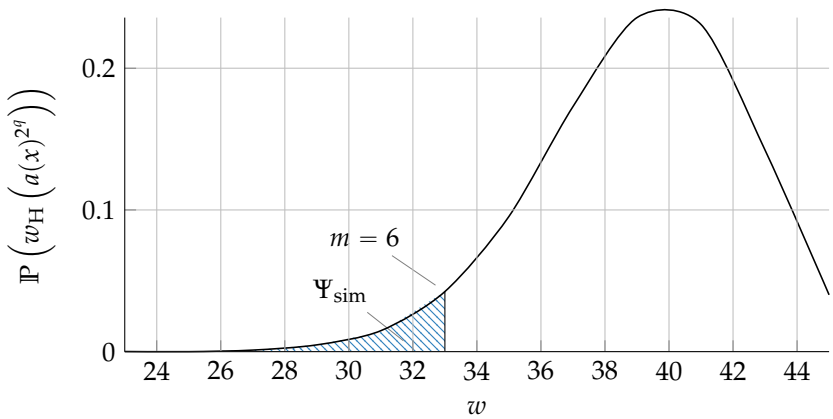


Figure 5.2: The simulated polynomial weight distribution with initial polynomial weight $w = 45$, $q = 4$ and $m \geqslant 6$.

plexity C^* is $\Psi_{\mathrm{sim}}^2 \cdot C_1 = 2^{80.69}$ bit operations, which is an improvement by a factor $2^{13.66}$ over the standard approach. With the algorithm in [BJMM12] we found that C_1 is $2^{69.79}$ with parameters $p = 8$, $l = 33$ and $\delta = 0$, which gives an overall attack complexity of $2^{77.90}$, thus breaking the 80-bit security.

Moreover, we have chosen a set of parameters to target a message-recovery attack, i.e., we want to have as low complexity as possible, with no constraints on the number of intercepted ciphertexts required. To achieve low complexity, we set the weight constraints very low in a small dimension, with $m = 13$ and $q = 5$, making the first step computationally cheap. Here, $\Psi_{\mathrm{sim}}^2 = 2^{-25.41}$. With ISD parameters $p = 2$ and $j = 11$, we get a complexity of $C_1 = 2^{45.96}$ bit operations. Since μ^* is as low as $2^{-8.57}$, we expect that any found codeword of weight $2 \cdot 16$ is the correct one. Instead of going directly up to full dimension (which would be too computationally costly), we choose to reconstruct to dimension 600×1200. By doing so, we obtain a $[1072, 600]$ linear code in which we recover a codeword of weight 48. With parameters $p = 4$ and $j = 34$, the complexity of this step is $C_2 = 2^{69.63}$. In order to succeed, we require that

$$w_{\mathrm{H}}\left(a(x)^{2^3}\right) + w_{\mathrm{H}}\left((a(x)P(x))^{2^3}\right) \leqslant 80,$$

which holds with probability $\geqslant \frac{1}{2}$. In total, we have in complexity $(\frac{1}{2})^{-1} \cdot (2^{25.41} \cdot 2^{45.96} + 2^{69.63}) \approx 2^{72.75}$. The standard (message-recovery) attack with Stern's algorithm on a $[9600, 4800]$ linear code recovering a

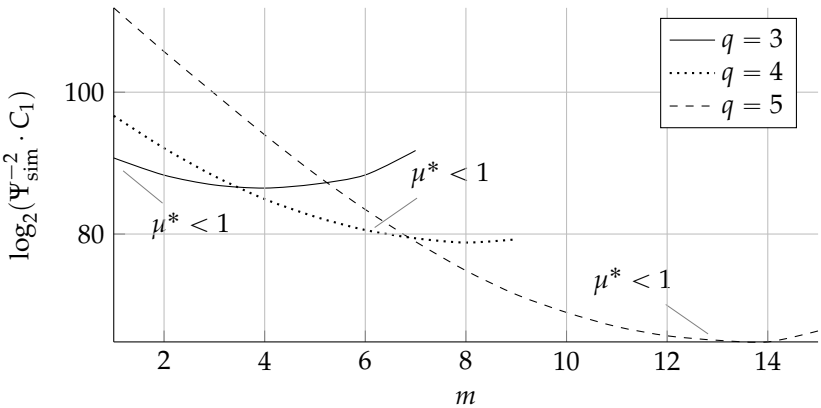


Figure 5.3: The complexity of the key-recovery attack targeting a QC-MDPC McEliece scheme with 80-bit security parameters $r = 4800$ and $w = 90$, using various attack parameters.

weight $t = 84$ codeword is $2^{101.3}$, with $p = 4$ and $j = 49$. Our attack gives an impressive improvement factor of $2^{28.55}$, requiring only $2^{25.41}$ intercepted ciphertexts.

- **Security level** $\lambda = 128$: For the parameters $r = 9856$ and $w = 142$, targeting 128-bit security, we achieve a attack complexity of $2^{127.7}$ with $q = 4$. First, we try recovering a codeword of weight 110 in a $[1232, 616]$ linear code. To obtain such an instance, we need on average $2^{11.87}$ public keys. Moreover, we spend $C_1 = 2^{115.8}$ operations on each instance, with parameters $p = 4$ and $j = 34$. Reconstruction consists of recovering a weight 32 codeword in a $[17952, 9856]$ linear code, which costs $2^{61.83}$ with parameters $p = 2$ and $j = 30$. The improvement over the standard attack is about $2^{19.25}$.

- **Security level** $\lambda = 256$: In the 256-bit security case, we have $r = 32768$ and $w = 274$. We begin by squaring $q = 5$ times, and then try to recover a weight 218 codeword in the $[2048, 1024]$ linear code, which costs $2^{236.6}$ with parameters $p = 6$ and $j = 49$, and requires around $2^{11.43}$ public keys. Finally, we recover a weight 52 codeword in the $[29216, 16384]$ linear code. Altogether, we gain an improvement factor of $2^{31.37}$ over the standard attack.

Note: With a large number of intercepted ciphertexts, an attack targeting multiple instances that are encrypted with the same key, will potentially yield

further improvement [Sen11], both in the standard attack and in the new attack. In the complexity results above, we did not take such approaches into consideration, but we could potentially gain a factor $\sqrt{N}$, with N being the number of intercepted ciphertexts.

5.5 SUMMARY

In this chapter, we have presented a new cryptanalytic method that exploits the property that the length of the public-key matrix in the QC-MDPC variant of McEliece PKC is divisible by a factor of 2. As shown, this allows for a reduction of the problem, which gives a lower attack complexity than claimed in [MTSB13]. The presented attack is very general and can be applied to any quasi-cyclic code having circulant sub-matrices of even dimension. We conclude that using an odd dimension is probably a better choice, as it avoids the squaring attack.

6

Searching for Low-Weight Polynomial Multiples

Opportunities multiply, as they are seized.
– *Sun Tzu*

This chapter introduces the problem of finding a *low-weight polynomial multiple*, which is tightly related to the minimum-distance problem. We present two different algorithms for solving the low-weight polynomial multiple problem under different circumstances. The results were originally proposed in [JL13] [LJ14]. We haven chosen to restrict our analysis to the binary case, but the presented algorithms can be used to solve the problem over any field $\mathbb{F}_q$.

Finding a low-weight multiple $K(x)$ of a binary polynomial $P(x)$ is believed to be a computationally hard problem. As of today, no known polynomial-time algorithm exists. Although it is not known how hard the problem is, one can quite easily show that it is no harder than finding a low-weight codeword in a random code (MDP), as there is a simple reduction to this problem. For the latter problem, any ISD algorithm, e.g., Stern's algorithm, can be used (see Chapter 3).

The problem of finding a low-weight multiple is of great importance in general in the field of cryptography. Some other applications are distinguishing attacks and correlation attacks on stream ciphers [MS89] [CT00]. Another area is finite field arithmetics. The results we present in this chapter apply equally well to these areas and improve state-of-art also here, even though we do not explicitly give the details.

- The first algorithm applies to the general low-weight polynomial multiple problem. Our algorithm has lower complexity than existing approaches for solving this problem. In analysis, we focus on the actual

computational complexity of the algorithm and not the asymptotics, similar to previous work in this area [MMT11] [JL11] [BJMM12]. We use coding-theoretic methods; the given polynomial can be used to define a code which has a cyclic structure. A main step in the proposed algorithm is to construct a new code with slightly higher dimension, having a larger number of low-weight codewords. Now finding a single one of them leads to the recovery of the low-weight polynomial multiple. The new algorithm also includes some additional processing that reduces the minimum distance of the code. Applying an algorithm for finding low-weight codewords on the constructed code yields a lower complexity for finding low-weight polynomial multiples compared to previous approaches. As an application, the algorithm is used to attack the public-key cryptosystem TCHo [FV07]. A consequence of the new algorithm is that the gap between claimed security level and actual algorithmic complexity of a key-recovery attack on TCHo is narrowed in all suggested instances. For some parameters, the complexity might be interpreted as below the security level.

- As a second main result in this chapter, we consider the problem of finding a polynomial multiple of weight 4, which is a very relevant problem in correlation attacks on stream ciphers. If a stream cipher uses a linear feedback shift register (LFSR) with feedback polynomial $P(x)$, we are often interested in finding one or many multiples of $P(x)$ of very low weight, typically 3, 4, or 5.

  Using similar ideas as described above, we present a new probabilistic algorithm for finding a multiple of weight 4, which is faster than all previous approaches. This will improve efficiency in the pre-computation phase in correlation attacks using weight 4. The algorithm works for larger even weights as well, but the asymptotic gain is less.

6.1 THE LOW-WEIGHT POLYNOMIAL MULTIPLE PROBLEM

Let us start by introducing the problem of finding all low-weight polynomial multiples as follows:

Problem 6.1 Given polynomial $P(x) \in \mathbb{F}_2[x]$ of degree d_P and two integers w and d, find all multiples $K(x) = P(x)Q(x)$ of weight at most w and degree at most d.

A related but different problem emerges when it is sufficient to find one single solution among possibly several solutions is as follows.

Problem 6.2 Given a polynomial $P(x) \in \mathbb{F}_2[x]$ of degree d_P and integers w and d, find a (if it exists) multiple $K(x) = P(x)Q(x)$ of weight at most w and degree at most d.

A major difference between these two problems lies in the fact that generalized birthday arguments [Wag02] sometimes can be used in Problem 2 whereas it is usually not applicable to Problem 1, as this technique does not necessarily find all possible solutions. It is also a question of the nature of the underlying problem giving rise to the low-weight polynomial multiple problem. In some cases a single multiple is enough and in other cases one is interested in finding many. We can also imagine a problem formulation where we output T multiples instead of all.

Another deviating point is the expected number of multiples. A rough estimation gives the expected number of low-weight multiples to be around

$$\frac{d^{w-1}}{(w-1)! \cdot 2^{d_P}}. \tag{6.1}$$

We may then have a set of instances where the expected number of low-weight multiples is very low, but we know from construction that such a multiple does exist. The second scenario is when the expected number of multiples is larger and the problem instance perhaps is a random instance ($P(x) \in \mathbb{F}_2[x]$ of degree d_P is randomly chosen among all such polynomials).

Looking at known algorithms for solving the low-weight polynomial multiple problem, the techniques differ depending on whether we are considering a fixed very small weight w, typically $w = 3, 4, 5$, or whether we are considering larger values of the weight w.

Undoubtedly, there are many flavors of this problem. We consider first the case relevant to TCHo. Thus, we give another modified problem formulation that fits the TCHo case.

Problem 6.3 (Low-Weight Polynomial Multiple**)** Given a polynomial $P(x) \in \mathbb{F}_2[x]$ of degree d_P and two integers w and d, find a (if it exists) multiple $K(x) = P(x)Q(x)$ of weight exactly w and degree exactly d. We denote the problem LWPM.

Let us give a very brief overview of previous algorithms for these problems. Several algorithms have a large initial cost. Thus, for some parameters an exhaustive search will have the best complexity.

6.1.1 TIME–MEMORY TRADE-OFF APPROACH

There exists a plethora of variations and improvements of this method. Among these, we find for instance the approach by Golić [Gol96], which we refer to

as the standard approach. Let d be the highest degree allowed. The algorithm formulated by Golić searches for polynomials of weights $w = 2j$ (and $w = 2j - 1$). The initial step consists of creating a list that contains the $\binom{d}{j}$ residues of the form $x^{i_1} + x^{i_2} + \cdots + x^{i_j} \bmod P(x)$, for $0 \leqslant i_1 < i_2 < \cdots < i_j < d$. These residues can be efficiently represented as integers, on which it is straightforward to apply a sort-and-match procedure. Any collision gives rise to a polynomial of weight $w = 2j$ being a multiple of $P(x)$. The algorithm requires time and memory in the order of $\binom{d}{j}$, which is roughly $d^{w/2}$ for w even.

Canteaut and Trabbia [CT00] introduced a memory-efficient method for finding multiples of weight $w + 1$. They first compute the residues $q_i(x) = x^i \bmod P(x)$ for $0 < i < d$ and store the exponent i in a table indexed by $q_i(x)$. After this, all sums of $w - 1$ elements in the table are computed forming $A = 1 + q_{i_1}(x) + \cdots + q_{i_{w-1}}(x)$. The last element x^j is accessed in the table from position A, and thus, $1 + q_{i_1}(x) + \cdots + q_{i_{w-1}}(x) + x^j$ is a multiple of $P(x)$. The time complexity is approximately $\binom{d-1}{w-1}$ and requires only linear memory.

Another approach is the match-and-sort approach by Chose, Joux and Mitton [CJM02]. Using a divide-and-conquer technique, the task of finding collisions in a search space of size n^w, is divided into smaller tasks. This is done by searching for collisions in smaller subsets with less restrictions. The solutions to the smaller subproblems are then sorted and put together to solve the whole problem. This approach has time complexity of about $d^{\lceil w/2 \rceil} \cdot \log d$ and requires $d^{\lceil (w-1)/4 \rceil}$ of space.

In [DLC07], Didier and Laigle-Chapuy consider using discrete logarithms instead of the direct representation of the involved polynomials to improve performance. The complexity for their approach is $\mathcal{O}(d)$ logarithm computations over $\mathbb{F}_2[x]/P(x)$ and approximately the same amount of memory.

The LWPM problem can also be viewed as a subset-sum (or knapsack) problem. Therefore all generic algorithms for solving the subset-sum problem can be used to solve LWPM (e.g. all birthday-type algorithms given in [Jou09]).

When the degree of the multiple can be large and there are many low-weight multiples, but it is sufficient to find only one, Wagner's generalized birthday approach [Wag02] becomes efficient. To expect to find a multiple using Wagner's algorithm, it is required that $d \approx 2^{n/(\lfloor \log w \rfloor + 1)}$. The time needed to perform such a search is approximately $w \cdot 2^{n/(\lfloor \log w \rfloor + 1)}$.

6.1.2 FINDING MINIMUM-WEIGHT CODEWORDS IN A LINEAR CODE

The low-weight polynomial multiple problem can be reduced to the function problem version of MDP, for which we can use information-set decoding al-

gorithms. A common reduction is as follows. Let

$$P(x) = p_0 + p_1 x + \cdots + p_{d_P} x^{d_P} \tag{6.2}$$

be the given polynomial and let

$$\mathbf{u} = \begin{pmatrix} u_0 & u_1 & \cdots & u_{d-d_P} \end{pmatrix} \tag{6.3}$$

be a length $d - d_P + 1$ binary vector. One can formulate the problem of finding a weight w polynomial $K(x)$ of degree $\leqslant d$, being a multiple of $P(x)$, as finding a binary vector $\mathbf{u}$ such that $\mathbf{u}\mathbf{G}(x)$ has exactly w non-zero coefficients, where

$$\mathbf{G}(x) = \begin{pmatrix} P(x) \\ xP(x) \\ \vdots \\ x^{d-d_P} P(x) \end{pmatrix}. \tag{6.4}$$

Writing also the polynomials as length d vectors, the problem reduces to finding a weight w codeword in the binary linear code generated by the Toeplitz generator matrix

$$\mathbf{G} = \begin{pmatrix} p_0 & p_1 & \cdots & p_{d_P} & & & \\ & p_0 & p_1 & \cdots & p_{d_P} & & \\ & & \ddots & \ddots & & \ddots & \\ & & & p_0 & p_1 & \cdots & p_{d_P} \end{pmatrix}, \tag{6.5}$$

having dimension $(d - d_P + 1) \times (d + 1)$, where the empty spaces correspond to zero elements. As the problem is reduced to finding a weight w codeword in the linear code generated by $\mathbf{G}$, conventional information-set decoding algorithms that find minimum weight codewords of the code can be used.

6.2 A NEW ALGORITHM SOLVING LWPM

Let us describe the main ingredients in the new algorithm. We use the established technique from coding theory described in Subsection 6.1.2, but we introduce some new modifications.

Recall that $K(x)$ represents the low-weight polynomial multiple we are looking for. Having the generator matrix $\mathbf{G}$ described in (6.5) in mind, our initial observation is that one can increase success probability in each iteration of the information-set decoding part by a factor $y + 1$ by allowing y shifts of the polynomial $K(x)$, i.e., including the polynomial multiples

$$xK(x), x^2K(x), \ldots, x^yK(x)$$

along with $K(x)$ in the solution space (this will be proved in Section 6.2.1). The trade-off is that the dimension of the generator matrix grows to $(d - d_P + 1 + y) \times (d + 1 + y)$. The new generator matrix will have the following structure:

$$\mathbf{G}_y = \begin{pmatrix} p_0 & p_1 & \cdots & p_{d_P} & & & & & & \\ & p_0 & p_1 & \cdots & p_{d_P} & & & & & \\ & & \ddots & \ddots & & \ddots & & & & \\ & & & p_0 & p_1 & \cdots & p_{d_P} & & & \\ & & & & p_0 & p_1 & \cdots & p_{d_P} & & \\ & & & & & \ddots & \ddots & & \ddots & \\ & & & & & & p_0 & p_1 & \cdots & p_{d_P} \end{pmatrix}. \tag{6.6}$$

The marked rectangle represents $\mathbf{G}$ and everything outside represents the expansion. Let the expanded matrix in (6.6) be denoted $\mathbf{G}_y$ and the code it generates be denoted $\mathcal{C}_y$.

Recall that the unknown low-weight polynomial is written in the form

$$K(x) = 1 + k_1 x + \cdots + k_{d-1} x^{d-1} + x^d, \tag{6.7}$$

i.e. the polynomial $K(x)$ has degree d. We will now show how to exploit the form of the polynomial.

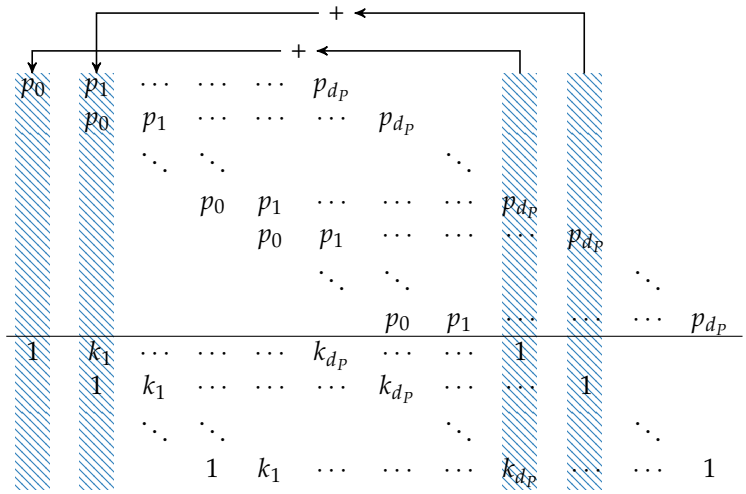
Theorem 6.4 For any polynomial $P(x)$, there exists a linear map $\mathbf{\Gamma}$ that transforms the code $\mathcal{C}_y$ into a new code given by $\mathbf{G}_y\mathbf{\Gamma}$, such that all weight w codewords corresponding to shifts of $K(x)$ will have weight $w - 2$ in the new code[1].

Proof. Given that $K(x)$ has degree d, its constant term and the coefficient of x^d are non-zero. Combining the corresponding columns in $\mathbf{G}_y$, i.e., adding the $(d+1)$th column to the first column of $\mathbf{G}_y$ and then removing the $(d+1)$th column from $\mathbf{G}_y$, will cause the codeword corresponding to $K(x)$ to decrease by two in weight (see Figure 6.1). The new codeword stemming from $K(x)$ will have weight $w - 2$.

Note that the symbols in the other weight w codewords will be permuted, but the weight of these codewords stays the same. We can repeat this for the second column, by adding the $(d+2)$th column and so on up to the yth and $(d+y)$th column. The consequence of the approach just outlined is that all codewords that correspond to shifts of $K(x)$ will have weight $w - 2$. It is easy to see that the operations described above can be expressed as a right multiplication by a matrix $\mathbf{\Gamma}$, giving the new generator matrix $\mathbf{G}_y\mathbf{\Gamma}$.

■

[1]This operation is equivalent to transforming $K(x)$ into $K(x)$ mod $(1 + x^d)$.



$$\xrightarrow{\boldsymbol{\Gamma}} \begin{array}{cccccc}
p_0 & p_1 & \cdots & p_{d_P} & & \\
 & p_0 & & \vdots & \ddots & \\
 & & \ddots & \vdots & & p_{d_P} \\
p_{d_P} & & & p_0 & \cdots & p_{d_P-1} \\
\vdots & \ddots & & & \ddots & \vdots \\
p_0 & \cdots & p_{d_P} & & & p_0 \\
\hline
0 & k_1 & \cdots & \cdots & \cdots & k_{d-1} \\
k_{d-1} & 0 & k_1 & \cdots & \cdots & k_{d-2} \\
\vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\
k_{d-y} & \cdots & k_{d-1} & 0 & k_1 & k_2
\end{array}$$

Figure 6.1: The upper part of the figure illustrates how the columns in $\mathbf{G}_y$ are added to form $\mathbf{G}_y\boldsymbol{\Gamma}$ and how weight-w codewords are transformed into weight $w-2$ codewords. The lower part of the figure shows the resulting generator matrix.

The matrix product $\mathbf{G}_y\boldsymbol{\Gamma}$ forms a new generator matrix of dimension $(d - d_P + 1 + y) \times (d+1)$, as illustrated in Figure 6.1. The final step is to apply information-set decoding on the code formed by $\mathbf{G}_y\boldsymbol{\Gamma}$, looking for codewords

Algorithm 10 (Solve-LWPM)

Input: Polynomial $P(x) \in \mathbb{F}_2[x]$ and algorithm parameters $w, y \in \mathbb{N}$.

Output: A polynomial multiple $K(x)$ of weight $w \in \mathbb{N}$.

1. From $P(x)$, create the corresponding generator matrix $\mathbf{G}$ according to (6.5);
2. Expand $\mathbf{G}$ by y extra rows and y extra columns, yielding in total $y+1$ codewords that represent $K(x)$, all of weight w. Let the expansion be $\mathbf{G}_y$;
3. Transform the codewords that represent $K(x)$ to weight $w-2$, by forming the generator matrix $\mathbf{G}_y\mathbf{\Gamma}$, in agreement with Theorem 6.4;
4. Input $\mathbf{G}_y\mathbf{\Gamma}$ into Algorithm 5, using optimum parameters with respect to (6.8), to find one codeword $\mathbf{u}$ among the $y+1$ weight w codewords that represent $K(x)$;
5. From $\mathbf{u}$, construct $K(x)$ by exhaustive search over at most $y+1$ polynomials and output $K(x)$;

of weight $w-2$. For the information-set decoding step, we use Algorithm 5. The explicit complexity expressions are given in (3.33). The algorithm for solving Problem 6.3 can be summarized as given in Algorithm 10.

6.2.1 COMPLEXITY ANALYSIS

In order to estimate the complexity of the algorithm solving LWPM, we use previously established results given in Chapter 3. Let C_{ISD} denote the expected number of binary operations performed by Algorithm 5 to find a codeword of weight w in an (n,k)-code $\mathcal{C}$, assuming that exactly one such codeword exists.

For an expanded (n,k) code $\mathcal{C}_y$ (according to (6.6)) with weight reduction by $\mathbf{\Gamma}$, we have $(n,k) = (d+1, d-d_P+1+y)$. Running Algorithm 10 on $\mathbf{G}_y\mathbf{\Gamma}$ will require complexity according to the following theorem.

Theorem 6.5 Algorithm 10 has an expected complexity given by

$$C^* = \min_{y \geqslant 0} \frac{C_{\mathsf{ISD}}(d+1, d-d_P+1+y, w-2)}{y+1}, \tag{6.8}$$

when the success probability of one iteration of Algorithm 5 is small.

Proof. Since the codewords are shifts of each other, we cannot conclude that they are independent. Therefore, it is not possible to apply Lemma 3.3. Instead, we need to analyze the behavior of the ISD algorithm to show finding

one codeword is independent of finding another one. Although the analysis apply to most ISD algorithms, we consider Algorithm 5 in our analysis.

The complexity function $C_{\mathsf{ISD}}(n, k, w)$ refers to the expected complexity of running Algorithm 5 with an instance where we have one single solution, i.e., only one codeword of weight w exists in the code, whereas in the case of LWPM, there will exist several weight w codewords. Having $y + 1$ possible solutions instead of one suggests that finding at least one is roughly $y + 1$ times more likely. However, for this to be true, the probability of finding one single codeword in one iteration, denoted ξ, must be small. In particular, we require that $y \cdot \xi \ll 1$. Secondly, we require the events of finding the shifted low-weight codewords to be independent of each other.

Let the set of solutions, i.e., the set of shifts of $K(x)$ represented as vectors, be the rows of the matrix

$$\mathbf{K} = \begin{pmatrix} 1 & k_1 & k_2 & \cdots & \cdots & \cdots & k_{d-1} & 1 & & & \\ & 1 & k_1 & k_2 & \cdots & \cdots & \cdots & k_{d-1} & 1 & & \\ & & \ddots & \ddots & \ddots & & & & \ddots & \ddots & \\ & & & 1 & k_1 & k_2 & \cdots & \cdots & \cdots & k_{d-1} & 1 \end{pmatrix}. \tag{6.9}$$

The weight reduction of the expanded generator matrix $\mathbf{G}_y$ will result in a new codeword matrix, which we write as

$$\mathbf{K\Gamma} = \begin{pmatrix} 0 & k_1 & k_2 & \cdots & \cdots & \cdots & \cdots & k_{d-1} \\ k_{d-1} & 0 & k_1 & k_2 & \cdots & \cdots & \cdots & k_{d-2} \\ \vdots & & \ddots & \ddots & \ddots & & & \vdots \\ k_{d-y} & \cdots & k_{d-1} & 0 & k_1 & k_2 & \cdots & k_{d-y-1} \end{pmatrix}. \tag{6.10}$$

The column permutation π acting on $\mathbf{G}_y\mathbf{\Gamma}$ (used in Algorithm 5) permutes all codewords accordingly, thus permuting the columns of $\mathbf{K}$. Let $r = d - d_P$. For a set of arbitrary indices $\{i_1, i_2, ..., i_r\} \subset \{0, 1, ..., d-1\}$, the resulting permutation of $\mathbf{K\Gamma}$ is

$$\pi(\mathbf{K\Gamma}) = \left(\begin{array}{cccc|c} k_{i_1} & k_{i_2} & \cdots & k_{i_r} & \\ k_{i_1-1} & k_{i_2-1} & \cdots & k_{i_r-1} & \cdots \\ \vdots & \vdots & & \vdots & \\ k_{i_1-y} & k_{i_2-y} & \cdots & k_{i_r-y} & \end{array}\right). \tag{6.11}$$

Now, assume that the two vectors

$$\mathbf{k} = \begin{pmatrix} k_{i_1} & k_{i_2} & \cdots & k_{i_r} \end{pmatrix} \text{ and } \mathbf{k}' = \begin{pmatrix} k_{i_1-j} & k_{i_2-j} & \cdots & k_{i_r-j} \end{pmatrix} \tag{6.12}$$

constitute two rows of the first r columns of $\pi(\mathbf{K\Gamma})$ for some j such that $1 \leqslant j \leqslant y$ and where each k_i is an i.i.d. random variable. Note that the indices are taken modulo d. For a codeword $\mathbf{k}$ to be considered as a possible solution in one iteration of Algorithm 5, a necessary but not sufficient condition is that $\mathbf{k}$ can have at most $2p$ non-zero elements in the first r columns. We want to show that these two events are »approximately independent«. We provide some informal argument. A more formal derivation would require quite some space, which we avoid.

The set of indices $\{i_1, i_2, ..., i_r\}$ are chosen uniformly in the permutation. As a consequence, there is a non-zero probability that $\{i_1, i_2, ..., i_r\} \cap \{i_1 - j, i_2 - j, ..., i_r - j\} \neq \emptyset$, meaning that one or several random variables in $\mathbf{k}$ and $\mathbf{k}'$ are identical. More specifically, we have

$$\mu \stackrel{\text{def}}{=} \mathbb{E}\left(|\{i_1, i_2, ..., i_r\} \cap \{i_1 - j, i_2, ..., i_r - j\}|\right) \tag{6.13}$$

common indices in $\{i_1, i_2, ..., i_r\}$ and $\{i_1 - l, i_2 - j, ..., i_r - j\}$ on average.

The probability of having i overlapping variables describes the probability function of a hypergeometric distribution, i.e.,

$$\mathbb{P}\left(j \text{ overlapping variables}\right) = \frac{\binom{r}{j}\binom{d-r}{r-j}}{\binom{d}{j}} \tag{6.14}$$

and thus,

$$\mu = \frac{r^2}{d}. \tag{6.15}$$

Let A_0 denote the sum of random variables in positions $i_1, i_2, ..., i_r$ and A_1 the sum of random variables in positions $i_1 - j, i_2 - j, ..., i_r - j$ respectively. Moreover, let B denote the sum of the intersecting variables in positions $\{i_1, i_2, ..., i_r\} \cap \{i_1 - j, i_2 - j, ..., i_r - j\}$.

The expected intersection is μ, so B is a sum of μ random variables. By assuming the worst-case $A_0 = 2p$, we obtain the following

$$\begin{aligned}
\mathbb{E}\left(A_1 \mid A_0 = 2p\right) = \; & \mathbb{E}\left(B \mid A_0 = 2p\right) + \mathbb{E}\left(A_1 - B \mid A_0 = 2p\right) \\
=& 2p \cdot \frac{\mu}{r} + (w - 2p) \cdot \frac{r - \mu}{d - r} = 2p \cdot \frac{r}{d} + (w - 2p) \cdot \frac{r - \frac{r^2}{d}}{d - r} \\
=& 2p \cdot \left[\frac{r \cdot (d - r) - d \cdot \left(r - \frac{r^2}{d}\right)}{d \cdot (d - r)}\right] + w \cdot \frac{r \cdot \left(1 - \frac{r}{d}\right)}{d \cdot \left(1 - \frac{r}{d}\right)} = w \cdot \frac{r}{d} \\
=& w \cdot \left(1 - \frac{d_P}{d}\right).
\end{aligned} \tag{6.16}$$

Hence, if $w \cdot (1 - \frac{d_P}{d}) \gg 2p$ then the expected value is significantly larger than $2p$. If so, A_1 is very unlikely to take a value below or equal to $2p$ and, thus, we argue that the events of finding the shifted codewords are approximately independent.

Under the two conditions $y \cdot \xi \ll 1$ and $w \cdot (1 - \frac{d_P}{d}) \gg 2p$, we can conclude that the probability of finding at least one out of $y + 1$ codewords is $1 - (1 - \xi)^{y+1} \approx (y + 1) \cdot \xi$, since all codewords are equally likely to be found. Moreover, the complexity $C_{\mathsf{ISD}}(, n, k, w)$ is $\mathcal{O}(\xi^{-1})$ and therefore Algorithm 10 has complexity

$$\mathcal{O}\left((y + 1)^{-1} \cdot \xi^{-1}\right). \tag{6.17}$$

This concludes the proof of Theorem 6.5.

■

6.2.2 SIMULATION RESULTS

To check the correctness of the algorithm and the complexity analysis, we have conducted some simulations. We consider a toy example of LWPM, running Algorithm 10 on an instance with a polynomial

$$\begin{aligned} P(x) = \; & 1 + x^1 + x^3 + x^6 + x^7 + x^8 + x^{10} + x^{11} + x^{16} + x^{20} + \\ & x^{25} + x^{28} + x^{29} + x^{32} + x^{33} + x^{34} + x^{37} + x^{38} + x^{39}. \end{aligned}$$

We are seeking a weight $w = 8$ multiple of $P(x)$ of degree 62. The solution is

$$K(x) = 1 + x + x^2 + x^4 + x^{11} + x^{36} + x^{37} + x^{62}.$$

In Figure 6.2, we are plotting the simulated success rate of each iteration of Algorithm 5 as a function of codeword multiplicity $y + 1$. The solid line shows the theoretical success probability function, according to (6.8). The triangle-shaped points show the simulated success probability. The square-shaped points show the simulated success probability of a single iteration when using the weight-reduction technique described in Theorem 6.4. Looking at Figure 6.2, we note that the probability increases by the factor $y + 1$. We also note that initially the lines are almost linear, but bent as the probability converges to 1.

In Figure 6.3, we are plotting the simulated number of operations in one iteration of Algorithm 5 (squared-shaped points). We note that the simulated operation count follows the same curve as the theoretical expected operation count (solid line).

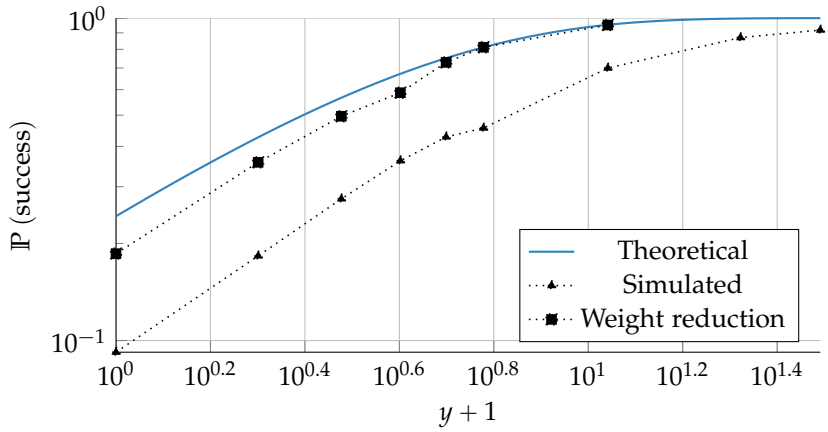


Figure 6.2: The probability of success in each iteration as a function of codeword multiplicity.

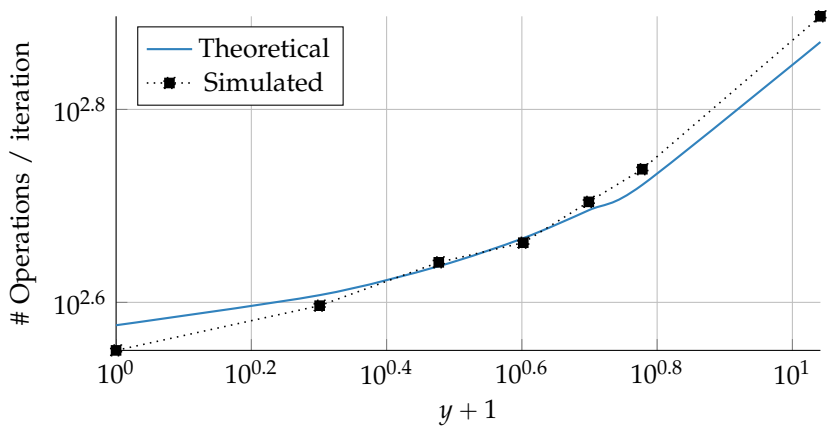


Figure 6.3: Operation count in each iteration as a function of codeword multiplicity.

6.2.3 THE ATTACK

The security of TCHo relies on the hardness of finding the secret polynomial multiple $K(x)$, given only the public polynomial $P(x)$. It is clear that solving LWPM for the instance $P(x)$ would result in a key-recovery attack. In [FV07] and [AvzG07], some methods are proposed to solve the LWPM problem. These methods are, however, unable to break TCHo.

By running Algorithm 10 on the instances proposed in [FV07], we get the

complexities presented in Table 6.1. According to [FV07], these instances are designed for a security level of 2^{80}. Note that the other algorithms mentioned in Section 6.1 have much higher complexity.

Instance			Complexity ($\log_2$)			Opt.
d	d_P	w	Stern	Algorithm 5	Algorithm 10	y_{opt}
25820	7000	45	100.69	100.08	90.61	200
24730	12470	67	85.56	85.75	77.65	230
44677	4433	25	97.48	96.47	84.15	250
24500	8000	51	98.91	98.45	89.48	200
17600	8795	81	97.43	96.51	89.21	110
31500	13200	65	99.84	99.80	91.13	250

Table 6.1: The first complexity column gives the complexity of using Stern's algorithm applied on (6.5), the second column for using Algorithm 5 applied on (6.5), i.e. $y = 0$, and the third using Algorithm 10. The optimum column gives the optimal choice of y in Algorithm 10.

From a more technical perspective, one can consider the complexity in word operations instead of single bit operations. This would decrease all the complexities above by approximately a factor 2^6 and would give a rough estimate of the required number of clock cycles. In implementation, we managed to perform on average $2^{3.3}$ bit operations per clock cycle.

Example 6.1 *Consider the case $d = 44677$, $d_P = 4433$ and $w = 25$ from Table 4.2. By minimization of (6.8) over y, p, l and z, we obtain the values $y_{\text{opt}} = 250$, $p_{\text{opt}} = 4$, $l_{\text{opt}} = 51$ and $z_{\text{opt}} = 18$. The generator matrix $\mathbf{G}_y\mathbf{\Gamma}$ has dimension 40495×44678 and 251 codewords of weight 23. Using Algorithm 10, we get a complexity of about 2^{84} bit operations.*

6.3 A NEW ALGORITHM FOR FINDING A WEIGHT-4 MULTIPLE

Inspired by the ideas contained in the previous sections, we turn our attention to a different problem. We already mentioned in the introduction that an important cryptographic application for algorithms finding low-weight multiples of binary polynomials is the area of correlation attacks. In particular, given a feedback polynomial $P(x)$ for a linear feedback shift register used in some stream cipher, one is interested in finding low-weight multiples of $P(x)$ as this can often be used in a correlation attack. The success of the correlation

attack (its complexity) is related to the weight of the multiple and typically grows exponentially with the weight. Hence, we are almost only considering such low weights as 3, 4, 5, or slightly larger.

If we find a low-weight multiple of a certain degree, this means in the correlation attack that we need to obtain an output sequence from the stream cipher of length at least the same as the degree of the multiple. So clearly, we are interested in finding the weight w multiples of lowest degree. These circumstances motivate the problem instance Problem 2 if we choose the maximum degree d for the multiple to some suitable value.

From a practical point of view, in a correlation attack, we do not need the algorithm to succeed every time, nor do we need to deliver exactly all multiples in our answer. With this as a background, we look at the following specific problem instance which is clearly very relevant for a correlation attack.

Problem 6.6 (Weight-4 Polynomial Multiple**)** Given a polynomial $P(x) \in \mathbb{F}_2[x]$ of degree d_P, find the weight-4 multiple $K(x) = P(x)Q(x)$ of lowest degree. We denote the problem W4PM.

We observe that the generalized birthday approach is not applicable here, as it is then very likely that we will not find the weight 4 multiple with lowest degree. The algorithms we know to solve this problem include the time-memory trade-off by Golić from Section 6.1.1.

We first note that for a randomly selected polynomial $P(x)$ of degree d_P, the expected degree of the desired weight-4 multiple of $P(x)$ is around $d = 2^{d_P/3}$.

In the time-memory approach, we would then compute all residues of the form $1 + x^{i_1} \bmod P(x)$, for $0 \leqslant i_1 < 2^{d_P/3}$, and store them all in a table sorted according to the value of the residue.

Then we would compute all residues of the form $x^{i_1} + x^{i_2} \bmod P(x)$, for $0 \leqslant i_1 < i_2 < 2^{d_P/3}$ and for each residue we compute we check if it is already in the table. When such an event occurs, we have found a weight-4 multiple. If we find many, we output the one with lowest degree. If we do not find any weight-4 multiple, we run again with a slightly larger d.

Clearly, the computational complexity of this approach is around $2^{2d_P/3}$ and the storage is around $2^{d_P/3}$. As an example, if the $P(x)$ polynomial has degree $d_P = 90$, the computational complexity would be around 2^{60}. For algorithms doing simple operations there is no better approach known. There is however another approach using a special kind of discrete log problem [DLC07] that can reduce the computational complexity for solving the W4PM problem to solving $2^{d_P/3}$ discrete log instances. Still, each discrete log instance requires quite a lot of computations to be solved.

We now present a new algorithm for solving the defined problem with computational complexity of only around $2^{d_P/3}$ and similar storage. The algo-

Algorithm 11 (Weight-4-multiple)

Input: Polynomial $P(x) \in \mathbb{F}_2[x]$ of degree d_P and algorithm parameter $\alpha \in \mathbb{N}$.

Output: A polynomial multiple $K(x) \in \mathbb{F}_2[x]$ of weight 4.

1 **repeat**
2 Select a random subset $\mathcal{Q} \subset \{1, 2, \ldots, d_P\}$ such that $|\mathcal{Q}| = d_P/3$;
3 From $P(x)$, create all residues $x^{i_1} \bmod P(x)$, for $0 \leqslant i_1 < 2^{d_P/3+\alpha}$ and put $(x^{i_1} \bmod P(x), i_1)$ in a list $\mathcal{L}_1$;
4 Create all residues $x^{i_1} + x^{i_2} \bmod P(x)$ such that $\phi_{\mathcal{Q}}(x^{i_1} + x^{i_2} \bmod P(x)) = 0$, for $0 \leqslant i_1 < i_2 < 2^{d_P/3+\alpha}$ and put in a list $\mathcal{L}_2$. This is done by merging the sorted list $\mathcal{L}_1$ by itself and keeping only residues $\phi_{\mathcal{Q}}(x^{i_1} + x^{i_2} \bmod P(x)) = 0$. The list $\mathcal{L}_2$ is sorted according to the residue value;
5 In the final step we merge the sorted list $\mathcal{L}_2$ with itself to create a list $\mathcal{L}$, keeping only residues $x^{i_1} + x^{i_2} + x^{i_3} + x^{i_4} = 0 \bmod P(x)$;
6 **until** $|\mathcal{L}| > 0$
7 **return** $\mathcal{L}$

rithm uses the idea of duplicating the desired multiple to many instances and then finding one of them with very large probability.

6.3.1 COMPLEXITY ANALYSIS

Let us analyze the algorithm using the theory we described in Section 1.7. Assume that $K(x)$ is the weight 4 multiple of lowest degree and assume that its degree is around $d_P/3$ as expected. The algorithm considers and creates all weight 4 multiples up to degree $2^{d_P/3+\alpha}$, but will only find those that include two monomials x^{i_1} and x^{i_2} such that $\phi_{\mathcal{Q}}(x^{i_1} + x^{i_2} \bmod P(x)) = 0$.

As $K(x)$ is of weight 4, any polynomial $x^{i_1}K(x)$ is also of weight 4 and since we consider all weight-4 multiples up to degree $2^{d_P/3+\alpha}$ we will consider $2^{d_P/3+\alpha} - 2^{d_P/3}$ such weight 4 polynomials, i.e. about $2^{d_P/3}(2^{\alpha} - 1)$ duplicates of $K(x)$. As the probability for a single weight 4 polynomial to have the condition $\phi_{\mathcal{Q}}(x^{i_1} + x^{i_2} \bmod P(x)) = 0$ can be approximated to be around $2^{-d_P/3}$, there will be a large probability that at least one duplicate $x^{i_1}K(x)$ will survive in Step 2 in the above algorithm and will be included in the output.

Regarding complexity, we note that the tables are all of size $2^{d_P/3}$. Creation of $\mathcal{L}_1$ costs roughly $2^{d_P/3}$ and creation of $\mathcal{L}_2$ costs about the same as we are only accessing entries in $\mathcal{L}_1$ with the same $d_P/3$ bits given by $\mathcal{Q}$. For a randomly chosen $\mathcal{Q}$ and for a sufficiently large α, one iteration of Algorithm 11

(inner loop) succeds with high probability. Of course, the average number of iterations and the work performed inside the loop constitutes a trade-off, determined by the parameter α.

6.3.2 SIMULATION RESULTS

We have simulated Algorithm 11 over random polynomials of a fixed degree. First, we used a brute-force algorithm to determine the minimum-degree polynomial multiple of weight 4 after which we experimented with different values on parameter α to see how it affects the probability of finding the minimum-degree multiple of weight 4.

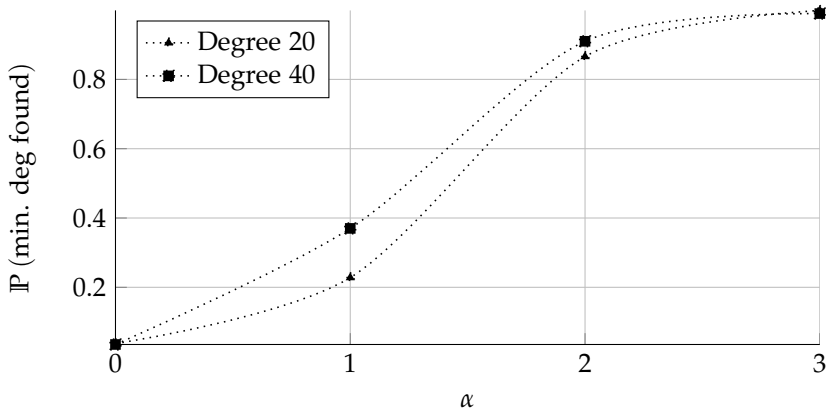


Figure 6.4: The probability of finding the minimum-degree polynomial multiple as a function of algorithm parameter α.

Figure 6.4 shows the simulated probability of finding the minimum-degree polynomial multiple taken over the ensemble of all degree-20 polynomials (line marked with triangles) and all degree-40 polynomials (line marked with squares) respectively.

Example 6.2 *For $d_P = 90$ the complexity of the classical approach is 2^{60}. Running Algorithm 11 with parameter $\alpha = 3$ yields a complexity of only 2^{33} with very low probability of not finding the lowest degree polynomial multiple.*

6.4 ATTACKS ON QC-MDPC MCELIECE IN RELATION TO LWPM

In this section, we provide a proof sketch of that the key-recovery problem we faced in Chapter 5 is no harder than the LWPM problem, up to polynomial factors.

Proposition 6.7 QC-MDPC polynomial-time reduces to LWPM.

Proof. Recall from Definition 1.10 that QC-MDPC is reducible to LWPM if a sub-routine or an oracle for LWPM also can be used to solve QC-MDPC efficiently, making at most polynomially many queries.

Suppose that we are given an instance of QC-MDPC, i.e., we are given the polynomial $P(x)$ that is defined by

$$P(x) = h_1^{-1}(x)h_0(x) \in \mathbb{F}_2[x]/(x^r + 1). \tag{6.18}$$

The task is to find the desired low-weight polynomial $h_1(x)$, for which it holds

$$\begin{cases} w_{\mathrm{H}}\left(h_1(x) \bmod (x^r+1)\right) = w, \\ w_{\mathrm{H}}\left(h_1(x)P(x) \bmod (x^r+1)\right) = w. \end{cases} \tag{6.19}$$

We have split the proof into three parts. They are as follows.

1. First, we claim that, without loss of generality, it suffices to show the case $n_0 = 2$. We provide a supporting argument for this claim later on. We write (6.18) as

$$h_1(x)P(x) = h_0(x) + s(x)(x^r + 1), \tag{6.20}$$

with $\deg s(x) < r$. Now, we define a polynomial

$$\begin{aligned} Q(x) \stackrel{\text{def}}{=} & P(x) + x^r P(x) + \cdots + x^{r(r+1)}P(x) \\ & + x^{r(r+2)} + x^{r(r+3)} + \cdots + x^{2r(r+3)}, \end{aligned} \tag{6.21}$$

and then show that

$$\begin{aligned} h_1(x)Q(x) = & h_1(x)P(x) + x^r h_1(x)P(x) + \cdots + x^{r(r+1)}h_1(x)P(x) \\ & + x^{r(r+2)}h_1(x) + x^{r(r+3)}h_1(x) + \cdots + x^{2r(r+3)}h_1(x) \end{aligned} \tag{6.22}$$

is a minimal solution. By (6.20), we can write (6.22) as

$$\begin{aligned} & h_0(x) + s(x)(x^r+1) + \cdots + x^{r(r+1)}\left[h_0(x) + s(x)(x^r+1)\right] \\ & + x^{r(r+2)}h_1(x) + \cdots + x^{2r(r+3)}h_1(x). \end{aligned} \tag{6.23}$$

The sum involving $s(x)$ is a telescopic sum, so (6.22) simplifies to

$$\begin{aligned} & s(x) + h_0(x) + \overbrace{x^r h_0(x) \cdots + x^{r(r+1)}h_0(x)}^{r+1 \text{ terms}} \\ & + x^{r(r+2)}s(x) + x^{r(r+2)}h_1(x) + \underbrace{x^{r(r+3)}h_1(x) \cdots + x^{2r(r+3)}h_1(x)}_{r+1 \text{ terms}}. \end{aligned} \tag{6.24}$$

Here, $w_{\mathrm{H}}(s(x) + h_0(x)) \leqslant r$ and $w_{\mathrm{H}}(s(x) + h_1(x)) \leqslant r$. Therefore

$$w_{\mathrm{H}}\left(h_1(x)Q(x)\right) \leqslant 2r + 2(r+1)w \tag{6.25}$$

We claim that this solution is minimal. To prove it, we make an argument by contradiction.

2. Assume that another polynomial $t(x)$ yields a minimal-weight solution, i.e., $t(x)Q(x)$ is minimal. We write $t(x)P(x) = t'(x) + s'(x)(x^r + 1)$. Then, we have

$$\begin{aligned} &s'(x) + t'(x) + \overbrace{x^r t'(x) \cdots + x^{r(r+1)} t'(x)}^{r+1 \text{ terms}} \\ &+ x^{r(r+2)} s'(x) + x^{r(r+2)} t(x) + \underbrace{x^{r(r+3)} t(x) \cdots + x^{2r(r+3)} t(x)}_{r+1 \text{ terms}}. \end{aligned} \tag{6.26}$$

By definition $w_{\mathrm{H}}(t(x)) \geqslant w + 1$ and $w_{\mathrm{H}}(t'(x)) \geqslant w + 1$ and, thus,

$$w_{\mathrm{H}}\left(t(x)Q(x)\right) \geqslant 2(r+1) \cdot (w+1) = 2(r+1) + 2(r+1)w. \tag{6.27}$$

The assumption of minimality implies that the weight of $w_{\mathrm{H}}(t(x)Q(x))$ should be less than (6.25), but that claim does not hold. So we have a contradiction and may therefore conclude that $Q(x)$ provides the minimal solution.

We now (informally) sketch the proof for that the argument holds for any $n_0 > 2$. We have that

$$P_i(x) = h_{n_0-1}^{-1}(x) h_i(x) \in \mathbb{F}_2[x]/(x^r + 1). \tag{6.28}$$

for $0 \leqslant i < n_0 - 1$. Construct the polynomial

$$\begin{aligned} Q'(x) \stackrel{\text{def}}{=} & P_0(x) + x^r P_0(x) + \cdots + x^{r(r+1)} P_0(x) \\ & + x^{r(r+2)} P_1(x) + x^{r(r+3)} P_1(x) + \cdots + x^{2r(r+3)} P_1(x) \\ & \qquad \vdots \\ & + x^{(n_0-1)r(r+n_0)} + x^{(n_0-1)r(r+n_0)+r} + \cdots + x^{n_0 r(r+n_0+1)}. \end{aligned} \tag{6.29}$$

As for the case $n_0 = 2$, we can write

$$h_{n_0-1}(x) P_i(x) = h_i(x) + s_i(x)(x^r + 1), \tag{6.30}$$

and for $0 \leqslant i < n_0 - 1$, the sum involving $s_i(x)$ will be a telescopic sum and, thus, collapse. For each term involving $h_i(x)$, the correct solution $h_{n_0-1}(x)$ will yield a minimal weight. Therefore, we may conclude minimal-weight solution for QC-MDPC will be obtained for the LWPM instance $Q'(x)$ for some minimal weight.

3. We construct an instance LWPM with the polynomial $Q(x)$ (or $Q'(x)$), degree $d = r$ and weight $n_0 \cdot r + n_0 \cdot w$. We then ask the oracle for a solution. If the oracle fails to find a solution, we decrease the weight until it succeeds to provide an answer. That answer must be a solution to QC-MDPC.

   Finally, we conclude that it is a polynomial-time reduction and the length of the input size grows by a polynomial factor $\mathcal{O}(n_0 \cdot r)$.

■

We stress that the reduction does not allow us to mount more efficient attacks than the squaring attack we presented in Chapter 5 using the new algorithm for solving LWPM. It is mainly interesting if the LWPM problem is shown to have a sub-exponential algorithm, as that would have negative ramifications for the security assumptions of QC-MDPC McEliece.

6.5 SUMMARY

In this chapter, we have presented two new algorithms for finding low-weight polynomial multiples and shown that they have a better computation complexity than previously known algorithms, breaking at least one instances of TCHo that were proposed for 80-bit security.

Additionally, we have provided a computational hardness relation between LWPM and QC-MDPC.

7

Learning Parity With Noise

If you optimize everything, you will always be unhappy.
– *Donald Ervin Knuth*

In recent years of modern cryptography, much effort has been devoted to finding efficient and secure low-cost cryptographic primitives targeting applications in very constrained hardware environments (such as RFID tags and low-power devices). Many proposals rely on the hardness assumption of *Learning Parity with Noise* (LPN), a fundamental problem in machine learning theory, which recently also has gained a lot of attention within the cryptographic society. The LPN problem is well-studied and it is intimately related to the problem of decoding random linear codes, which is one of the most important problems in coding theory. Being a supposedly hard problem, LPN is a good candidate for post-quantum cryptography, where other hard number-theoretic problems such as factoring and the discrete log problem fall short. The inherent properties of LPN also makes it ideal for light-weight cryptography.

The first cryptographic construction to employ the LPN problem was the Hopper-Blum (HB) identification protocol [HB01]. HB is a minimalistic protocol that is secure in a *passive* attack model. Aiming to secure the HB scheme also in an *active* attack model, Juels and Weis [JW05], and Katz and Shin [KS06] proposed a modified scheme. The modified scheme, which was given the name HB^+, extends HB with one extra round. It was later shown by Gilbert *et al.* [GRS05] that the HB^+ protocol is vulnerable to active attacks, i.e., man-in-the-middle attacks, where the adversary is allowed to intercept and attack an ongoing authentication session to learn the secret. Gilbert *et al.* [GRS08a] subsequently proposed a variant of the Hopper-Blum protocol called $HB^\#$.

Apart from repairing the protocol, the constructors of HB# introduced a more efficient key representation using a variant of LPN called Toeplitz-LPN.

In [GRS08b], Gilbert *et al.* proposed a way to use LPN in encryption of messages, which resulted in the cryptosystem LPN-C. Kiltz *et al.* [KPC+11] and Dodis *et al.* [DKPW12] showed how to construct message authentication codes (MACs) using LPN. The most recent contribution to LPN-based constructions is a two-round identification protocol called Lapin, proposed by Heyse *et al.* [HKL+12], and an LPN-based encryption scheme called Helen, proposed by Duc and Vaudenay [DV13]. The Lapin protocol is based on an LPN variant called Ring-LPN, where the samples are elements of a polynomial ring.

Two major threats against LPN-based cryptographic constructions are generic information-set decoding algorithms that decode random linear codes and variants of the BKW algorithm, originally proposed by Blum, Kalai and Wasserman [BKW03]. Being the asymptotically most efficient[1] approach, the BKW algorithm employs an iterated collision procedure on the queries. In each iteration, colliding entries sum together to produce a new entry with smaller dependency on the information bits but with an increased noise level. Once the dependency from sufficiently many information bits are removed, the remaining are exhausted to find the secret. Although the collision procedure is the main reason for the efficiency of the BKW algorithm, it leads to a requirement of an immense amount of queries compared to ISD. Notably, for some cases, e.g., when the noise level is very low, ISD algorithms yield the most efficient attack.

Levieil and Fouque [LF06] proposed to use fast Walsh-Hadamard transform in the BKW algorithm when searching for the secret. In an unpublished paper, Kirchner [Kir11] suggested to transform the problem into systematic form, where each information (key) bit then appears as an observed symbol, perturbed by noise. This requires the adversary only to exhaust the biased noise variables rather than the key bits. When the error rate is low, the noise variable search space is very small and this technique decreases the attack complexity. Building on the work by Kirchner [Kir11], Bernstein and Lange [BL13] showed that the ring structure of Ring-LPN can be exploited in matrix inversion, further reducing the complexity of attacks on for example Lapin. None of the known algorithms manage to break the 80-bit security of Lapin. Nor do they break the parameters proposed in [LF06], which were suggested as design parameters of LPN-C [GRS08b] for 80-bit security.

In this chapter we present a new algorithm for solving the LPN problem, as proposed in [GJL14]. We employ a new technique that we call *subspace distinguishing*, which exploits coding theory (or rate-distortion theory) to decrease

[1] For a fixed error rate.

the dimension of the secret. The trade-off is a small increase in the sample noise. The new algorithm performs favorably in comparison to »state-of-the-art« algorithms and we manage to break previously unbroken parameters of HB variants and LPN-C, and provide a new upper bound on the security of Lapin (irreducible case). As an example, we attack the common $\mathsf{LPN}(512, \frac{1}{8})$-instance and break its 80-bit security barrier. A comparison of complexity of different algorithms[2] is shown in Table 7.1.

Algorithm	Complexity ($\log_2$)		
	Queries	Time	Memory
Levieil-Fouque [LF06]	75.7	87.5	84.8
Bernstein-Lange [BL13]	68.6	85.7	77.6
New algorithm (LF1)	65.0	80.7	74.0
New algorithm (LF2)	63.6	79.7	72.6

Table 7.1: Comparison of different algorithms for solving LPN with parameters $(512, \frac{1}{8})$.

7.1 THE LPN PROBLEM

We will now give a more thorough description of the LPN problem. Let k be a security parameter and let $\mathbf{x}$ be a binary vector of length k.

Definition 7.1 (LPN oracle) An LPN oracle $\Pi^{\eta}_{\mathsf{LPN}}$ for an unknown vector $\mathbf{x} \in \{0,1\}^k$ with $\eta \in (0, \frac{1}{2})$ returns pairs of the form

$$\left(\mathbf{g} \xleftarrow{\$} \{0,1\}^k, \langle \mathbf{x}, \mathbf{g} \rangle + e\right), \tag{7.1}$$

where $e \leftarrow \mathsf{Ber}_{\eta}$.

We receive a number n of noisy versions of scalar products of $\mathbf{x}$ from the oracle $\Pi^{\eta}_{\mathsf{LPN}}$, and our task is to recover $\mathbf{x}$.

[2]The Bernstein-Lange algorithm is originally proposed for Ring-LPN, and by a slight modification [BL13], one can apply it to the LPN instances as well. It shares the beginning steps (i.e., the steps of Gaussian elimination and the collision procedure) with the new algorithm, so for a fair comparison, we use the same implementation of these steps when computing their complexity.

Problem 7.1 (Learning Parity with Noise) Given an LPN oracle $\Pi^{\eta}_{\mathsf{LPN}}$, the $\mathsf{LPN}(k, \eta)$ (search) problem consists of finding the vector $\mathbf{x}$. An algorithm $\mathcal{A}_{\mathsf{LPN}}$ is said to (t, n, θ)-solve an instance (k, η) if it runs in time at most t, makes at most n oracle queries and solves (k, η) with probability

$$\mathbb{P}\left(\mathcal{A}_{\mathsf{LPN}}(\Pi^{\eta}_{\mathsf{LPN}}) = \mathbf{x} \mid \mathbf{x} \xleftarrow{\$} \{0,1\}^k\right) \geqslant \theta. \tag{7.2}$$

Let $\mathbf{y}$ be a vector of length n and let $y_i = \langle \mathbf{x}, \mathbf{g}_i \rangle$. For known random vectors $\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_n$, we can easily reconstruct an unknown $\mathbf{x}$ from $\mathbf{y}$ using linear algebra. In LPN, however, we receive instead noisy versions of $y_i, i = 1, 2, \ldots, n$. Writing the noise in position i as $e_i, i = 1, 2, \ldots, n$ we obtain

$$z_i = y_i + e_i = \langle \mathbf{x}, \mathbf{g}_i \rangle + e_i. \tag{7.3}$$

In matrix form, the same is written as $\mathbf{z} = \mathbf{x}\mathbf{G} + \mathbf{e}$, where

$$\mathbf{z} = \begin{pmatrix} z_1 & z_2 & \cdots & z_n \end{pmatrix}, \tag{7.4}$$

and the matrix $\mathbf{G}$ is formed as

$$\mathbf{G} = \begin{pmatrix} \mathbf{g}_1^{\mathrm{T}} & \mathbf{g}_2^{\mathrm{T}} & \cdots & \mathbf{g}_n^{\mathrm{T}} \end{pmatrix}. \tag{7.5}$$

This shows that the LPN problem is simply a decoding problem, where $\mathbf{G}$ is a random $k \times n$ generator matrix, $\mathbf{x}$ is the information vector and $\mathbf{z}$ is the received vector after transmission of a codeword on the binary symmetric channel with error probability η.

7.2 THE BKW ALGORITHM

The BKW algorithm, as proposed by Blum, Kalai and Wasserman in [BKW03], is an algorithm that solves the LPN problem in sub-exponential time, requiring $2^{\mathcal{O}(k/\log k)}$ queries and time. To achieve this, the algorithm uses an iterative sort-and-match procedure (see Section 1.7 for more details) on the columns of the query matrix $\mathbf{G}$, which iteratively reduces the dimension of $\mathbf{G}$.

1. **Reduction phase**: Recall the k-COL problem. If the queries, represented by the columns of $\mathbf{G}$, constitute the elements of the set $\mathcal{S}$, then the BKW algorithm takes a parameter t and as a first step solves the 2^t-COL problem (but leaves a residue of weight 1).

   The operation $\diamond$ is realized in the following way. Initially, one searches for all combinations of two columns in $\mathbf{G}$ that add to zero in the last b entries. Let

$$\mathcal{M} \stackrel{\text{def}}{=} \{k - b + 1, k - b + 2, \ldots, k\} \tag{7.6}$$

and define a filtering function $\phi_{\mathcal{M}} : \mathbb{F}_2^k \to \mathbb{F}_2^b$. Assume that one finds two columns $\mathbf{g}_{i_1}^{\mathrm{T}}, \mathbf{g}_{i_2}^{\mathrm{T}}$ such that

$$\mathbf{g}_{i_1} + \mathbf{g}_{i_2} = (* \quad * \quad \cdots \quad * \quad \underbrace{0 \quad 0 \quad \cdots \quad 0}_{b \text{ symbols}}), \tag{7.7}$$

where $*$ means any value, i.e., they belong to the same *partition* (or equivalence class) and fulfill $\phi_{\mathcal{M}}(\mathbf{g}_{i_1}) = \phi_{\mathcal{M}}(\mathbf{g}_{i_2})$. Then, a new vector

$$\mathbf{g}_1^{(2)} = \mathbf{g}_{i_1} + \mathbf{g}_{i_2} \tag{7.8}$$

is computed. An *observed symbol* is also formed, corresponding to this new column by forming

$$z_1^{(2)} = z_{i_1} + z_{i_2} = y_1^{(2)} + e_1^{(2)} = \langle \mathbf{x}, \mathbf{g}_1^{(2)} \rangle + e_1^{(2)}, \tag{7.9}$$

where now $e_1^{(2)} = e_{i_1} + e_{i_2}$. It can be verified that $\mathbb{P}(e_1^{(2)} = 0) = \frac{1}{2} \cdot (1 + \epsilon^2)$. The algorithm proceeds by adding the same element, say $\mathbf{g}_{i_1}$, to the other elements in the partition forming

$$z_2^{(2)} = z_{i_1} + z_{i_3} = y_2^{(2)} + e_2^{(2)} = \langle \mathbf{x}, \mathbf{g}_2^{(2)} \rangle + e_2^{(2)}, \tag{7.10}$$

and so forth. The resulting columns are stored in a matrix $\mathbf{G}_2$,

$$\mathbf{G}_2 = \left((\mathbf{g}_1^{(2)})^{\mathrm{T}} \quad (\mathbf{g}_2^{(2)})^{\mathrm{T}} \quad \ldots \quad (\mathbf{g}_{n-2^b}^{(2)})^{\mathrm{T}} \right). \tag{7.11}$$

If n is the number of columns in $\mathbf{G}$, then the number of columns in $\mathbf{G}_2$ will be $n - 2^b$. Note that the last b entries of every column in $\mathbf{G}_2$ are all zero. In connection to this matrix, the vector of observed symbols is

$$\mathbf{z}_2 = \left(z_1^{(2)} \quad z_2^{(2)} \quad \cdots \quad z_{n-2^b}^{(2)} \right), \tag{7.12}$$

where $\mathbb{P}\left(z_i^{(2)} = y_i^{(2)}\right) = \frac{1}{2} \cdot (1 + \epsilon^2)$, for $1 \leqslant i \leqslant n - 2^b$.

We now iterate the same (with a new ϕ function), picking one column and then adding it to another suitable column in $\mathbf{G}_i$ giving a sum with an additional b entries being zero, forming the columns of $\mathbf{G}_{i+1}$. Repeating the same procedure an additional $t - 2$ times will reduce the number of unknown variables to $k - b \cdot t$ in the remaining problem.

For each iteration the noise level is squared. By the piling-up lemma (Lemma 1.3) we have that

$$\mathbb{P}\left(\sum_{j=1}^{2^t} e_i = 0 \right) = \frac{1}{2} \cdot \left(1 + \epsilon^{2^t}\right). \tag{7.13}$$

Algorithm 12 (BKW)

Input: Algorithm parameters $b, t, n \in \mathbb{N}$.

Output: First bit $x_1 \in \mathbb{F}_2$ of secret vector $\mathbf{x} \in \mathbb{F}_2^k$.

```
repeat
    (Reduction phase) Query the oracle for n queries of the form (g, z)
    Create a query matrix G = (g_1^T  g_2^T  ··· g_n^T) and observed vector
    z = (z_1  z_2  ···  z_n)
    for i ∈ {1, 2, ..., n} do
        S ← S ∪ (g_i, z_i)
    for i ∈ {1, 2, ..., t} do
        Partition S according to b · i last bits;
        for each partition P ∈ S do
            Pick a random (g', z') ∈ P and remove it from P;
            Replace all remaining elements (g, z) ∈ P with (g + g', z + z');
    (Solving phase) Find a column vector g'_i in G' such that only the first
    bit g_1 is non-zero and the remaining positions are all-zero. Then, the
    observed value z_i is also an observation of x_1;
until sufficiently many observations have been obtained
Determine the secret bit x_1 by majority decision;
return x_1
```

Hence, the bias decreases quickly to low levels as t increases. Therefore, we want to keep t as low as possible.

2. **Solving phase**: In the final step, the BKW algorithm looks for a column vector in $\mathbf{G}_{t-1}$ such that only the first bit of the vector is non-zero. If the algorithm finds such a vector, then that sample constitutes a very noisy observation the first bit x_1 of $\mathbf{x}$. The algorithm stores the observation and repeats the reduction-phase procedure with new samples from the oracle, until sufficiently many observations of the secret bit x_1 have been obtained. Then, it uses a majority decision to determine x_1. The whole procedure is given in Algorithm 12.

7.2.1 LF1 AND LF2 VARIANTS

The BKW algorithm is a powerful theoretic construction and because the algorithm operates solely on independent samples, it is possible to provide rig-

orous analysis using probabilistic arguments without heuristic assumptions. However, the provability comes at a quite high expense – the algorithm discards a lot of samples that could be used in solving the problem. This was first pointed out by Levieil and Fouque in [LF06]. The authors of [LF06] suggested that all samples should be kept after the reduction and not only the ones having weight 1. Instead of determining the secret bit by bit using majority decision, the whole $k - t \cdot b$ bit secret may be determined using Walsh transformation. The authors suggested two methods: LF1 and LF2 – the methods are essentially the same, but differ in how the columns to be merged are chosen.

- LF1 picks a column in each partition, and then adds it to the remaining samples in the same partition (entries having the same last b entries). This is identical to how BKW operates in its merging steps.

  The number of samples is reduced by 2^b after each merge operation. Hence, after a series of t merges, the number of samples is about

  $$r(t) = n - t \cdot 2^b. \tag{7.14}$$

  The algorithm uses fast Walsh-Hadamard transform to determine the remaining secret of dimension $k - t \cdot b$. Thus, no samples are discarded and the algorithm does, in contrast BKW, not query the oracle a multiple number of times. Therefore, a factor 2^b is lost in terms of query complexity.

  The LF1 method was subsequently adopted by Bernstein and Lange in [BL13].

- The other method, LF2, computes all pairs within the same partition. It produces more samples at the cost of increased dependency, thereby gaining more efficiency in practice.

  Given that there are on average $\frac{n}{2^b}$ samples in one partition, we expect around

  $$2^b \binom{n/2^b}{2} \tag{7.15}$$

  samples at the end of one merge step in LF2, or more generally

  $$r'(t) = 2^b \cdot \binom{r'(t-1)/2^b}{2}, \tag{7.16}$$

  after t merging steps.

  Like LF1, a fast Walsh-Hadamard transform is used to determine the secret. Combined with a more conservative use of samples, LF2 is expected to be at least as efficient as LF1 in practice. In particular, LF2 has great advantage when the attacker has restricted access to the oracle.

We have illustrated the different methods in Figure 7.1.

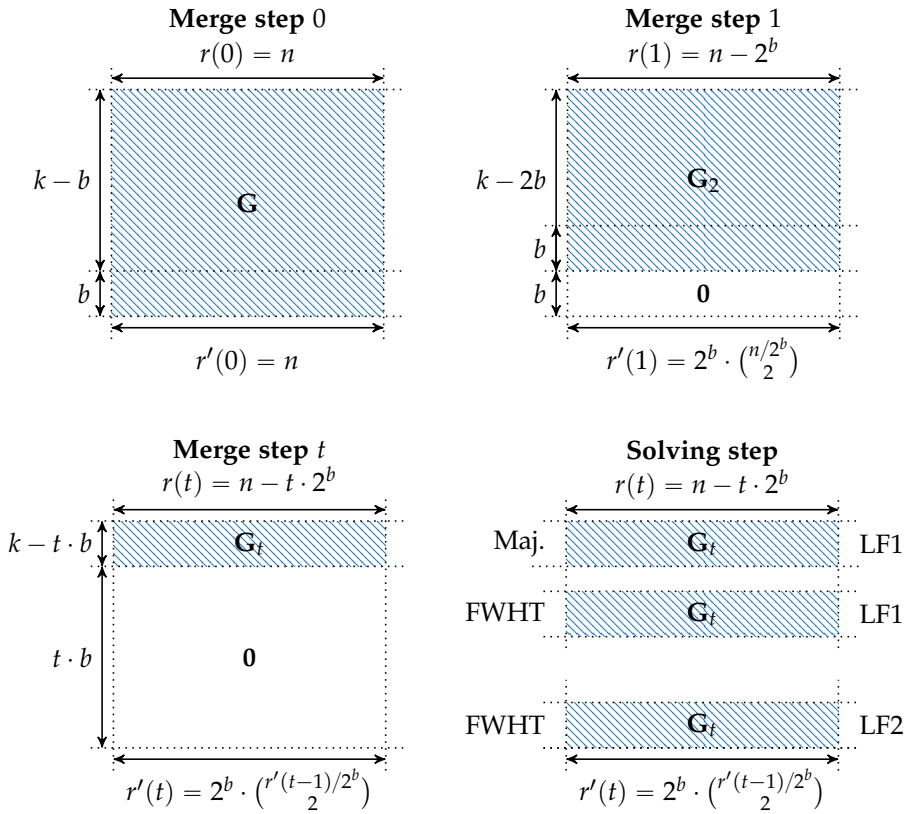


Figure 7.1: In the above, we illustrate t merging steps and sample count at each t with respect to BKW/LF1, $r(t)$ and LF2, $r'(t)$.

7.3 A NEW ALGORITHM

In this section we aim to give a very basic description of the idea used to give a new and more efficient algorithm for solving the LPN problem. A more detailed analysis will be provided in later sections.

Assume that we have an initial LPN problem described by

$$\mathbf{G} = \begin{pmatrix} \mathbf{g}_1^{\mathrm{T}} & \mathbf{g}_2^{\mathrm{T}} & \cdots & \mathbf{g}_n^{\mathrm{T}} \end{pmatrix} \tag{7.17}$$

and $\mathbf{z} = \mathbf{x}\mathbf{G} + \mathbf{e}$, where $\mathbf{z} = \begin{pmatrix} z_1 & z_2 & \cdots & z_n \end{pmatrix}$ and

$$z_i = y_i + e_i = \langle \mathbf{x}, \mathbf{g}_i \rangle + e_i. \tag{7.18}$$

As previously shown in [Kir11] and [BL13], we may through Gaussian elimination transform $\mathbf{G}$ into systematic form. Assume that the first k columns are linearly independent and forms the matrix $\mathbf{D}$. With a change of variables $\hat{\mathbf{x}} = \mathbf{x}\mathbf{D}^{-1}$ we get an equivalent problem description with

$$\mathbf{G}_{\text{sys}} = \begin{pmatrix} \mathbf{I} & \hat{\mathbf{g}}_{k+1}^{\mathrm{T}} & \hat{\mathbf{g}}_{k+2}^{\mathrm{T}} & \cdots & \hat{\mathbf{g}}_{n}^{\mathrm{T}} \end{pmatrix}. \tag{7.19}$$

We compute

$$\hat{\mathbf{z}} = \mathbf{z} + \begin{pmatrix} z_1 & z_2 & \ldots & z_k \end{pmatrix} \mathbf{G}_{\text{sys}} = \begin{pmatrix} \mathbf{0} & \hat{z}_{k+1} & \hat{z}_{k+2} & \ldots & \hat{z}_n \end{pmatrix}. \tag{7.20}$$

In this situation, one may start performing a number of merging steps on columns $k+1$ to n, reducing the dimension k of the problem to something smaller. This will result in a new problem instance where noise in each position is larger, except for the first systematic positions. We may write the problem after performing t merging steps in the form

$$\mathbf{G}' = \begin{pmatrix} \mathbf{I} & \mathbf{g}_1'^{\,\mathrm{T}} & \mathbf{g}_2'^{\,\mathrm{T}} & \cdots & \mathbf{g}_m'^{\,\mathrm{T}} \end{pmatrix} \tag{7.21}$$

and

$$\mathbf{z}' = \begin{pmatrix} \mathbf{0} & z_1' & z_2' & \ldots & z_m' \end{pmatrix}, \tag{7.22}$$

where now $\mathbf{G}'$ has dimension $k' \times m$ with $k' = k - t \cdot b$ and m is the number of columns remaining after the merge step. These steps are illustrated in Figure 7.2. We have $\mathbf{z}' = \mathbf{x}'\mathbf{G}' + \tilde{\mathbf{e}}$,

$$\mathbb{P}\left(x_i' = 0\right) = \frac{1}{2} \cdot (1 + \epsilon) \tag{7.23}$$

and

$$\mathbb{P}\left(\langle \mathbf{x}', \mathbf{g}_i' \rangle = z_i\right) = \frac{1}{2} \cdot (1 + \epsilon^{2^t}). \tag{7.24}$$

Now, we will explain the basics of the new idea we propose. In a problem instance as above, we may look at the random variables $y_i' = \langle \mathbf{x}', \mathbf{g}_i' \rangle$. The bits in $\mathbf{x}'$ are mostly zero but a few are set to one (there are around $k' \cdot \eta$ non-zero elements).

The first step in the solving phase is to introduce a covering code with covering radius d_C. Vectors $\mathbf{g}_i'$ are of length k', so we initially consider a code of length k' and some dimension l, having generator matrix $\mathbf{F}$. For each vector $\mathbf{g}_i'$, we now find the codeword in the code spanned by $\mathbf{F}$ that is closest in terms

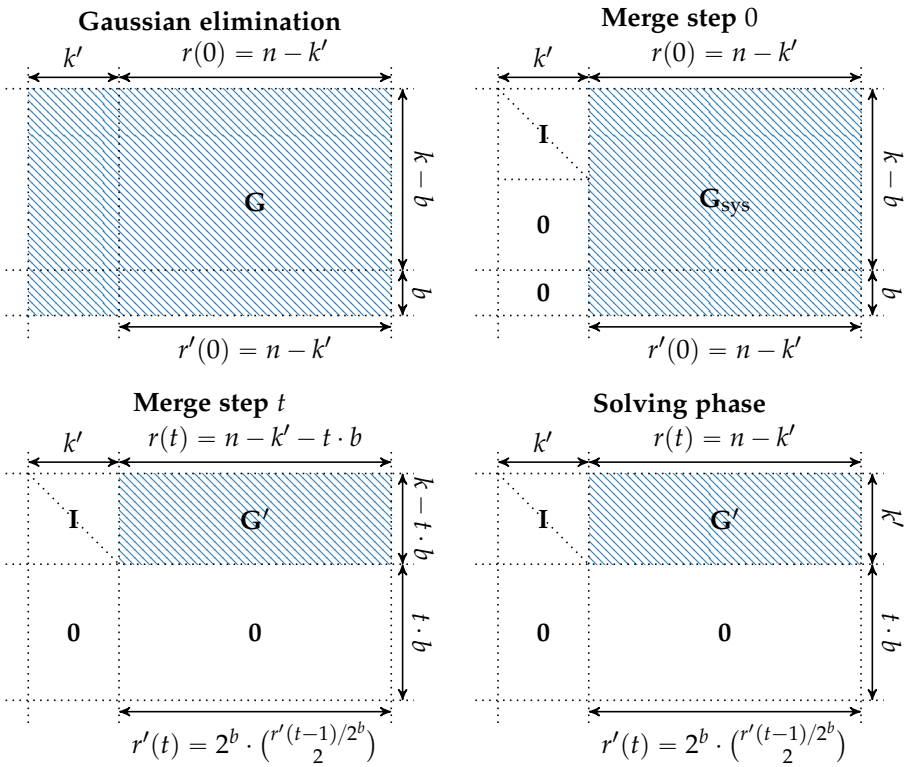


Figure 7.2: Here, we illustrate the different steps of the new algorithm, using the LF1 and the LF2 merging approaches. In the figure, we only show the upper systematic part used in hypothesis testing.

of Hamming distance to $\mathbf{g}'_i$. Assume that this codeword is denoted $\mathbf{c}_i$. Then, we can write

$$\mathbf{g}'_i = \mathbf{c}_i + \mathbf{e}'_i, \tag{7.25}$$

where $\mathbf{e}'_i$ is a biased vector, having weight at most d_C. It remains to examine exactly how biased the bits in $\mathbf{e}'_i$ will be. Going back to our previous expressions we can write

$$y'_i = \langle \mathbf{x}', \mathbf{g}'_i \rangle = \langle \mathbf{x}', \mathbf{c}_i + \mathbf{e}'_i \rangle = \langle \mathbf{x}', \mathbf{c}_i \rangle + \langle \mathbf{x}', \mathbf{e}'_i \rangle \tag{7.26}$$

and since $\mathbf{c}_i = \mathbf{u}_i \mathbf{F}$ for some $\mathbf{u}_i$, we can write (this is shown in the proof of Lemma 7.3)

$$y'_i = \langle \mathbf{x}' \mathbf{F}^{\mathrm{T}}, \mathbf{u}_i \rangle + \langle \mathbf{x}', \mathbf{e}'_i \rangle. \tag{7.27}$$

We may introduce $\mathbf{x}'' \stackrel{\text{def}}{=} \mathbf{x}'\mathbf{F}^{\mathrm{T}}$ as a length l vector of unknown bits (linear combinations of bits from $\mathbf{x}'$) and write

$$y'_i = \langle \mathbf{x}'', \mathbf{u}_i \rangle + \langle \mathbf{x}', \mathbf{e}'_i \rangle. \tag{7.28}$$

As we established before, the term $\langle \mathbf{x}', \mathbf{e}'_i \rangle$ is clearly biased, but we did not give any explicit formulas. Having introduced the necessary details, we now formulate the following proposition.

Proposition 7.2 (Bias from covering code [BV15]) If the covering code $\mathbf{F}$ has optimal radius covering, then the probability $\mathbb{P}\left(\langle \mathbf{x}', \mathbf{e}'_i \rangle = 1 \mid w_{\mathrm{H}}\left(\mathbf{x}'\right) = c\right)$ is given by

$$|\mathcal{B}_2(k'', d_C)|^{-1} \cdot \sum_{i \text{ odd}}^{\min(c, d_C)} \binom{c}{i} \cdot |\mathcal{B}_2(k'' - c, d_C - i)| \stackrel{\text{def}}{=} \varphi(c), \tag{7.29}$$

where k'' is the dimension of $\mathbf{x}'$ and d_C is the covering distance.

Proof. Let the c non-zero positions of $\mathbf{x}'$ represent a set of bins and the $k'' - c$ zero positions another set of bins.

$$\underbrace{\sqcup\ \sqcup\ \cdots\ \sqcup}_{c} \;\Big|\; \underbrace{\sqcup\ \sqcup\ \sqcup\ \sqcup\ \cdots\ \sqcup}_{k''-c}$$

If there is an odd number of balls in the c bins, then $\langle \mathbf{x}', \mathbf{e}'_i \rangle = 1$. Suppose that there are i balls. Then, there are $\binom{c}{i}$ ways to arrange the balls within those bins. In total, we may place up to $j \stackrel{\text{def}}{=} \min(c, d_C)$ balls, so there remains up to $j - i$ balls to be placed in the other set of $k'' - c$ bins, which counts to $|\mathcal{B}_2(k'' - c, d_C - i)|$ possibilities. The summation includes all odd i. ■

We now put the analysis together, taking both the error introduced by the LPN oracle and the merging steps, and the error introduced by the covering code into consideration. Since we have

$$\mathbb{P}\left(y'_i = z'_i\right) = \frac{1}{2} \cdot (1 + \epsilon^{2^t}), \tag{7.30}$$

we get

$$\mathbb{P}\left(\langle \mathbf{x}'', \mathbf{u}_i \rangle = z'_i\right) = \frac{1}{2} \cdot (1 + \epsilon^{2^t} \cdot \epsilon'), \tag{7.31}$$

where ϵ' is the bias determined by the distance between $\mathbf{g}'_i$ and the closest codeword in the code we are using, From (7.29), we can determine bias ϵ' from the covering code as $\epsilon' = 1 - 2\varphi(c)$, where c is the number of positions in $\mathbf{x}'$ set to one.

The last step in the new algorithm now selects, in accordance with (1.40), about

$$m = 2\ln 2 \cdot \frac{-l \cdot \log_2 \theta}{\left(\epsilon^{2^t} \cdot \epsilon'\right)^2} \tag{7.32}$$

samples $z'_1, z'_2, \ldots, z'_m$ and for each guess of the 2^l possible values of $\mathbf{x}''$, we compute how many times $\langle \mathbf{x}'', \mathbf{u}_i \rangle = z'_i$ when $i = 1, 2, \ldots, m$. As this step is similar to a correlation attack scenario, we know that it can be efficiently computed using fast Walsh-Hadamard transform. After recovering $\mathbf{x}''$, it is an easy task to recover remaining unknown bits of $\mathbf{x}'$. The whole procedure is summarized in Algorithm 13.

7.3.1 A TOY EXAMPLE

In order to illustrate the ideas and convince the reader that the proposed algorithm can be more efficient than previously known methods, we consider an example. We assume an LPN instance of dimension $k = 160$, where we allow at most 2^{24} received samples and we allow at most around 2^{24} vectors of length 160 to be stored in memory. Furthermore, the error probability is $\eta = 0.1$.

For this particular case, we propose the following algorithm.

1. The first step is to compute the systematic form,

$$\mathbf{G}_{\text{sys}} = \begin{pmatrix} \mathbf{I} & \hat{\mathbf{g}}_{k+1}^{\mathrm{T}} & \hat{\mathbf{g}}_{k+2}^{\mathrm{T}} & \cdots & \hat{\mathbf{g}}_{n}^{\mathrm{T}} \end{pmatrix}$$

   and

$$\hat{\mathbf{z}} = \mathbf{z} + \begin{pmatrix} z_1 & z_2 & \ldots & z_k \end{pmatrix} \mathbf{G}_{\text{sys}} = \begin{pmatrix} \mathbf{0} & \hat{z}_{k+1} & \hat{z}_{k+2} & \ldots & \hat{z}_n \end{pmatrix}.$$

   Here $\mathbf{G}_{\text{sys}}$ has dimension 160 and $\hat{\mathbf{z}}$ has length at most 2^{24}.

2. In the second step we perform $t = 4$ merging steps (using the BKW/LF1 approach), the first step removing 22 bits and the remaining three each removing 21 bits. This results in $\mathbf{G}' = \begin{pmatrix} \mathbf{I} & \mathbf{g}_1'^{\mathrm{T}} & \mathbf{g}_2'^{\mathrm{T}} & \cdots & \mathbf{g}_m'^{\mathrm{T}} \end{pmatrix}$ and $\mathbf{z}' = \begin{pmatrix} \mathbf{0} & z'_1 & z'_2 & \ldots & z'_m \end{pmatrix}$, where now $\mathbf{G}'$ has dimension $75 \times m$ and m is about $3 \cdot 2^{21}$. We have $\mathbf{z}' = \mathbf{x}'\mathbf{G}'$,

$$\mathbb{P}\left(x'_i = 0\right) = \frac{1}{2} \cdot (1 + \epsilon),$$

   where $\epsilon = 0.8$ and

$$\mathbb{P}\left(\langle \mathbf{x}', \mathbf{g}'_{\mathbf{i}} \rangle = z_i\right) = \frac{1}{2} \cdot (1 + \epsilon^{16}).$$

   Hence, the resulting problem has dimension 75 and the bias is $\epsilon^{2^t} = (0.8)^{16}$.

Algorithm 13 (New algoritm)

Input: Query matrix $\mathbf{G} \in \mathbb{F}_2^{k \times n}$, query vector $\mathbf{z} \in \mathbb{F}_2^n$ and algorithm parameters $a, b, c, t, k'', l, p \in \mathbb{N}$.

Output: Secret vector $\mathbf{x}$.

```
Query the oracle for n queries of the form (g, z) and create a query
  matrix G = (g_1^T  g_2^T  ··· g_n^T) and observed vector z = (z_1  z_2  ···  z_n);
Pick a [k'', l] linear code C with good covering property;
repeat
    (Reduction phase) Pick random column permutation π;
    Perform Gaussian elimination on π(G) resulting in G_0 = (I|L_0);
    for i ∈ {1, 2, ..., n} do
        S ← S ∪ (g_i, z_i)
    for i ∈ {1, 2, ..., t} do
        Partition S according to b · i last bits
        for each partition P ∈ S do
            Pick a random (g', z') ∈ P and remove it from P;
            Replace all remaining elements (g, z) ∈ P with (g + g', z + z');
    (Solving phase) Partition the columns of L_t by the non-all-zero k''
    bits and group them by their nearest codewords;
    k_1 ← k − a · b − k'';
    for x'_2 ∈ B_2(k_1, p) do
        Update the observed samples in accordance with x'_2;
        for y ∈ {0, 1}^l do
            Use fast Walsh-Hadamard transform to compute the numbers
            of 1:s and 0:s observed respectively;
            Perform hypothesis testing whose threshold is defined as a
            function of c;
until acceptable hypothesis is found
```

3. In the third step we then select a suitable code of length 75. In this example we choose a block code which is a direct sum of 25 $[3, 1, 3]$ repetition codes[3], i.e., the dimension is 25. We map every vector $\mathbf{g}'_i$ to

[3]In the sequel, we denote this code construction as concatenated repetition code. For this $[75, 25, 3]$ linear code, the covering radius is 25, but we could see from this example that what matters is the average weight of the error vector, which is much smaller than 25.

the nearest codeword by simply selecting chunks of three consecutive bits and replace them by either 000 or 111. With probability $\frac{3}{4}$ we will change one position and with probability $\frac{1}{4}$ we will not have to change any position. In total we expect to change $(\frac{3}{4} \cdot 1 + \frac{1}{4} \cdot 0) \cdot 25$ positions. The expected weight of the length 75 vector $\mathbf{e}'_i$ is $\frac{1}{4} \cdot 75$, so the expected bias is $\epsilon' = \frac{1}{2}$. As $\mathbb{P}\left(x'_i = 1\right) = 0.1$, the expected number of non-zero positions in $\mathbf{x}'$ is 7.5. Assuming we have only $c = 6$ non-zero positions, we get

$$\mathbb{P}\left(\langle \mathbf{x}'', \mathbf{u}_i \rangle = z'_i\right) = \frac{1}{2} \cdot \left(1 + 0.8^{16} \cdot \left(\frac{1}{2}\right)^6\right) = \frac{1}{2} \cdot (1 + 2^{-11.15}).$$

For the repetition code, there are »bad events« that make the distinguishing to fail. When two of the errors in $\mathbf{x}''$ fall into the same concatenation, then the bias is zero. If there are three errors in the same concatenation, the the bias is negative. The probability for none of these events to happen is

$$\binom{25}{6} \cdot \binom{3}{1}^6 \cdot \binom{75}{6}^{-1} \approx 0.64.$$

4. In the last step we then run through 2^{25} values of $\mathbf{x}''$ and for each of them we compute how often $\mathbf{x}'' \cdot \mathbf{u}_i^{\mathrm{T}} = z'_i$ for $i = 1, \ldots, 3 \cdot 2^{21}$. Again since we use fast Walsh-Hadamard transform, the cost of this step is not much more than 2^{25} operations. The probability of having no more than 6 ones in $\mathbf{x}'$ is about 0.37. All bad events considered, we need to repeat the whole process a few times.

In comparison with other algorithms, the best approach we can find is the Kirchner [Kir11] and the Bernstein and Lange [BL13] approaches, where one can do up to 5 merging steps. Removing 21 bits in each step leaves 55 remaining bits. Using fast Walsh-Hadamard transform with $0.8^{-64} = 2^{20.6}$ samples, we can include another 21 bits in this step, but there are still 34 remaining variables that needs to be guessed.

Overall, the simple algorithm sketched above is outperforming the best previous algorithm using optimal parameter values[4].

[4] Adopting the same method to implement their overlapping steps, for the $(160, \frac{1}{10})$ LPN instance, the Bernstein-Lange algorithm and the new algorithm cost $2^{35.70}$ and $2^{33.83}$ bit operations, respectively. Thus, the latter offers an improvement with a factor roughly 4 to solve this small-scale instance.

We verified in simulation that the proposed algorithm works in practice, both in the LF1 and the LF2 setting using a rate $R = \frac{1}{3}$ concatenated repetition code.

7.4 ALGORITHM DESCRIPTION

Having introduced the key idea in a simplistic manner, we now formalize it by stating a new five-step LPN solving algorithm (see Algorithm 13) in detail. Its first three steps combine several well-known techniques on this problem, i.e., changing the distribution of secret vector [Kir11], sorting and merging to make the length of samples shorter [BKW03], and partial secret guessing [BL13], together. The efficiency improvement comes from a novel idea introduced in the last two subsections—if we employ a linear covering code and rearrange samples according to their nearest codewords, then the columns in the matrix subtracting their corresponding codewords lead to sparse vectors desired in the distinguishing process. We later propose a new distinguishing technique—subspace hypothesis testing, to remove the influence of the codeword part using fast Walsh-Hadamard transform. The algorithm consists of five steps, each described in separate subsections.

7.4.1 GAUSSIAN ELIMINATION

Recall that our LPN problem is given by $\mathbf{z} = \mathbf{x}\mathbf{G} + \mathbf{e}$, where $\mathbf{z}$ and $\mathbf{G}$ are known. We can apply an arbitrary column permutation π without changing the problem (but we change the error locations). A transformed problem is $\pi(\mathbf{z}) = \mathbf{x}\pi(\mathbf{G}) + \pi(\mathbf{e})$. This means that we can repeat the algorithm many times using different permutations, which very much resembles the operation of information-set decoding algorithms.

Continuing, we multiply by a suitable $k \times k$ matrix $\mathbf{D}$ to bring the matrix $\mathbf{G}$ to a systematic form, $\mathbf{G}_{\text{sys}} = \mathbf{D}\mathbf{G}$. The problem remains the same, except that the unknowns are now given by the vector $\tilde{\mathbf{x}} = \mathbf{x}\mathbf{D}^{-1}$. This is just a change of variables. As a second step, we also add the codeword $\begin{pmatrix} z_1 & z_2 & \cdots & z_k \end{pmatrix} \hat{\mathbf{G}}$ to our known vector $\mathbf{z}$, resulting in a received vector starting with k zero entries. Altogether, this corresponds to the change $\hat{\mathbf{x}} = \mathbf{x}\mathbf{D}^{-1} + \begin{pmatrix} z_1 & z_2 & \cdots & z_k \end{pmatrix}$.

Our initial problem has been transformed and the problem is now written as

$$\hat{\mathbf{z}} = \begin{pmatrix} \mathbf{0} & \hat{z}_{k+1} & \hat{z}_{k+2} \cdots & \hat{z}_n \end{pmatrix} = \hat{\mathbf{x}}\mathbf{G}_{\text{sys}} + \mathbf{e}, \tag{7.33}$$

where now $\mathbf{G}_{\text{sys}}$ is in systematic form. Note that these transformations do not affect the noise level. We still have a single noise variable added in every position.

TIME–MEMORY TRADE-OFF

Schoolbook implementation of the above Gaussian elimination procedure requires about $\frac{1}{2} \cdot n \cdot k^2$ bit operations; we propose however to reduce its complexity by using a more sophisticated time–memory trade-off technique. We store intermediate results in tables, and then derive the final result by adding several items in the tables together. The detailed description is as follows.

For a fixed s, divide the matrix $\mathbf{D}$ in $a = \lceil \frac{k}{s} \rceil$ parts, i.e.,

$$\mathbf{D} = \begin{pmatrix} \mathbf{D}_1 & \mathbf{D}_2 & \dots & \mathbf{D}_a \end{pmatrix}, \tag{7.34}$$

where $\mathbf{D}_i$ is a sub-matrix with s columns (except possibly the last matrix $\mathbf{D}_a$). Then store all possible values of $\mathbf{D}_i \mathbf{x}^{\mathrm{T}}$ for $\mathbf{x} \in \mathbb{F}_2^s$ in tables indexed by i, where $1 \leqslant i \leqslant a$. For a vector $\mathbf{g} = \begin{pmatrix} \mathbf{g}_1 & \mathbf{g}_2 & \dots & \mathbf{g}_a \end{pmatrix}$, the transformed vector is

$$\mathbf{D}\mathbf{g}^{\mathrm{T}} = \mathbf{D}_1 \mathbf{g}_1^{\mathrm{T}} + \mathbf{D}_2 \mathbf{g}_2^{\mathrm{T}} + \ldots + \mathbf{D}_a \mathbf{g}_a^{\mathrm{T}}, \tag{7.35}$$

where $\mathbf{D}_i \mathbf{g}_i^{\mathrm{T}}$ can be read directly from the table.

The cost of constructing the tables is about $\mathcal{O}(2^s)$, which can be negligible if memory in the later merge step is much larger. Furthermore, for each column, the transformation costs no more than $k \cdot a$ bit operations; so, this step requires

$$C_1 = (n-k) \cdot k \cdot a < n \cdot k \cdot a$$

bit operations in total if 2^s is much smaller than n.

7.4.2 MERGING COLUMNS

This next step consists of merging columns. The input to this step is $\hat{\mathbf{z}}$ and $\mathbf{G}_{\text{sys}}$. We write $\mathbf{G}_{\text{sys}} = \begin{pmatrix} \mathbf{I} & \mathbf{L}_0 \end{pmatrix}$ and process only the matrix $\mathbf{L}_0$. As the length of $\mathbf{L}_0$ is typically much larger than the systematic part of $\mathbf{G}_{\text{sys}}$, this is roughly no restriction at all. We then use the a sort-and-match technique as in the BKW algorithm, operating on the matrix $\mathbf{L}_0$. This process will give us a sequence of matrices denoted $\mathbf{L}_0, \mathbf{L}_1, \mathbf{L}_2, \ldots, \mathbf{L}_t$.

Let us denote the number of columns of $\mathbf{L}_i$ by $r(i)$, with $r(0) = r'(0) = n - k'$. Adopting the LF1 type technique, every step operating on columns will reduce the number of samples by 2^b, yielding that

$$m = r(t) = r(0) - t \cdot 2^b \iff n - k' = m + t \cdot 2^b \tag{7.36}$$

Using the setting of LF2, the number of samples is

$$\begin{aligned} m = r'(t) &= 2^b \cdot \binom{r'(t-1)/2^b}{2} \\ \Longrightarrow n - k' &\approx \sqrt[2^{t+1}]{2^{(b+1)(2^{t+1}-1)} \cdot m}. \end{aligned} \tag{7.37}$$

The expression for $r'(t)$ does not appear in [LF06], but it can be found in [BV15].

Apart from the process of creating the $\mathbf{L}_i$ matrices, we need to update the received vector in a similar fashion. A simple way is to put $\hat{\mathbf{z}}$ as a first row in the representation of $\mathbf{G}_{\text{sys}}$. This procedure will end with a matrix $\begin{pmatrix}\mathbf{I} & \mathbf{L}_t\end{pmatrix}$, where $\mathbf{L}_t$ will have all $t \cdot b$ last entries in each column all zero. By discarding the last $t \cdot b$ rows we have a given matrix of dimension $k - t \cdot b$ that can be written as $\mathbf{G}' = \begin{pmatrix}\mathbf{I} & \mathbf{L}_t\end{pmatrix}$, and we have a corresponding received vector $\mathbf{z}' = \begin{pmatrix}\mathbf{0} & z'_1 & z'_2 & \cdots & z'_m\end{pmatrix}$. The first $k' = k - t \cdot b$ positions are only affected by a single noise variable, so we can write

$$\mathbf{z}' = \mathbf{x}'\mathbf{G}_{\text{sys}} + \begin{pmatrix}e_1 & e_2 & \cdots & e_k & \tilde{e}_1 & \tilde{e}_2 & \cdots & \tilde{e}_m\end{pmatrix}, \tag{7.38}$$

for some unknown $\mathbf{x}'$ vector, where

$$\tilde{e}_i = \sum_{i_j \in \mathcal{T}_i,\ |\mathcal{T}_i| \leqslant 2^t} e_{i_j} \tag{7.39}$$

and $\mathcal{T}_i$ contains the positions that have been added up to form the $(k' + i)$th column of $\mathbf{G}'$. By the piling-up lemma, the bias for $\tilde{e}_i$ increases to ϵ^{2^t}. We denote the complexity of this step C_2, where

$$C_2 = \sum_{i=1}^{t} (k + 1 - i \cdot b) \cdot (n - i \cdot 2^b) \approx (k + 1) \cdot t \cdot n. \tag{7.40}$$

7.4.3 PARTIAL SECRET GUESSING

The previous procedure outputs $\mathbf{G}'$ with dimension $k' = k - t \cdot b$ and $m = n - k - t \cdot 2^b$ columns. We removed the bottom $t \cdot b$ bits of $\hat{\mathbf{x}}$ to form the length k' vector $\mathbf{x}'$, with $\mathbf{z}' = \mathbf{x}'\mathbf{G}' + \tilde{\mathbf{e}}$. We now divide $\mathbf{x}'$ into two parts:

$$\mathbf{x}' = \begin{pmatrix}\mathbf{x}'_1 & \mathbf{x}'_2\end{pmatrix}, \tag{7.41}$$

where $\mathbf{x}'_1$ is of length k''. In this step, we simply guess all vectors $\mathbf{x}_2 \in \mathcal{B}_2(k' - k'', p)$ for some p and update the observed vector $\mathbf{z}'$ accordingly. This transforms the problem to that of attacking a new smaller LPN problem of dimension k'' with the same number of samples. Firstly, note that this will only work if $w_{\text{H}}\left(\mathbf{x}'_2\right) \leqslant p$, and we denote this probability by $P(p, k' - k'')$. Secondly, we need to be able to distinguish a correct guess from incorrect ones and this is the task of the remaining steps. The complexity of this step is

$$C_3 = m \cdot \sum_{i=0}^{p} \binom{k' - k''}{i} i. \tag{7.42}$$

7.4.4 COVERING-CODING METHOD

In this step, we use a $[k'', l]$ linear code $\mathcal{C}$ with covering radius d_C to group the columns. That is, we rewrite

$$\mathbf{g}_i' = \mathbf{c}_i + \mathbf{e}_i', \tag{7.43}$$

where $\mathbf{c}_i$ is the nearest codeword in $\mathcal{C}$, and $w_{\mathrm{H}}\left(\mathbf{e}_i'\right) \leqslant d_C$. The employed linear code is characterized by a systematic generator matrix

$$\mathbf{F} = \begin{pmatrix} \mathbf{I} & \mathbf{A} \end{pmatrix} \in \mathbb{F}_2^{l \times k''}, \tag{7.44}$$

that has the corresponding parity-check matrix

$$\mathbf{H} = \begin{pmatrix} \mathbf{A}^{\mathrm{T}} & \mathbf{I} \end{pmatrix} \in \mathbb{F}_2^{(k''-l) \times k''}. \tag{7.45}$$

There are several ways to select a code. An efficient way of realizing the above grouping idea is by a table-based syndrome-decoding technique. The procedure is as follows:

1. We construct a constant-time query table containing $2^{k''-l}$ items, in each of which stores the syndrome and its corresponding minimum-weight error vector.

2. If the syndrome $\mathbf{H}\mathbf{g}_i'^{\mathrm{T}}$ is computed, we then find its corresponding error vector $\mathbf{e}_i'$ by checking in the table; adding them together yields the nearest codeword $\mathbf{c}_i$.

The remaining task is to calculate the syndrome efficiently. We sort the vectors $\mathbf{g}_i'$ according to the first l bits, where $0 \leqslant i \leqslant m$, and group them into 2^l partitions denoted by $\mathcal{P}_j$ for $1 \leqslant j \leqslant 2^l$. Starting from the partition $\mathcal{P}_1$ whose first l bits are all zero, we can derive the syndrome by reading its last $k'' - l$ bits without any additional computational cost. If we know one syndrome in $\mathcal{P}_j$, we then can compute another syndrome in the same partition within $2(k'' - l)$ bit operations, and another in a different partition whose first l-bit vector has Hamming distance 1 from that of $\mathcal{P}_j$ within $3(k'' - l)$ bit operations. Therefore, the complexity of this step is

$$C_4 = (k'' - l) \cdot (2m + 2^l). \tag{7.46}$$

Notice that the selected linear code determines the syndrome table, which can be pre-computed within complexity $\mathcal{O}(k'' \cdot 2^{k''-l})$. The optimal parameter suggests that this cost is acceptable compared with the total attacking complexity.

The bias ϵ' in $\mathbf{e}'_i$ is determined by (7.29). This plays an important role in the later hypothesis testing step: if we rearrange the columns $\mathbf{e}'_i$ as a matrix, then it is sparse; therefore, we can view the ith value in one column as a random variable $X_i \sim \mathsf{Ber}_{\varphi(c)}$, where d is the expected distance. Naturally, we can bound it by the covering radius[5]. Moreover, if the bias is large enough, then it is reasonable to consider X_i, for $1 \leqslant i \leqslant i_1$, as independent variables. We have given an illustration of the query matrix at this step in Figure 7.3.

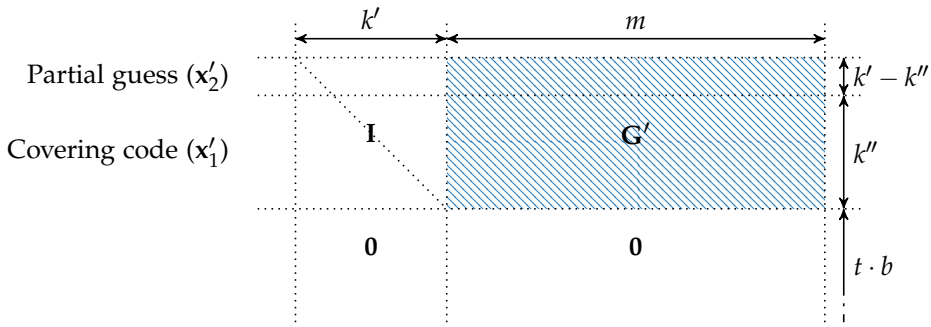


Figure 7.3: After the columns have been merged t times, we have a matrix as shown above. In the upper part, we perform the partial secret guessing. The remaining part will be projected (with distortion) into a smaller space of dimension l using a covering code.

For some instances, optimal parameters does not allow for building the full syndrome table, i.e., when $k'' \cdot 2^{k''-l}$ becomes too large. We may then split the search space into two (or several) separate spaces by using a concatenated code construction. As an example, let $\mathcal{C}'$ be a concatenation of two $[k''/2, l/2]$ linear codes. Then, the syndrome tables can be built in $\mathcal{O}(k'' \cdot 2^{k''/2-l/2})$ time and memory. Assuming that the two codes are identical and their sphere-covering bound is d', then the expected weight of $\mathbf{e}'_i$ is $2d'$.

7.4.5 SUBSPACE HYPOTHESIS TESTING

In the subspace hypothesis testing step, we group the (processed) samples $(\mathbf{g}'_i, z'_i)$ in sets $L(\mathbf{c}_i)$ according to their nearest codewords and define the function $f_L(\mathbf{c}_i)$ as

$$f_L(\mathbf{c}_i) = \sum_{(\mathbf{g}'_i, z'_i) \in L(\mathbf{c}_i)} (-1)^{z'_i}. \tag{7.47}$$

[5]In the sequel, we replace the covering radius by the sphere-covering bound to estimate the expected distance d, i.e., d is the smallest integer, s.t. $\sum_{i=0}^{d} \binom{k''}{i} > 2^{k''-l}$. We give more explanation in Section 7.7.

The employed systematic linear code $\mathcal{C}$ describes a bijection between the linear space $\mathbb{F}_2^l$ and the set of all codewords in $\mathbb{F}_2^{k''}$, and moreover, due to its systematic feature, the corresponding information vector appears explicitly in their first l bits. We can thus define a new function

$$g(\mathbf{u}) = f_L(\mathbf{c}_i), \tag{7.48}$$

such that $\mathbf{u}$ represents the first l bits of $\mathbf{c}_i$ and exhausts all points in $\mathbb{F}_2^l$.

The Walsh transform of g is defined as

$$G(\mathbf{v}) = \sum_{\mathbf{u}\in\mathbb{F}_2^l} g(\mathbf{u})(-1)^{\langle \mathbf{v},\mathbf{u}\rangle}. \tag{7.49}$$

Here we exhaust all candidates of $\mathbf{v} \in \mathbb{F}_2^l$ by computing the Walsh transform.

The following lemma illustrates the reason why we can perform hypothesis testing on the subspace $\mathbb{F}_2^l$.

Lemma 7.3 There exits a unique vector $\mathbf{v} \in \mathbb{F}_2^l$ s.t.,

$$\langle \mathbf{v}, \mathbf{u}\rangle = \langle \mathbf{x}', \mathbf{c}_i\rangle. \tag{7.50}$$

Proof. As $\mathbf{c}_i = \mathbf{u}\mathbf{F}$, we obtain

$$\langle \mathbf{x}', \mathbf{c}_i\rangle = \mathbf{x}'(\mathbf{u}\mathbf{F})^{\mathrm{T}} = \mathbf{x}'\mathbf{F}^{\mathrm{T}}\mathbf{u}^{\mathrm{T}} = \langle \mathbf{x}'\mathbf{F}^{\mathrm{T}}, \mathbf{u}\rangle. \tag{7.51}$$

Thus, we construct the vector $\mathbf{v} = \mathbf{x}'\mathbf{F}^{\mathrm{T}}$ that fulfills the requirement. On the other hand, the uniqueness is obvious. ■

Before we continue to go deeper into the details of the attack, we will now try illustrate how the subspace hypothesis test is performed. Consider the following.

Rewrite $\mathbf{g}_i'$ as codeword $\mathbf{c}_i = \mathbf{u}'\mathbf{F}$ and discrepancy $\mathbf{e}_i'$

$$\begin{pmatrix} * \\ \vdots \\ * \\ \hline y_i' \\ \hline * \\ \vdots \end{pmatrix} = \underbrace{\begin{pmatrix} x_1 \\ \vdots \\ x_{k''} \\ \hline 0 \\ \vdots \end{pmatrix}^{\mathrm{T}}}_{\text{Secret } \mathbf{x}'} \underbrace{\left(\begin{array}{c|c|c} * & g_1 & * \\ \vdots & \vdots & \vdots \\ * & g_{k''} & * \\ \hline & 0 & \\ & \vdots & \end{array}\right)}_{\text{Query matrix}} = \begin{pmatrix} x_1 \\ \vdots \\ x_{k''} \\ \hline 0 \\ \vdots \end{pmatrix}^{\mathrm{T}} \left(\begin{array}{c|c|c} * & (\mathbf{u}'\mathbf{F} + \mathbf{e}_i')_1 & * \\ \vdots & \vdots & \vdots \\ * & (\mathbf{u}'\mathbf{F} + \mathbf{e}_i')_{k''} & * \\ \hline & 0 & \\ & \vdots & \end{array}\right).$$

As a next step, we can separate the discrepancy $\mathbf{e}'_i$ from $\mathbf{u}'\mathbf{F}$, which yields

$$\begin{pmatrix} x_1 \\ \vdots \\ x_{k''} \\ \hline 0 \\ \vdots \end{pmatrix}^{\mathrm{T}} \left(\begin{array}{cc|c|c} * & * & (\mathbf{u}'\mathbf{F})_1 & * \\ \vdots & \vdots & \vdots & \vdots \\ * & * & (\mathbf{u}'\mathbf{F})_{k''} & * \\ \hline & & 0 & \\ & & \vdots & \end{array}\right) = \begin{pmatrix} * \\ \vdots \\ * \\ \hline y'_i + \langle \mathbf{x}, \mathbf{e}'_i \rangle \\ \hline * \\ \vdots \end{pmatrix}.$$

We now see that the dimension of the problem has been reduced, i.e, $\mathbf{x}'_1\mathbf{F}^{\mathrm{T}} \in \mathbb{F}_2^l$, where $l < k''$. A simple transformation yields

$$\begin{pmatrix} (\mathbf{x}'_1\mathbf{F}^{\mathrm{T}})_1 \\ \vdots \\ (\mathbf{x}'_1\mathbf{F}^{\mathrm{T}})_l \\ \hline 0 \\ \vdots \end{pmatrix}^{\mathrm{T}} \left(\begin{array}{cc|c|c} * & * & u'_1 & * \\ \vdots & \vdots & \vdots & \vdots \\ * & * & u'_l & * \\ \hline & & 0 & \\ & & \vdots & \end{array}\right) = \begin{pmatrix} * \\ \vdots \\ * \\ \hline y'_i + \langle \mathbf{x}'_1, \mathbf{e}'_i \rangle \\ \hline * \\ \vdots \end{pmatrix}.$$

Since $w_{\mathrm{H}}\left(\mathbf{e}'_i\right) \leqslant w$ and $w_{\mathrm{H}}\left(\mathbf{x}'_i\right) \approx \eta \cdot k''$, the contribution from $\langle \mathbf{x}'_1, \mathbf{e}'_i \rangle$ is very small. Note that $\mathbf{e}'_i$ is the error from the above procedure, and that we did not include the error from the oracle and the merging procedure. Recall that the sequence received from oracle is $z_i = y_i + e_i$, that after merging the columns of $\mathbf{G}$ becomes $z'_i = y'_i + \tilde{e}_i$. All things considered (all sources of error piled on the sequence), we have

$$z'_i + \langle \mathbf{x}', \mathbf{e}_i \rangle = y_i + \tilde{e}_i + \langle \mathbf{x}'_1, \mathbf{e}'_i \rangle. \tag{7.52}$$

Given the candidate $\mathbf{v}$, $G(\mathbf{v})$ is the difference between the number of predicted 0:s and the number of predicted 1:s for the bit $z'_i + \langle \mathbf{x}', \mathbf{e}_i \rangle$. If $\mathbf{v}$ is the correct guess, then it is Bernouilli distributed according to

$$\frac{1}{2} \cdot \left(1 + \epsilon^{2^t} \cdot (1 - 2\varphi(c))\right) = \frac{1}{2} \cdot \left(1 + \epsilon^{2^t} \cdot \epsilon'\right), \tag{7.53}$$

where c is the weight of $\mathbf{x}'$; otherwise, it is considered random. Thus, the best candidate $\mathbf{v}_{\mathrm{opt}}$ is the one that maximizes the absolute value of $G(\mathbf{v})$, i.e.,

$$\mathbf{v}_{\mathrm{opt}} = \arg\max_{\mathbf{v} \in F_2^l} |G(\mathbf{v})|, \tag{7.54}$$

and we need approximately

$$2 \ln 2 \cdot \frac{-l \cdot \log_2 \theta}{(\epsilon^{2^t} \cdot \epsilon')^2} \tag{7.55}$$

samples to distinguish these two cases, where $\epsilon' = 1 - 2\varphi(c)$. Here, we set $\theta = \frac{1}{4}$ and obtain

$$\frac{4 \ln 2 \cdot l}{(\epsilon^{2^t} \cdot \epsilon')^2}. \tag{7.56}$$

Using this setting, the probability of error (i.e., both α and β) is actually less than 2^{-10} for the $(512, \frac{1}{8})$ instance [Sel08].

Since the weight w is unknown, we assume that $w \leqslant c$ and then query for samples. If the assumption is valid, we can distinguish the two distributions correctly; otherwise, we obtain a false positive which can be recognized without much cost, and then choose another permutation to run the algorithm again. The procedure will continue until we find the secret vector $\mathbf{x}$.

We use the fast Walsh-Hadamard transform technique to accelerate the distinguishing step. As the hypothesis testing runs for every guess of $\mathbf{x}'_2$, the overall complexity of this step is

$$C_5 \stackrel{\text{def}}{=} l \cdot 2^l \cdot \sum_{i=0}^{p} \binom{k' - k''}{i}. \tag{7.57}$$

7.5 ANALYSIS

In the previous section we already indicated the complexity of each step. We now put it together in a single complexity estimate. We first formulate the formula for the possibility of having at most w errors in j positions $P(w, j)$, which follows a binomial distribution, i.e.,

$$P(w, j) \stackrel{\text{def}}{=} \sum_{i=0}^{w} \binom{j}{i} (1 - \eta)^{j-i} \cdot \eta^i. \tag{7.58}$$

The complexity consists of three parts:

- **Inner complexity**: The complexity of each step in the algorithm, i.e.,

  $$C = C_1 + C_2 + C_3 + C_4 + C_5. \tag{7.59}$$

  These steps will be performed every iteration.

- **Guessing**: The probability of making a correct guess on the weight of $\mathbf{x}'_2$, i.e.,

  $$P_{\text{guess}} \stackrel{\text{def}}{=} \mathbb{P}\left(w_{\text{H}}\left(\mathbf{x}'_2 \leqslant p\right)\right) = P(p, k' - k''). \tag{7.60}$$

- **Testing**: The probability that the subspace hypothesis test succeeds for all different information patterns. In an ideal (non-concatenated) construction, this probability is given by

$$P_{\text{test}} \stackrel{\text{def}}{=} \mathbb{P}\left(w_{\text{H}}\left(\mathbf{x}_1' \leqslant c\right)\right) = P(c, k''). \tag{7.61}$$

The success probability in one iteration is $P(p, k' - k'') \cdot P(c, k'')$. The presented algorithm is of the Las Vegas type, and in each iteration, the complexity accumulates step by step. Hence, by Proposition 1.11, the following theorem is revealed.

Theorem 7.4 (The complexity of Algorithm 13) Let n be the number of samples required and a, b, c, t, k'', l, p be algorithm parameters. For the LPN instance with parameter (k, η), the number of bit operations required for a successful run of the new attack, denoted $C^*(a, b, c, t, k'', l, p)$, is equal to

$$\begin{aligned} &P_{\text{guess}}^{-1} \cdot P_{\text{test}}^{-1} \cdot \Bigg\{ C_{\mathsf{Gauss}}(n, k) + (k+1) \cdot t \cdot n \\ &+ \sum_{i=0}^{p} \binom{k' - k''}{i} (m \cdot i + l \cdot 2^l) + (k'' - l) \cdot (2m + 2^l) \Bigg\}, \end{aligned} \tag{7.62}$$

under the condition that

$$m \geqslant \frac{4 \ln 2 \cdot l}{(\epsilon^{2^t} \cdot \epsilon')^2}. \tag{7.63}$$

Proof. The complexity of one iteration is given by $C_1 + C_2 + C_3 + C_4 + C_5$, with $C_1 = C_{\mathsf{Gauss}}(n, k)$. The expected number of iterations is the inverse of $P_{\text{guess}} \cdot P_{\text{test}} = P(p, k' - k'') \cdot P(c, k'')$ (see Proposition 1.11), hence

$$C^* = \frac{C_{\mathsf{Gauss}}(n, k) + C_2 + C_3 + C_4 + C_5}{P(p, k' - k'') \cdot P(c, k'')} \tag{7.64}$$

Substituting the formulas into the above will complete the proof. The condition (7.67) ensures that we have enough samples to determine the correct guess with high probability. ■

Until now, we have only considered to use a single code for the covering code part. In some cases, performing syndrome decoding may be too expensive for optimal parameters and to overcome this, we need to use a concatenated code construction.

In the previous analysis, we assumed that the number of errors in the secret was less than c, in order to bound the bias $\epsilon' = 1 - 2\varphi(c)$. When a concatenated construction is used, we have seen that some problems may arise. Recall

the $[3, 1, 3]$ repetition code. When the number of errors of the secret exceeded 1 in one concatenation the samples will be zero or negative biased and therefore, distinguishing will definitely fail. Taking such bad events into account, we get an additional factor piled on the complexity. This factor depends completely on the code construction.

We may set an explicit lower-bound on the bias $\epsilon' \geqslant \gamma$ from the covering code part, which is attained by only a certain set $\mathcal{E}_\gamma$ of (good) error patterns in the secret. For a concatenation of two codes, we have divided the vector into two parts

$$\mathbf{x}_1' = \begin{pmatrix} \hat{\mathbf{x}}_1 & \hat{\mathbf{x}}_2 \end{pmatrix} \tag{7.65}$$

and hence, we may write the success probability $P_{\text{test}} \stackrel{\text{def}}{=} \mathbb{P}\left(\mathbf{x}_1' \in \mathcal{E}_\gamma\right)$ as

$$\sum_{(\hat{\mathbf{x}}_1\ \hat{\mathbf{x}}_2) \in \mathcal{E}_\gamma} \eta^{k''/2 - w_{\mathrm{H}}(\hat{\mathbf{x}}_2)} (1-\eta)^{w_{\mathrm{H}}(\hat{\mathbf{x}}_1)} \cdot \eta^{k''/2 - w_{\mathrm{H}}(\hat{\mathbf{x}}_2)} (1-\eta)^{w_{\mathrm{H}}(\hat{\mathbf{x}}_1)}. \tag{7.66}$$

As a result, the complexity expression for a concatenated construction changes in the probability P_t.

The complexity C_4 remains the same, but the pre-computation of the syndrome tables has a lowered complexity since the codes are smaller and can be treated separately. Since the pre-computation complexity $\mathcal{O}(k'' \cdot 2^{k''/2 - l/2})$ must be less or match the total attacking complexity, the lowered time complexity allows for looser constraints on the algorithm parameters. In the concatenated construction, the number of required samples changes as it is determined by the lower bound γ, i.e.,

$$m \geqslant \frac{4 \ln 2 \cdot l}{(\epsilon^{2^t} \cdot \gamma)^2}. \tag{7.67}$$

Apart from these differences, the complexity expression is the same as for the non-concatenated construction.

7.6 RESULTS

We now present numerical results of the new algorithm attacking three key LPN instances, as shown in Table 7.2. All aiming for achieving 80-bit security, the first one is with parameter $(512, \frac{1}{8})$, widely accepted in various LPN-based cryptosystems (e.g., HB$^+$ [JW05], HB$^\#$ [GRS08a], LPN-C [GRS08b]) after the suggestion from Levieil and Fouque [LF06]; the second one is with increased length $(532, \frac{1}{8})$, adopted as the parameter of the irreducible Ring-LPN instance employed in Lapin [HKL$^+$12]; and the last one is a new design parameter we recommend to use in the future. The attacking details on different protocols

will be given later. We note that the new algorithm has significance not only for the above applications but also for some LPN-based cryptosystems without explicit parameter settings (e.g., [DKPW12] [KPC+11]).

Instance	Parameters								$\log_2 C^*$
	t	a	b	l	k''	p	c	$\log_2 n$	
$(512, \frac{1}{8})$	5	9	62	60	180	2	19	65.31	79.72
$(532, \frac{1}{8})$	5	9	65	63	189	2	19	66.45	82.66
$(592, \frac{1}{8})$	5	10	70	76	210	1	23	74.49	89.88

Table 7.2: The complexity for solving different LPN instances with the new algorithm when using LF2 merging heuristic.

In [LF06], Levieil and Fouque proposed an active attack on HB$^+$ by choosing the random vector **a** from the reader to be **0**. To achieve 80-bit security, they suggested to adjust the lengths of secret keys to 80 and 512, respectively, instead of being both 224. Its security is based on the assumption that the LPN instance with parameter $(512, \frac{1}{8})$ can resist attacks in 2^{80} bit operations. But we break it in $2^{79.72}$ bit operations, thereby yielding an active attack on 80-bit security of HB$^+$ authentication protocol straightforwardly.

Using similar structures, Gilbert *et al.* proposed two different cryptosystems, one for authentication (HB$^\#$) and the other for encryption (LPN-C). By setting the random vector from the reader and the message vector to be both **0**, we obtain an active attack on HB$^\#$ authentication protocol and a chosen-plaintext-attack on LPN-C, respectively. As their protocols consist of both secure version (random-HB$^\#$ and LPN-C) and efficient version (HB$^\#$ and Toeplitz LPN-C), we need to analyze separately.

Toeplitz matrix is a matrix in which each ascending diagonal from left to right is a constant. Thus, when employing a Toeplitz matrix as the secret, if we attack its first column successively, then only one bit in its second column is unknown. So the problem is transformed to that of solving a new LPN instance with parameter $(1, \frac{1}{8})$. We then deduce the third column, the fourth column, and so forth. The typical parameter settings of the number of the columns (denoted by m) are 441 for HB$^\#$, and 80 (or 160) for Toeplitz LPN-C. In either case, the cost for determining the vectors except for the first column is bounded by 2^{40}, negligible compared with that of attacking one $(512, \frac{1}{8})$ LPN instance. Therefore, we break the 80-bit security of these »efficient« versions that use Toeplitz matrices.

If the secret matrix is chosen totally at random, then there is no simple connection between different columns to exploit. One strategy is to attack

column by column, thereby deriving an algorithm whose complexity is that of attacking a $(512, \frac{1}{8})$ LPN instance multiplied by the number of the columns. That is, if $m = 441$, then the overall complexity is about $2^{79.72} \cdot 441 \approx 2^{88.50}$. We may slightly improve the attack by exploiting that the different columns share the same random vector in each round.

7.6.1 LAPIN WITH AN IRREDUCIBLE POLYNOMIAL

In [HKL+12], Heyse *et al.* use a $(532, \frac{1}{8})$ Ring-LPN instance with an irreducible polynomial to achieve 80-bit security. We show here that this parameter setting is not secure enough for Lapin to thwart attacks on the level of 2^{80}. Although the new attack on a $(532, \frac{1}{8})$ LPN instance requires $2^{82.66}$ bit operations, larger than 2^{80}, there are two key issues to consider:

- Ring-LPN is believed to be no harder than the standard LPN problem. For the instance in Lapin using a quotient ring modulo the irreducible polynomial $x^{532} + x + 1$, it is possible to optimize the procedure for inverting a ring element, thereby resulting in a more efficient attack than the generic one.

- We describe the merging steps using setting of LF1 in the new algorithm, but we obtain a more efficient attack in practice when adopting the LF2 heuristic, whose effectiveness has been stated and proven in the implementation part of [LF06]. We suggest to increase the size of the employed irreducible polynomial in Lapin for 80-bit security.

7.7 MORE ON THE COVERING-CODING METHOD

In this section, we describe some more aspects of the covering-coding technique, thus emphasizing the most novel and essential step in the new algorithm.

SPHERE-COVERING BOUND

We use sphere-covering bound (Theorem 2.1) – for two reasons – to estimate the bias ϵ' contributed by the new technique.

- Firstly, there is a well-known conjecture [CHL97] in coding theory, i.e., the covering density approaches 1 asymptotically if the code length goes to infinity. Thus, it is sensible to assume for a »good« code, when the code length k'' is relatively large.

- Secondly, we could see from the previous example that the key feature desired is a linear code with low average error weights, which is smaller

than its covering radius. From this perspective, the sphere-covering bound brings us a good estimation.

By concatenating five $[23, 12]$ Golay codes, we construct a $[115, 60]$ linear code with covering radius 15. Its expected weight of error vector is quite close to the sphere-covering bound for this parameter (with gap only 1). We believe in the existence of linear codes with length around 125, rate approximately $\frac{1}{2}$ and average error weight that reaches the sphere-covering bound.

ATTACKING PUBLIC-KEY CRYPTOGRAPHY

We know various decodable covering codes that could be employed in the new algorithm, e.g., rate about $\frac{1}{2}$ linear codes that are table-based syndrome decodable, concatenated codes built on Hamming codes, Golay codes and repetition codes, etc.. For the cryptographic schemes targeted by the proposed attack, i.e., HB variants, LPN-C, and Lapin with an irreducible polynomial, the first three are efficient; but in the realm of public-key cryptography (e.g., schemes proposed by Alekhnovich [Ale03], Damgård and Park [DP12], Duc and Vaudenay [DV13]), the situation alters. For these systems, their security is based on LPN instances with huge secret length (tens of thousands) and extremely low error probability (less than half a percent), so due to the competitive average weight of the error vector shown by the previous example in Section 7.3.1, the concatenation of repetition codes with much lower rate seems more applicable—by low-rate codes, we remove more bits when using the covering-coding method.

ALTERNATIVE MERGING PROCEDURE

Although the covering-coding method is employed only once in the new algorithm, we could derive numerous variants, and among them, one may find a more efficient attack. For example, we could replace one or two steps in the later stage of the merging procedure by adding two vectors decoded to the same codeword together. This alternative technique is similar to that invented by Lamberger *et al.* in [LMRS12] [LT13a] for finding near-collisions of hash function. By this procedure, we could eliminate more bits in one step at the cost of increasing the error rate; this is a trade-off, and the concrete parameter setting should be analyzed more thoroughly later.

7.8 SUMMARY

We have described a new algorithm for solving the LPN problem that employs an approximation technique using covering codes together with a subspace

hypothesis testing technique to determine the value of linear combinations of the secret bits. Complexity estimates show that the algorithm beats all the previous approaches, and in particular, we can present academic attacks on instances of LPN that has been suggested in different cryptographic primitives.

The new technique has only been described in a rather simplistic manner, due to space limitations. There are a few obvious improvements, one being the use of soft decoding techniques and another one being the use of more powerful constructions of good codes. There are also various modified versions that need to be further investigated. One such idea is to use the new technique inside a merge step, thereby removing more bits in each step at the expense of introducing another contribution to the bias. An interesting open problem is whether these ideas can improve the asymptotic behavior of the BKW algorithm.

8

LPN over a Polynomial Ring

"Hallo, Rabbit", he [Pooh] said, "is that you?"
"Let's pretend it isn't", said Rabbit, "and see what happens."
– *A. A. Milne, Winnie-the-Pooh*

In the previous chapter, we investigated the general LPN problem and how to solve it using various techniques. In this chapter, we will focus on a special case of LPN called Ring-LPN, which is restricted to elements over a polynomial ring.

Ring-LPN is an interesting building block when constructing cryptographic primitives. For instance, the two-round identification protocol called Lapin proposed by Heyse *et al.* [HKL+12][1] is based on Ring-LPN. Using the inherent properties of rings, the proposed protocol becomes very efficient and well-suited for use in constrained environments. Briefly, in Ring-LPN, the oracle returns instead elements v, $v = s \cdot r + e$, from a polynomial ring $\mathbb{F}_2[x]/(f)$, i.e., $v, s, r, e \in \mathbb{F}_2[x]/(f)$. The problem can use either an irreducible polynomial f, of a reducible one. The choice of a reducible polynomial can make use of the *Chinese Remainder Theorem*[2] (CRT) to provide a very efficient implementation of the cryptographic primitive.

We propose a new algorithm to solve the reducible case of Ring-LPN. By investigating more on the properties of the ring structure and the reducibility of the polynomial, we demonstrate that if the minimum weight of the linear code defined by the CRT transform is low, then the problem is effortless to solve, hence providing a design criteria for cryptosystems based on the hardness of Ring-LPN with a reducible polynomial.

[1] Clarification to the non-French-speaking reader: Lapin deciphers to rabbit.
[2] Readers unfamiliar with CRT may refer to virtually any textbook on number theory.

We then specify the attack for Lapin [HKL+12] and obtain a complexity gain that makes it possible to break the claimed 80-bit security. In Table 8.2, we compare the complexity of our algorithm with the best known algorithms[3] designed to solve LPN and Ring-LPN. The time complexity is measured in bit operations and memory complexity is measured in bits.

ATTACKS ON RING-LPN

As the Ring-LPN instances are standard LPN instances, the attacking algorithm for the latter one is applicable to its ring instance. The pioneering researchers (e.g., Lapin – see Section 8.3) used the hardness level of the LPN problem obtained from [LF06] to measure the security of their authentication protocol based on the Ring-LPN problem. Almost at the same time, Bernstein and Lange [BL13] realized that simply ignoring the ring structure is a bad idea since this special algebraic property may reveal information about the secret, and subsequently derived an improved attack taking advantage of both the ring structure and Kirchner's technique. Their attack is generic since it applies to Ring-LPN implemented with both reducible and irreducible polynomials, and is advantageous in the sense of memory costs as well as query complexity. However, even for the time-optimized case[4], with around 2^{81} bits of memory, it requires quite a large number of bit operations, i.e., about 2^{88}, far away from breaking the 80-bit security of Lapin protocol.

8.1 THE RING-LPN PROBLEM

Consider a polynomial $f(x)$ over $\mathbb{F}_2$ (simply denoted f). The degree of f is denoted by $\deg f$. For any two polynomials f, g in the quotient ring $\mathbb{F}_2[x]$, long division of polynomials tells us that there are unique polynomials q, r such that $g = qf + r$. The unique polynomial $r \in \mathbb{F}_2[x]$ with $\deg r < \deg f$ is the representative of g in the quotient ring $\mathbb{F}_2[x]/(f)$ and is denoted $g \bmod f$. We define R to be the quotient ring $\mathbb{F}_2[x]/(f)$. So R consists of all polynomials in $\mathbb{F}_2[x]$ of degree less than $\deg f$ and arithmetics are done modulo f.

If the polynomial f factors such that every factor is of degree strictly less than f, then the polynomial f is said to be *reducible*; otherwise, it is said to be *irreducible*. When a reducible polynomial factors as $f = f_1 f_2 \cdots f_m$, where f_i is relatively prime to f_j for $1 \leqslant i, j \leqslant m$ and $i \neq j$, there exists a unique representation for every $r \in R$ according to the Chinese Remainder Theorem,

[3] We choose parameters to optimize their time complexity.

[4] We found that with parameters $q = 2^{58.59}$, $a = 6$, $b = 65$, $l = 53$ and $W = 4$, the complexity is $2^{85.9}$. The time complexity is slightly lower than stated in [BL13].

i.e.,

$$r \mapsto (r \bmod f_1, r \bmod f_2, \ldots, r \bmod f_m). \tag{8.1}$$

8.1.1 FORMAL DEFINITION

Being a subclass of LPN, the Ring-LPN problem is defined similarly. Fix an unknown value $s \in R$, where $s \xleftarrow{\$} R$. We can request samples depending on s through an oracle, which we define as follows.

Definition 8.1 (Ring-LPN **oracle)** A Ring-LPN oracle $\Pi^{\epsilon}_{\text{Ring-LPN}}$ for an unknown polynomial $s \in R$ with $\eta \in (0, \frac{1}{2})$ returns pairs of the form

$$(r,\ r \cdot s + e)\,, \tag{8.2}$$

where $r \xleftarrow{\$} R$ and $e \xleftarrow{\$} \text{Ber}^R_\eta$ is a low-weight ring element chosen with Bernoulli distribution over $\mathbb{F}_2$. An extreme case is, when η is exactly $\frac{1}{2}$, that the oracle $\Pi^0_{\text{Ring-LPN}}$ outputs a random sample distributed uniformly on $R \times R$.

The problem is now to recover the unknown value s after a number q of queries to the oracle. We define the search problem version of Ring-LPN in the following way.

Problem 8.1 (Ring-LPN**)** The search problem of Ring-LPN is said to be (t, q, θ)-solvable if there exists an algorithm $\mathcal{A}(\Pi^{\epsilon}_{\text{Ring-LPN}})$ that can find the unknown polynomial $s \in R$ in time at most t and using at most q oracle queries such that

$$\mathbb{P}\left(\mathcal{A}(\Pi^{\epsilon}_{\text{Ring-LPN}}) = s\right) \geqslant \theta. \tag{8.3}$$

The decisional Ring-LPN assumption, states that it is hard to distinguish between uniformly random samples and samples from the oracle $\Pi^{\epsilon}_{\text{Ring-LPN}}$. We can express the decision problem as follows.

Problem 8.2 (Decisional Ring-LPN**)** The decision problem of Ring-LPN is said to be (t, q, θ)-solvable if there exists an algorithm $\mathcal{A}'$ such that

$$\left|\mathbb{P}\left(\mathcal{A}'(\Pi^{\epsilon}_{\text{Ring-LPN}}) = \text{yes}\right) - \mathbb{P}\left(\mathcal{A}'(\Pi^{0}_{\text{Ring-LPN}}) = \text{yes}\right)\right| \geqslant \theta, \tag{8.4}$$

and $\mathcal{A}'$ is running in time t and making q queries.

The hardness of Ring-LPN is unknown, but the LPN problem has been shown to be **NP**-hard in the worst case. The assumption is that Ring-LPN is also hard.

In the paper by Heyse *et al.* [HKL+12], it was proposed to use Ring-LPN as the underlying hard problem to build an authentication protocol. The security relies on the assumption that Ring-LPN is as hard as LPN. However, this is a conjecture as there is only a reduction from Ring-LPN to LPN, but not the converse. In the following, we show how to reduce Ring-LPN to LPN.

TRANSFORMING RING-LPN TO LPN

Given the polynomial r with $\deg r \leqslant t$, we denote $\mathbf{r}$ as its coefficient vector, i.e., if r equals $\sum_{i=0}^{t} r_i x^i$, then $\mathbf{r} = \begin{pmatrix} r_0 & r_1 & \cdots & r_t \end{pmatrix}$. With this notation, we can define a mapping from one Ring-LPN instance to d standard LPN instances represented in the following matrix form:

$$\begin{array}{cccc} \tau: & R \times R & \longrightarrow & \mathbb{F}_2^{t\times t} \times \mathbb{F}_2^t \\ & \cup & & \cup \\ & (r, r\cdot s + e) & \longmapsto & (\mathbf{A}, \mathbf{A}\mathbf{s} + \mathbf{e}), \end{array} \tag{8.5}$$

where the ith column of the matrix $\mathbf{A}$ is the transposed coefficient vector of $r \cdot x^i \bmod f$.

RING-LPN WITH A REDUCIBLE POLYNOMIAL

We consider the Ring-LPN problem with the ring $R = \mathbb{F}_2[x]/(f)$, where the polynomial f factors into distinct irreducible factors over $\mathbb{F}_2$. That is, the polynomial f is written as $f = f_1 f_2 \cdots f_m$, where each f_i is irreducible. One of the specified instances of Lapin [HKL+12] uses a product of five different irreducible polynomials. This instance is the main target for cryptanalysis.

8.2 A NEW ALGORITHM FOR RING-LPN WITH REDUCIBLE POLYNOMIAL

We aim to provide a description of a new algorithm (Algorithm 14) for the Ring-LPN problem with a reducible polynomial, originally proposed in [GJL15]. First, we reduce the problem into a smaller one, while keeping the error at a reasonable level. Then, we further reduce the unknown variables using well-established collision techniques. The last step consists of exhausting the remaining unknown variables in an efficient way.

8.2.1 A LOW-WEIGHT CODE FROM THE CRT MAP

Let $f = f_1 f_2 \cdots f_m$ be a reducible polynomial. In the following, the underlying irreducible polynomial f_i in the quotient ring $\mathbb{F}_2[x]/(f_i)$ is fixed. Therefore, without loss of generality, we denote it as f_1 for notational simplicity. Let $\deg f = t$ and $\deg f_1 = l$.

Proposition 8.3 There exists a (surjective) linear map from $\psi : R \to \mathbb{F}_2[x]/(f_1)$ determined by the CRT transform. The linear map can be described as a $[t, l]$ linear code with generator matrix $\mathbf{G}_\psi$, in which the ith column is the transposed coefficient vector of the polynomial $x^{i-1} \bmod f_1$.

More specifically, a received sample $(r, r \cdot s + e)$ from the Ring-LPN oracle $\Pi^{\epsilon}_{\text{Ring-LPN}}$ can be transformed into a considerably smaller instance

$$\psi : (r, r \cdot s + e) \mapsto (r \bmod f_1, (r \bmod f_1) \cdot (s \bmod f_1) + e \bmod f_1). \tag{8.6}$$

For simplicity, we write $\hat{r} = r \bmod f_1$, $\hat{s} = s \bmod f_1$ and $\hat{e} = e \bmod f_1$. As before, we may write $\hat{r} = \sum_{i=0}^{l-1} \hat{r}_i x^i$, etc.

The new instance has a smaller dimension, as $\deg f_1 < \deg f$. However, the distribution of the errors is also changed. The error distribution in the larger ring is $\mathsf{Ber}_{\frac{1}{2}(1-\epsilon)}$, but in the smaller ring each noise variable ($\hat{e} = e \bmod f_1$) is a sum of several entries from e. The number of noise variables that constitutes a new error position ($\hat{e} = e \bmod f_1$) depends entirely on the relation between f and f_1.

The following example f_1 to be one of the irreducible polynomials employed in [HKL+12].

Example 8.1 *Let* $\deg f = 621$, *let* $f_1 = x^{127} + x^8 + x^7 + x^3 + 1$ *and assume* $f_1 | f$. *If we consider* $\hat{e}_i$ *to be the ith entity of the coefficient vector of* $\hat{e} = e \bmod f_1$, *then we express* $\hat{e}_0$ *as a sum of bits from e as follows,*

$$\begin{aligned}\hat{e}_0 = e_0 + {} & e_{127} + e_{246} + e_{247} + e_{251} + e_{254} + e_{365} + \\ & e_{367} + e_{375} + e_{381} + e_{484} + e_{485} + e_{486} + \\ & e_{487} + e_{489} + e_{491} + e_{492} + e_{495} + e_{499} + \\ & e_{500} + e_{501} + e_{502} + e_{505} + e_{508} + e_{603} + e_{607}.\end{aligned}$$

In particular, we are interested in linear relations that have as few noise variables from e involved as possible. We use the piling-up lemma (Lemma 1.3) to determine the new bias in $\hat{e}$ *after summing up a number of error bits. For instance, the above linear relation has weight 26. Hence, by the Piling-up lemma, the bias of* $\hat{e}$ *in that particular position (position 0) is* ϵ^{26}, *i.e.,*

$$\mathbb{P}(\hat{e}_0 = 1) = \frac{1}{2} \cdot (1 - \epsilon^{26}).$$

Throughout this chapter we assume that f_1 is an irreducible polynomial. However, this condition is not a necessity, as the essential feature of the new attack is a CRT map. Actually, it is sufficient if the two polynomials f_1 and f/f_1 are co-prime; for example, we could set the polynomial with small degree to be the product of several irreducible polynomials and obtain a solution as well.

8.2.2 USING LOW-WEIGHT RELATIONS TO BUILD A DISTINGUISHER

We will now show how to build a distinguisher for Ring-LPN with a reducible polynomial using the CRT transformation described in the previous section.

In Example 8.1, we give a linear relation expressing an error variable $\hat{e}_0$ in the smaller ring as a sum of relatively few error variables in the larger ring. In our example, the polynomial f_1 »behaves well«; it is very sparse and yields a low-weight relation expressing a single noise variable $\hat{e}_0$. However, this will generally not be the case: it may be very difficult to find a desired linear relation with few error variables.

We observe an interesting connection between this problem and searching for codewords with minimum distance in a linear code. The CRT transformation can be viewed as

$$\begin{pmatrix} 1 \bmod f_1 \\ x \bmod f_1 \\ \vdots \\ x^{t-1} \bmod f_1 \end{pmatrix} = \underbrace{\left(\begin{array}{c} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{l-1} \\ \hline \mathbf{g}_l \\ \mathbf{g}_{l+1} \\ \vdots \\ \mathbf{g}_{t-1} \end{array}\right)}_{=\mathbf{G}_\psi^{\mathrm{T}}} \begin{pmatrix} 1 \\ x \\ \vdots \\ x^{l-1} \end{pmatrix}, \tag{8.7}$$

where the top part of $\mathbf{G}_\psi^{\mathrm{T}}$ is an identity matrix and each row $\mathbf{g}_i$, for $0 \leqslant i \leqslant t-1$, is the coefficient vector of the polynomial $x^i \bmod f_1$.

Thereby, expressing the error polynomial in the smaller ring

$$\begin{aligned} e \bmod f_1 &= \underbrace{\begin{pmatrix} e_0 & e_1 & \cdots e_{t-1} \end{pmatrix}}_{\stackrel{\text{def}}{=}\mathbf{e}^{\mathrm{T}}} \mathbf{G}_\psi^{\mathrm{T}} \begin{pmatrix} 1 \\ x \\ \vdots \\ x^{l-1} \end{pmatrix} \\ &= \underbrace{\begin{pmatrix} \hat{e}_0 & \hat{e}_1 & \cdots \hat{e}_{l-1} \end{pmatrix}}_{\stackrel{\text{def}}{=}\hat{\mathbf{e}}^{\mathrm{T}}} \begin{pmatrix} 1 \\ x \\ \vdots \\ x^{l-1} \end{pmatrix}, \end{aligned} \tag{8.8}$$

we obtain

$$\mathbf{e}^{\mathrm{T}}\mathbf{G}_\psi^{\mathrm{T}} = \hat{\mathbf{e}}^{\mathrm{T}} \Rightarrow \hat{\mathbf{e}} = \mathbf{G}_\psi \mathbf{e}. \tag{8.9}$$

Let d_ψ be the minimum distance of the linear code generated by $\mathbf{G}_\psi$. Then by definition, there exists at least one vector $\mathbf{m}$ such that the product $\mathbf{m}\mathbf{G}_\psi$ has Hamming weight exactly d_ψ. More specifically,

$$\langle \hat{\mathbf{e}}, \mathbf{m} \rangle = \left\langle \mathbf{e}, \mathbf{m}\mathbf{G}_\psi \right\rangle, \tag{8.10}$$

where $\langle \mathbf{e}, \mathbf{m}\mathbf{G}_\psi \rangle$ is a sum of d_ψ noise variables. Thus, according to the Piling-up lemma we obtain the following proposition.

Proposition 8.4 (Lower bound on new bias) If the minimum distance of the linear code $\mathbf{G}_\psi$ is d_ψ, then the largest bias of some linear combination of noise variables in the smaller ring is no less than ϵ^{d_ψ}.

Consequently, in order to determine the security related to a certain polynomial, we need to determine the minimum distance of the code generated by $\mathbf{G}_\psi$. By applying an information-set decoding algorithm, we can find the minimum distance.

In the example of Lapin, applying ISD algorithms is not necessary since the polynomials f_i, $1 \leqslant i \leqslant 5$ are very sparse and admit a very low row-weight of the generator matrix, well below the Gilbert-Varshamov bound (which is around 154).

8.2.3 RECOVERING THE SECRET POLYNOMIAL

After determining the strongest bias ($\langle \hat{\mathbf{e}}, \mathbf{m} \rangle$), we move to the recovery part. We divide this part into three different steps

1. **Transforming to** LPN: Recall that $\deg f_1 = l$. We ask the oracle $\Pi^{\epsilon}_{\text{Ring-LPN}}$ for N samples $(\hat{r}_{(i)}, \hat{s} \cdot \hat{r}_{(i)} + \hat{e}_{(i)})$ and then convert each of them to l standard LPN samples by the mapping τ defined in Section 8.1.1. Write these samples in the matrix form $(\hat{\mathbf{A}}_\mathbf{i}, \hat{\mathbf{A}}_\mathbf{i}\hat{\mathbf{s}} + \hat{\mathbf{e}}_i)$. Then, by multiplication with the vector $\mathbf{m}$, we construct a new LPN sample,

$$(\mathbf{m}\hat{\mathbf{A}}_\mathbf{i}, \mathbf{m}\hat{\mathbf{A}}_\mathbf{i}\hat{\mathbf{s}} + \langle \hat{\mathbf{e}}_i, \mathbf{m} \rangle), \tag{8.11}$$

   from each Ring-LPN sample. According to Proposition 8.4, the created samples are with large bias, i.e., no less than ϵ^{d_ψ}.

   The overall computational complexity of this step is bounded by,

$$C_1 = l^2 \cdot N + (l+1) \cdot l \cdot N = l \cdot N \cdot (2l+1). \tag{8.12}$$

2. **Merging samples**: Put the new samples in a list $\mathcal{L}$ indexed by its last k entries of the vector $\mathbf{m}\hat{\mathbf{A}}_\mathbf{i}$. Merging the list $\mathcal{L}$ by itself yields a list $\mathcal{L}' = \mathcal{L} \diamond \mathcal{L}$ with elements $\mathbf{m}\hat{\mathbf{A}}_\mathbf{i}$ and $\mathbf{m}\hat{\mathbf{A}}_\mathbf{j}$ (denoted $\hat{\hat{\mathbf{r}}}_i$ and $\hat{\hat{\mathbf{r}}}_j$, respectively),

$$(\hat{\hat{\mathbf{r}}}_i, \langle \hat{\hat{\mathbf{r}}}_i, \hat{\mathbf{s}} \rangle + \langle \hat{\mathbf{e}}_i, \mathbf{m} \rangle) + (\hat{\hat{\mathbf{r}}}_j, \langle \hat{\hat{\mathbf{r}}}_j, \hat{\mathbf{s}} \rangle + \langle \hat{\mathbf{e}}_j, \mathbf{m} \rangle) = (\hat{\mathbf{r}}', \langle \hat{\mathbf{r}}', \hat{\mathbf{s}} \rangle + \hat{e}'), \tag{8.13}$$

   yields a vector $\hat{\mathbf{r}}'$ that has at most $l - k$ non-zero positions. The number of such samples is approximately $M = N^2/2^k$. The new samples, such

as $(\hat{\mathbf{r}}', \hat{v}')$, depend only on $l-k$ coefficients of the secret $\hat{s}$ and has a bias that is

$$\epsilon' = \epsilon^{2d_\psi}. \tag{8.14}$$

Calculating the relative entropy between the distribution of error in $(\hat{\mathbf{r}}', \hat{v}')$ and the uniform distribution, we find from (1.40) that the number of samples required is

$$M \geqslant 2\ln 2 \cdot \frac{-(l-k)\cdot \log\theta}{\epsilon^{4d_\psi}}, \tag{8.15}$$

for some error probability θ. For instance, we may set $\theta = \frac{1}{4}$, then we obtain

$$M \geqslant \frac{4\ln 2 \cdot (l-k)}{\epsilon^{4d_\psi}}. \tag{8.16}$$

Storing the N LPN samples uses $l \cdot N$ bit operations, and performing the birthday procedure requires

$$l \cdot \frac{N^2}{2^k} \tag{8.17}$$

bit operations. Thus, the total complexity of this step is,

$$C_2 = l \cdot N \cdot \left(1 + \frac{N}{2^k}\right). \tag{8.18}$$

Thus, at this point, we have generated M vector samples $(\hat{\mathbf{r}}'_i, \hat{v}_i)$. All $\hat{\mathbf{r}}'_i$ vectors have dimension no more than $l-k$ as we cancelled out k bits of $\hat{\mathbf{r}}'_i$. Hence, it is enough to consider only $l-k$ bits of $\hat{s}$, i.e., we assume that $\hat{s}$ is of dimension $l-k$. We are then prepared for the final step.

3. **Distinguishing the best candidate**: Group the samples $(\hat{\mathbf{r}}'_i, \hat{v}_i)$ in sets $L(\hat{\mathbf{r}}'_i)$ according to $\hat{\mathbf{r}}'_i$ and then define the function $f_L(\hat{\mathbf{r}}'_i)$ as

$$f_L(\hat{\mathbf{r}}'_i) = \sum_{(\hat{\mathbf{r}}'_i, \hat{v}_i) \in L(\hat{\mathbf{r}}'_i)} (-1)^{\hat{v}_i}. \tag{8.19}$$

The Walsh transform of f_L is defined as

$$F(\hat{\mathbf{s}}) = \sum_{\hat{\mathbf{r}}'_i} f_L(\hat{\mathbf{r}}'_i)(-1)^{\langle \hat{\mathbf{s}}, \hat{\mathbf{r}}'_i \rangle}. \tag{8.20}$$

Here we exhaust all the 2^{l-k} candidates of $\mathbf{s}$ by computing the Walsh transform.

Algorithm 14 (Partial recovery of Ring-LPN)

Input: Algorithm parameter $N \in \mathbb{N}$.

Output: Secret vector $\hat{\mathbf{s}}_{\text{opt}}$.

1 (**Pre-processing**) Determine the minimum weight d_ψ of the linear code generated by the CRT transformation and find its corresponding linear relation $\mathbf{m}$;
2 Ask the oracle $\Pi^{\epsilon}_{\text{Ring-LPN}}$ for N samples, and then transform each of them to a standard LPN sample with the largest bias, which is no less than ϵ^{d_ψ};
3 Use the birthday technique to reduce the secret length at the cost of decreasing the bias;
4 Perform fast Walsh-Hadamard transform on the remaining $l-k$ bits of $\hat{\mathbf{s}}$;
5 Output the $\hat{\mathbf{s}}_{\text{opt}}$ that maximizes the absolute value of the transform;

Given the candidate $\hat{\mathbf{s}}$, $F(\hat{\mathbf{s}})$ is the difference between the number of predicted 0:s and the number of predicted 1:s for the bit $\hat{v}'_i + \langle \hat{\mathbf{s}}, \hat{\mathbf{r}}'_i \rangle$. If $\hat{\mathbf{s}}$ is the correct guess, then it is distributed according to $\text{Ber}_{\frac{1}{2}(1-\epsilon^{2d_\psi})}$; otherwise, it is considered random. Thus, the best candidate $\hat{\mathbf{s}}_{\text{opt}}$ is the one that maximizes the absolute value of $F(\hat{\mathbf{s}})$, i.e.,

$$\hat{\mathbf{s}}_{\text{opt}} = \arg\max_{\hat{\mathbf{s}} \in F_2^{l-k}} |F(\hat{\mathbf{s}})|, \tag{8.21}$$

and we need approximately ϵ^{4d_ψ} samples to distinguish these two cases. Note that false positives are quickly detected in an additional step and this does not significantly increase complexity.

We employ fast Walsh-Hadamard transform technique to accelerate the distinguishing step. For well-chosen parameters, the complexity is approximately,

$$C_3 = (l-k) \cdot 2^{l-k}. \tag{8.22}$$

From the new algorithm, there are some important consequences to consider when choosing parameters to thwart our attack. We give some very brief comments, assuming that every smaller ring is of approximately the same size:

- Choosing a large number of factors in f seems a bit dangerous, as the dimension of the code $\mathbf{G}_\psi$ becomes small. In our attack we used a birthday argument to reduce the dimension of $\hat{s}$, but this might not be

Protocol 1 (Lapin Authentication)

Tag		**Reader**
		$c \xleftarrow{\$} \{0,1\}^\lambda$;
	$\xleftarrow{c}$	
$r \xleftarrow{\$} R^*; e \xleftarrow{\$} \mathsf{Ber}_\eta^R$;		
$z \leftarrow r \cdot (s \cdot \pi(c) + s') + e$;		
	$\xrightarrow{(r,z)}$	
		if $r \notin R^*$ **then** reject;
		$e' \leftarrow z - r \cdot (s \cdot \pi(c) + s')$;
		if $w_{\mathrm{H}}(e') > n \cdot \eta'$ **then** reject
		else accept;

necessary if the dimension is already very low. Instead we may search for special $\hat{r}$ (e.g., many $\hat{r} = 1$) values that allow quick recovery of $\hat{s}$.

- One should use irreducible polynomials f_i with degree around $\frac{n}{m}$ for $1 \leqslant i \leqslant m$ such that for every f_i the corresponding linear code $\mathbf{G}_\psi$ has minimum distance as large as possible. From the Gilbert-Varshamov bound, we roughly know what to expect. However, the GV bound may not be asymptotically true for codes generated by the CRT transform.

- Following this line, a necessary but probably insufficient condition on ϵ is that
$$\frac{4 \ln 2 \cdot n}{m \cdot \epsilon^{4d_\psi}} \geqslant 2^b \tag{8.23}$$
for b-bit security[5].

8.3 LAPIN AUTHENTICATION PROTOCOL

In this section, we describe the Lapin two-round authentication protocol. Protocol 1 describes the procedure used by Lapin, in which information is exchanged between the tag and the reader.

Let $R = \mathbb{F}_2[x]/(f)$ be a ring and R^* the set of units in R. The protocol is defined over the ring R. Let π be a mapping, chosen such that for all $c, c' \in \{0,1\}^\lambda$, $\pi(c) - \pi(c') \in R \backslash R^*$ if and only if $c = c'$. Furthermore, let

[5]This is just one concern as there may be many other aspects to be taken into consideration that will make the problem solvable. For instance, if d_ψ and ϵ are very large the constraint can be satisfied while the problem in a larger ring remains easy.

$\eta \in (0, \frac{1}{2})$ be a Bernoulli distribution parameter and $\eta' \in (\eta, \frac{1}{2})$ a threshold parameter. The elements R, $\pi : \{0,1\}^\lambda \to R$, η and η' are public parameters. The ring elements $s, s' \in R$ constitute the secret key. Suppose that we have a key-generation oracle; it will give us the key consisting of two ring elements $s, s' \xleftarrow{\$} R$. The secret key is shared among the tag and the reader.

8.4 THE IMPROVED VERSION FOR THE PROPOSED INSTANCE OF LAPIN

In [HKL+12], Heyse *et al.* employ the following reducible polynomial,

$$\begin{aligned} f = \; & \underbrace{\left(x^{127} + x^8 + x^7 + x^3 + 1\right)}_{f_1} \cdot \underbrace{\left(x^{126} + x^9 + x^6 + x^5 + 1\right)}_{f_2} \\ & \cdot \underbrace{\left(x^{125} + x^9 + x^7 + x^4 + 1\right)}_{f_3} \cdot \underbrace{\left(x^{122} + x^7 + x^4 + x^3 + 1\right)}_{f_4} \\ & \cdot \underbrace{\left(x^{121} + x^8 + x^5 + x^1 + 1\right)}_{f_5}, \end{aligned} \tag{8.24}$$

as the underlying structure of the quotient ring R. This parameter setting is then adopted by Gaspar *et al.* in a more recent paper [GLS14], to show that the original protocol and its hardware variant, Mask-Lapin, have much gain, compared with the implementation from block ciphers (e.g., AES), in the sense of resisting power analysis attacks by masking.

However, in the sense of thwarting the new attack we present, it is not a good selection. We give two reasons as follows.

- As previously stated, for any polynomial f_i ($1 \leqslant i \leqslant 5$), the generator matrix $\mathbf{G}_\psi$ of the corresponding code is sparse, hence yielding that we could roughly adopt one row vector in $\mathbf{G}_\psi$ as its minimum-weight codeword. Then, the vector $\mathbf{m}$ is of Hamming weight 1 and we could save the computational cost for linearly combining several samples to form a new one with the largest bias.

- Secondly, for the Lapin instance, the largest bias always holds at the last row of the generator matrix $\mathbf{G}_\psi$, when modulo operation is taken over each irreducible factor. Furthermore, for Ring-LPN samples $(\hat{r}_{(i)}, \hat{s} \cdot \hat{r}_{(i)} + \hat{e}_{(i)})$, if we find collisions on the last k positions of $\hat{r}_{(i)}$ (k is larger than 10), then the last row vector in the matrix form of the merged sample $(\hat{r}', \hat{r}' \cdot \hat{s} + \hat{e}')$ is of the form

$$\begin{pmatrix} 0 & \cdots & 0 & \hat{r}'_{l-k-1} & \hat{r}'_{l-k-2} & \cdots & \hat{r}'_0 \end{pmatrix}. \tag{8.25}$$

Algorithm 15 (Improved partial key recovery for Lapin)

Input: Algorithm parameters $N, k, w \in \mathbb{N}$.

Output: Secret vector $\hat{\mathbf{s}}_{\text{opt}}$.

1 (**Pre-processing**) Find the weight w of the $(l-1)$th row in the generator matrix of the CRT transformation;
2 Ask the oracle $\Pi^{\epsilon}_{\text{Ring-LPN}}$ for N samples, index them by the last k coefficients of $\hat{r}$, and then search for all collisions

$$(\hat{r}', \hat{r}' \cdot \hat{s} + \hat{e}_1 + \hat{e}_2) = (\hat{r}_1, \hat{s} \cdot \hat{r}_1 + \hat{e}_1) + (\hat{r}_2, \hat{s} \cdot \hat{r}_2 + \hat{e}_2),$$

where the last k coefficients of $\hat{r}_1$ and $\hat{r}_2$ are the same;
3 Generate standard LPN samples from the Ring-LPN samples, and then perform fast Walsh-Hadamard transform on $l-k$ bits of $\hat{\mathbf{s}}$;
4 Output the $\hat{\mathbf{s}}_{\text{opt}}$ that maximizes the absolute value of the transform;

This vector can be read from the polynomial $\hat{r}'$ directly, without any computation cost.

Therefore, we could present a specific algorithm for solving the Lapin instance, see Algorithm 15, which is more efficient than the generic one. After determining the weight w of the last row vector in the generator matrix $\mathbf{G}_{\psi}$, we ask the oracle $\Pi^{\epsilon}_{\text{Ring-LPN}}$ for N samples $(\hat{r}_{(i)}, \hat{s} \cdot \hat{r}_{(i)} + \hat{e}_{(i)})$ and search for collisions by the last k coefficients of $\hat{r}$ directly. Then, for each collision represented by a merged Ring-LPN sample $(\hat{r}', \hat{v}')$, we construct a new standard LPN sample, where the vector is generated by (8.25), and the observed value is the coefficient of x^{l-1} in the polynomial $\hat{v}'$. These samples are with the largest bias. The distinguishing step is the same as that in the generic algorithm and we present the detailed complexity analysis and numerical results in the consecutive sections.

8.5 COMPLEXITY ANALYSIS

Having introduced the two versions in detail, we now estimate their complexity. It is straightforward that the generic algorithm (Algorithm 14) costs

$$C = C_1 + C_2 + C_3 \tag{8.26}$$

bit operations. However, the best complexity is obtained when using Algorithm 15 (which is tailored for the proposed instance in Lapin). We state its

complexity in the following theorem.

Theorem 8.5 (The complexity of Algorithm 15) Let w be the weight of the last row in the CRT transformation matrix $\mathbf{G}_\psi$. Then, the complexity of Algorithm, denoted C^*, is given by

$$C^* = l \cdot \left(N + \frac{N^2}{2^k} \right) + (l-k) \cdot 2^{l-k}, \tag{8.27}$$

under the condition that

$$\frac{N^2}{2^k} \geqslant \frac{4 \ln 2 \cdot (l-k)}{\epsilon^{4w}}. \tag{8.28}$$

Proof. We analyze step by step:

1. First, we store the samples received from N oracle calls into a table using $l \cdot N$ bit operations.

2. Clearing k bits in a collision procedure yields $N^2/2^k$ samples and can be performed in $l \cdot N^2/2^k$ bit operations. As the required standard LPN instance can be read directly from the Ring-LPN instance, the transformation has no computational cost.

3. Afterwards, we perform a fast Walsh-Hadamard transform. If the number of unknown bits $l-k$ is at least $\log_2(N^2/2^k)$, then the complexity is $(l-k) \cdot 2^{l-k}$. This is the final step.

Summarizing the individual steps yields C^* which finalizes the proof. ■

8.6 RESULTS

We now present numerical results of the improved partial key recovery attack on the authentication protocol Lapin. The attack we describe concerns the instance of Lapin using the degree 621 polynomial f and with the parameter $\eta = \frac{1}{6}$. As claimed in [HKL+12], this given instance is designed to resist the best known attack on Ring-LPN within the complexity 2^{80}. However, we have shown that it is possible to greatly improve the attack complexity. To compute the complexity, we set the number of samples to

$$N = \sqrt{\frac{4 \ln 2 \cdot (l-k) \cdot 2^k}{\epsilon^{4w}}} \tag{8.29}$$

and minimize (8.27) over l. The improvements can be seen in Table 8.1.

For the distinguishing attack, the complexity is only $2^{75.02}$. For actual recovery of the secret polynomial, we need all five coordinates in the CRT representation, which gives an upper bound on the security that is roughly $2^{77.66}$. A comparison with previous algorithms is given in Table 8.2.

Polynomial f_i	Parameters			Complexity ($\log_2$)
	k	w	$\log_2 N$	
$x^{127} + x^8 + x^7 + x^3 + 1$	62	26	65.16	75.53
$x^{126} + x^9 + x^6 + x^5 + 1$	61	26	64.66	75.48
$x^{125} + x^9 + x^7 + x^4 + 1$	61	26	64.65	75.41
$x^{122} + x^7 + x^4 + x^3 + 1$	63	27	65.59	75.02
$x^{121} + x^8 + x^5 + x^1 + 1$	58	29	64.11	75.24

Table 8.1: The complexity for attacking each modulus of the proposed instance using a reducible polynomial f.

Algorithm	Complexity ($\log_2$)		
	Queries	Time	Memory
Levieil-Fouque [LF06]	82.0	103.4	100.6
Bernstein-Lange [BL13]	79.3	102.9	97.92
Our attack (search)	63	77.66	74.24
Our attack (decision)	62	75.02	72.59

Table 8.2: Comparison of different algorithms for attacking Lapin with reducible polynomial.

8.7 SUMMARY

In this chapter, we have proposed a new generic algorithm to solve the reducible case of Ring-LPN. By exploiting the ring structure further, our new algorithm is much more efficient than previous algorithms, enough to break the claimed 80-bit security for one of the two proposed instances of Lapin.

We have shown that a linear code arising from the CRT transform characterizes the performance of this attack through its minimum distance. This is combined with some standard techniques of using birthday or possibly generalized birthday arguments and efficient recovery through fast Walsh-Hadamard transform.

The low-weight property of the polynomials in the Lapin case makes the problem considerably easier than otherwise and thus makes Lapin susceptible to our attack. Using really low-weight irreducible polynomials such as $x^{127} + x + 1$ can give rise to linear relations with weight as low as 10 or even less. We have not seen that such polynomials have been pointed out as very weak

before.

The description of the new algorithm was influenced by the Lapin case. There are more improvements that can be described in the general case. One such improvement is the use of a generalized birthday technique [Wag02]. This will allow us to consider larger dimensions at the cost of increasing the noise level. We have also noted that the simple bit-oriented samples can be replaced by more complicated vectorial samples, which will give a stronger bias.

9

Concluding Remarks

In this dissertation, some new techniques for cryptanalysis were proposed. The proposed information-set decoding algorithm, based on Stern's algorithm, was shown to improve complexity over previous approaches by generalized birthday techniques, which also enabled us to provide new security bounds for coding-based cryptographic schemes.

We presented a new variant of the McEliece public-key cryptosystem based on convolutional codes to deal with the lack of entropy in the Goppa code construction, by introducing large amounts of parity checks. We presented parameters for (around) 80-bit security. Although we found that the proposed constructions are probably not best choices for McEliece-type public-key cryptography, the idea of using convolutional codes is still attractive and it would be interesting to further investigate similar constructions. A more concrete open problem is to improve decryption complexity in the tail-biting construction, as it currently does not scale well with the security parameter λ.

Furthermore, we introduced a new cryptanalytic technique based on squaring and described two attacks on the QC-MDPC McEliece proposal, which exploit that each block in the underlying code can be written as a polynomial ring of even dimension. Moreover, we provided new recommendations on how to choose the code, so that the squaring attack can be avoided. As noted, the attack applies to the tightly related QC-LDPC proposals [BBC13], so similar recommendations should be issued here as well. Investigating if the attack impacts other areas of cryptanalysis is an interesting research direction.

We presented two algorithms for solving the general case (any weight w) of LWPM and the special case W4PM (weight $w = 4$). Our results for the first algorithm show that we can break 80-bit security barrier of at least one instance used by the TCHo cryptosystem. We also showed that it is possible to successfully use the generalized birthday technique for finding polynomials

of smaller weight (3,4,5) by using a similar multiplicity technique. In addition, we provided some hardness results for the attacks against QC-MDPC McEliece in relation to LWPM.

Finally, we treated the LPN problem and its ring variant Ring-LPN, for which two algorithms were proposed. The first algorithm, which is applicable to any instance of LPN, gives an improvement over previously known algorithms and breaks the 80-bit security of the common $(512, \frac{1}{8})$ instance. The $(512, \frac{1}{8})$ instance is used in many cryptographic constructions for 80-bit security, and therefore should be replaced. It left as an open problem whether this technique can be used to improve the asymptotic complexity of LPN-solving algorithms and information-set decoding. Other algorithms in this thesis could also benefit from using similar techniques, in particular the algorithms proposed in Chapter 6 – here be dragons. The second presented algorithm for Ring-LPN gives an improvement over conjectured security of certain instances of Lapin; in particular, it renders the reducible 80-bit security instance insecure. We suggested necessary (to avoid our attack) but possibly insufficient constraints for reducible polynomials intended for use in Ring-LPN.

At the end of the day, we have made small steps towards exploring the uncharted lands of code-based cryptography. Practical quantum computing still remains in its cradle, but we have certainly an eventful future ahead of us.

References

[AFS05] D. Augot, M. Finiasz, and N. Sendrier, »A Family of Fast Syndrome Based Cryptographic Hash Functions,« in *Proceedings of Mycrypt*, ser. Lecture Notes in Computer Science. Springer-Verlag, 2005, pp. 64–83.

[Ale03] M. Alekhnovich, »More on average case vs approximation complexity,« in *FOCS*. IEEE Computer Society, 2003, pp. 298–307.

[ASE91] N. Alon, J. H. Spencer, and P. Erdős, *The Probabilistic Method*. Wiley, 1991.

[AvzG07] L. E. Aimani and J. von zur Gathen, »Finding low weight polynomial multiples using lattices,« in *Cryptology ePrint Archive, Report 2007/423*, 2007, http://eprint.iacr.org/.

[Bar94] S. Barg, »Some new NP-complete coding problems,« in *Probl. Peredachi Inf.*, vol. 30, 1994, pp. 23–28.

[BBC08] M. Baldi, M. Bodrato, and F. Chiaraluce, »A new analysis of the McEliece cryptosystem based on QC-LDPC codes,« in *In Security and Cryptography for Networks – SCN'2008*, ser. Lecture Notes in Computer Science, vol. 5229. Springer-Verlag, 2008, pp. 246–262.

[BBC13] M. Baldi, M. Bianchi, and F. Chiaraluce, »Optimization of the parity-check matrix density in QC-LDPC code-based McEliece cryptosystems,« in *IEEE International Conference on Communications Workshops*, June 2013, pp. 707–711.

[BCGM07] M. Baldi, F. Chiaraluce, R. Garello, and F. Mininni, »Quasi-Cyclic Low-Density Parity-Check Codes in the McEliece Cryptosystem,« in *ICC '07. IEEE International Conference on Communications*, June 2007, pp. 951–956.

[Ber05] D. J. Bernstein, »Understanding brute force,« in *Workshop Record of ECRYPT STVL Workshop on Symmetric Key Encryption eSTEAM 2005/036*, 2005.

[Ber09] D. J. Bernstein, »Introduction to post-quantum cryptography,« in *Post-Quantum Cryptography*, D. J. Bernstein, J. Buchmann, and E. Dahmen, Eds. Springer-Verlag, 2009, pp. 1–14.

[BJMM12] A. Becker, A. Joux, A. May, and A. Meurer, »Decoding Random Binary Linear Codes in $2^{n/20}$: How $1 + 1 = 0$ Improves Information Set Decoding,« pp. 520–536, 2012.

[BKW00] A. Blum, A. Kalai, and H. Wasserman, »Noise-Tolerant Learning, the Parity Problem, and the Statistical Query Model,« *Proceedings of STOC 2000*, pp. 435–440, May 2000.

[BKW03] A. Blum, A. Kalai, and H. Wasserman, »Noise-Tolerant Learning, the Parity Problem, and the Statistical Query Model,« *Journal of the ACM*, vol. 50, no. 4, pp. 506–519, July 2003.

[BL13] D. J. Bernstein and T. Lange, »Never Trust a Bunny,« in *Cryptographic Primitives Based on Hard Learning Problems*, ser. Lecture Notes in Computer Science. Springer-Verlag, 2013, pp. 137–148.

[BLP11] D. J. Bernstein, T. Lange, and C. Peters, »Smaller decoding exponents: Ball collision decoding,« in *Advances in Cryptology—CRYPTO 2011*, ser. Lecture Notes in Computer Science, P. Rogway, Ed., vol. 6841. Springer-Verlag, 2011, pp. 743–760.

[BV15] S. Bogos and S. Vaudenay, »On Solving LPN using BKW Variants,« 2015.

[CC98] A. Canteaut and F. Chabaud, »A new algorithm for finding minimum-weight words in a linear code: application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511,« *IEEE Transactions on Information Theory*, vol. 44, pp. 367–378, 1998.

[CFS01] N. Courtois, M. Finiasz, and N. Sendrier, »How to achieve a McEliece-based digital signature scheme,« in *Advances in Cryptology—ASIACRYPT 2001*, ser. Lecture Notes in Computer Science, G. Goos, J. Hartmanis, and J. van Leeuwen, Eds., vol. 2248. Springer-Verlag, 2001, pp. 157–174.

[CHL97] G. Cohen, I. Honkala, and S. Litsyn, *Covering codes*, ser. North-Holland mathematical library. Amsterdam, Lausanne, New York: Elsevier, 1997.

[CJM02] P. Chose, A. Joux, and M. Mitton, »Fast correlation attacks: An algorithmic point of view,« *Lecture Notes in Computer Science*, vol. 2332, pp. 209–221, 2002.

[CT91] T. Cover and J. A. Thomas, *Elements of Information Theory*, ser. Wiley series in Telecommunication. Wiley, 1991.

[CT00] A. Canteaut and M. Trabbia, »Improved fast correlation attacks using parity-check equations of weight 4 and 5,« in *Advances in Cryptology—EUROCRYPT 2000*, ser. Lecture Notes in Computer Science, B. Preneel, Ed., vol. 1807. Springer-Verlag, 2000, pp. 573–588.

[Del78] P. Delsarte, »Bilinear forms over a finite field,« in *Journal of Combinatorial Theory*, vol. 25, 1978, pp. 226–241.

[DH76] W. Diffie and M. E. Hellman, »New directions in cryptography,« *IEEE Transactions on Information Theory*, vol. 22, pp. 644–654, 1976.

[DKPW12] Y. Dodis, E. Kiltz, K. Pietrzak, and D. Wichs, »Message Authentication, Revisited,« in *Advances in Cryptology—EUROCRYPT 2012*, ser. Lecture Notes in Computer Science, D. Pointcheval and T. Johansson, Eds., vol. 7237. Springer-Verlag, 2012, pp. 355–374.

[DLC07] F. Didier and Y. Laigle-Chapuy, »Finding low-weight polynomial multiples using discrete logarithm,« in *International Symposium on Information Theory—ISIT 2007*, A. Goldsmith, A. Shokrollahi, M. Medard, and R. Zamir, Eds., 2007.

[DP12] I. Damgård and S. Park, »Is Public-Key Encryption Based on LPN Practical?« in *Cryptology ePrint Archive, Report 2012/699*, 2012, http://eprint.iacr.org/.

[Dum91] I. Dumer, »The use of information sets in decoding of linear codes,« *In Proceedings of 5th Joint Soviet-Swedish International Workshop on Information Theory*, pp. 50 – 52, 1991.

[DV13] A. Duc and S. Vaudenay, »HELEN: A Public-Key Cryptosystem Based on the LPN and the Decisional Minimal Distance Problems,« in *In proceedings of AFRICACRYPT 2013*, 2013, pp. 107–126.

[EBvT78] R. J. M. E. Berlekamp and H. C. van Tilborg, »On the inherent intractability of certain coding problems,« in *IEEE Transactions on Information Theory*, vol. 24, 1978.

[EOS07] D. Engelbert, R. Overbeck, and A. Schmidt, »A summary of McEliece-type cryptosystems and their security,« 2007.

[Fey82] R. Feynman, »Simulating physics with computers,« *International Journal of Theoretical Physics*, vol. 21, pp. 467 – 488, 1982.

[FOPT10] J. C. Faugére, A. Otmani, L. Perret, and J.-P. Tillich, »Algebraic cryptanalysis of McEliece variants with compact keys,« in *Advances in Cryptology—EUROCRYPT 2010*, ser. Lecture Notes in Computer Science. Springer-Verlag, 2010, pp. 279–298.

[FS09] M. Finiasz and N. Sendrier, »Security Bounds for the Design of Code-Based Cryptosystems,« in *Advances in Cryptology—ASIACRYPT 2009*, ser. Lecture Notes in Computer Science, M. Matsui, Ed., vol. 4586. Springer-Verlag, 2009, pp. 88–105.

[FV06] M. Finiasz and S. Vaudenay, »When stream cipher analysis meets public-key cryptography,« in *Selected Areas in Cryptography*, ser. Lecture Notes in Computer Science, E. Biham and A. M. Youssef, Eds., vol. 4356. Springer-Verlag, 2006, pp. 266–284.

[FV07] M. Finiasz and S. Vaudenay, »TCHo : A hardware-oriented trapdoor cipher,« in *ACISP*, ser. Lecture Notes in Computer Science, J. Pieprzyk, H. Ghodosi, and E. Dawson, Eds., vol. 4586. Springer-Verlag, 2007, pp. 184–199.

[GI01] V. Guruswami and P. Indyk, »Expander-based constructions of efficiently decodable codes,« in *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, October 2001, pp. 658–667.

[GJL14] Q. Guo, T. Johansson, and C. Löndahl, »Solving LPN Using Covering Codes,« in *Advances in Cryptology—ASIACRYPT 2014*, ser. Lecture Notes in Computer Science, P. Sarkar and T. Iwata, Eds., vol. 62. Springer-Verlag, 2014, pp. 1–20.

[GJL15] Q. Guo, T. Johansson, and C. Löndahl, »A New Algorithm for Solving Ring-LPN with a Reducible Polynomial,« 2015, (Submitted).

[GLS14] L. Gaspar, G. Leurent, and F. X. Standaert, »Hardware Implementation and Side-Channel Analysis of Lapin,« in *CT-RSA 2014*, 2014.

[Gol96] J. D. Golić, »Computation of low-weight parity-check polynomials,« *Electronic Letters*, vol. 32, no. 21, pp. 1981–1982, October 1996.

[Gol07] O. Goldreich, *Foundations of Cryptography*. Cambridge University Press, 2007.

[Gri04] R. P. Grimaldi, *Discrete and Combinatorial Mathematics*. Pearson, 2004.

[GRS05] H. Gilbert, M. J. B. Robshaw, and Y. Seurin, »An active attack against hb^+—a provably secure lightweight authentication protocol,« in *Cryptology ePrint Archive, Report 2005/237*, 2005, http://eprint.iacr.org/.

[GRS08a] H. Gilbert, M. J. B. Robshaw, and Y. Seurin, »$HB^\#$: Increasing the Security and the Efficiency of HB^+,« in *Advances in Cryptology—EUROCRYPT 2008*, ser. Lecture Notes in Computer Science, N. P. Smart, Ed., vol. 4965. Springer-Verlag, 2008, pp. 361–378.

[GRS08b] H. Gilbert, M. J. B. Robshaw, and Y. Seurin, »How to encrypt with the lpn problem,« in *In proceedings of ICALP*, ser. Lecture Notes in Computer Science, L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldorsson, A. Ingolfsdottir, and I. Walukiewicz, Eds., vol. 5126. Springer-Verlag, 2008, pp. 679–690.

[HB01] N. J. Hopper and M. Blum, »Secure human identification protocols,« in *Advances in Cryptology—ASIACRYPT 2001*, ser. Lecture Notes in Computer Science, C. Boyd, Ed., vol. 2248. Springer-Verlag, 2001, pp. 52–66.

[Hel80] M. Hellman, »A cryptanalytic time-memory trade-off,« *IEEE Transactions on Information Theory*, vol. IT-26, no. 4, pp. 401–406, July 1980.

[HKL+12] S. Heyse, E. Kiltz, V. Lyubashevsky, C. Paar, and K. Pietrzak, »An Efficient Authentication Protocol Based on Ring-LPN,« in *Fast Software Encryption 2012*, ser. Lecture Notes in Computer Science, vol. 4965. Springer-Verlag, 2012, pp. 346–365.

[HL09] M. Herrmann and G. Leander, »A practical key recovery attack on basic TCHo,« in *Public Key Cryptography – PKC 2009*, ser. Lecture Notes in Computer Science, S. Jarecki and G. Tsudik, Eds., vol. 5443. Springer-Verlag, 2009, pp. 411–424.

[HS13] Y. Hamdaoui and N. Sendrier, »A Non Asymptotic Analysis of Information Set Decoding,« in *Cryptology ePrint Archive, Report 2013/162*, 2013, http://eprint.iacr.org/.

[HvMG13] S. Heyse, I. von Maurich, and T. Güneysu, »Smaller Keys for Code-Based Cryptography: QC-MDPC McEliece Implementations on Embedded Devices,« in *Cryptographic Hardware and Embedded Systems—CHES 2013*, ser. Lecture Notes in Computer Science, G. Bertoni and J. S. Coron, Eds., vol. 8086. Springer-Verlag, 2013, pp. 273–292.

[JL11] T. Johansson and C. Löndahl, »An improvement to Stern's algorithm,« 2011. [Online]. Available: http://lup.lub.lu.se/record/2204753

[JL13] T. Johansson and C. Löndahl, »A new algorithm for finding low-weight polynomial multiples and its application to tcho,« in *Workshop on Coding and Cryptography*, L. Budaghyan, T. Helleseth, and M. G. Parker, Eds., 2013.

[Jou09] A. Joux, *Algorithmic Cryptanalysis*. Chapman & Hall/CRC, 2009.

[JW05] A. Juels and S. A. Weis, »Authenticating pervasive devices with human protocols,« in *Advances in Cryptology—CRYPTO 2005*, ser. Lecture Notes in Computer Science, V. Shoup, Ed., vol. 3621. Springer-Verlag, 2005, pp. 293–308.

[JZ99] R. Johannesson and K. Zigangirov, *Fundamentals of Convolutional Coding*, ser. IEEE Series on Digital and Mobile Communication. IEEE Press, 1999.

[Kir11] P. Kirchner, »Improved Generalized Birthday Attack,« in *Cryptology ePrint Archive, Report 2011/337*, 2011, http://eprint.iacr.org/.

[Knu98] D. E. Knuth, *The art of computer programming*, 3rd ed. Reading, Mass.: Addison-Wesley, 1998, vol. 2: Seminumerical algorithms.

[KPC+11] E. Kiltz, K. Pietrzak, D. Cash, A. Jain, and D. Venturi, »Efficient Authentication from Hard Learning Problems,« in *Advances in Cryptology—EUROCRYPT 2012*, ser. Lecture Notes in Computer Science, K. G. Patterson, Ed., vol. 6632. Springer-Verlag, 2011, pp. 7–26.

[KS06] J. Katz and J. S. Shin, »Parallel and concurrent security of the hb and hb$^+$ protocols,« in *Advances in Cryptology—EUROCRYPT 2006*, ser. Lecture Notes in Computer Science, S. Vaudenay, Ed., vol. 4004. Springer-Verlag, 2006, pp. 73–87.

[LB88] P. J. Lee and E. F. Brickell, »An observation on the security of McEliece's public-key cryptosystem.« in *Advances in Cryptology—EUROCRYPT'89*, ser. Lecture Notes in Computer Science, C. G. Günther, Ed., vol. 330. Springer-Verlag, 1988, pp. 275–280.

[LC04] S. Lin and D. J. Costello, *Error Control Coding, Second Edition*. Prentice-Hall, Inc., 2004.

[LF06] E. Levieil and P.-A. Fouque, »An Improved LPN Algorithm,« in *In Proceedings of Security and Cryptography for Networks*, ser. Lecture Notes in Computer Science, M. Prisco and M. Yung, Eds., vol. 4116. Springer-Verlag, 2006, pp. 348–359.

[LJ12] C. Löndahl and T. Johansson, »A new version of McEliece PKC based on convolutional codes,« in *Information and Communications Security*, T. W. Chim and T. H. Yuen, Eds., vol. 7618, no. 2, 2012, pp. 625–640.

[LJ14] C. Löndahl and T. Johansson, »Improved Algorithms for Finding Low-Weight Polynomial Multiples in $\mathbb{F}_2[x]$ and Some Cryptographic Applications,« in *Designs, Codes, and Cryptography*, vol. 73, no. 2, 2014, pp. 625–640.

[LJS+15] C. Löndahl, T. Johansson, M. K. Shooshtari, M. Ahmadian-Attari, and M. R. Aref, »Squaring Attacks on McEliece Public-Key Cryptosystems Using Quasi-Cyclic Codes of Even Dimension,« 2015, (Submitted).

[LMRS12] M. Lamberger, F. Mendel, V. Rijmen, and K. Simoens, »Memoryless near-collisions via coding theory,« in *Designs, Codes, and Cryptography*, vol. 62, 2012, pp. 1–18.

[LT13a] M. Lamberger and E. Teufl, »Memoryless near-collisions, revisited,« in *Information Processing Letters*, vol. 113, 2013, pp. 60–66.

[LT13b] G. Landais and J.-P. Tillich, »An Efficient Attack of a McEliece Cryptosystem Variant Based on Convolutional Codes,« in *Post-Quantum Cryptography*, ser. Lecture Notes in Computer Science, P. Gaborit, Ed., vol. 7932. Springer-Verlag, 2013, pp. 102–117.

[Man80] Y. Manin, »Vychislimoe i nevychislimoe (computable and non-computable) (in russian),« *Sov. Radio*, pp. 13 – 15, 1980.

[McE78] R. J. McEliece, »A public-key cryptosystem based on algebraic coding theory,« *DSN Progress Report 42–44*, pp. 114–116, 1978.

[Meu12] A. Meurer, »A Coding-Theoretic Approach to Cryptanalysis,« Ph.D. dissertation, Ruhr-Universität Bochum, Faculty of Mathematics, Horst Görtz Institute for IT-Security, 2012.

[MH78] R. Merkle and M. E. Hellman, »Hiding information and signatures in trapdoor knapsacks,« *Information Theory, IEEE Transactions on*, vol. 24, no. 5, pp. 525–530, Sep 1978.

[MMT11] A. May, A. Meurer, and E. Thomae, »Decoding Random Linear Codes in $\tilde{o}(2^{0.054n})$,« in *Advances in Cryptology—ASIACRYPT 2011*, ser. Lecture Notes in Computer Science. Springer-Verlag, 2011.

[Mor98] B. Moret, *The Theory of Computation*. Addison-Wesley, 1998.

[MRS09] C. Monico, J. Rosenthal, and A. Shokrollahi, »Using low density parity check codes in the McEliece cryptosystem,« in *International Symposium on Information Theory—ISIT 2000*, 2009, p. 215.

[MS78] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, 2nd ed. North-holland Publishing Company, 1978.

[MS89] W. Meier and O. Staffelbach, »Fast correlation attacks on certain stream ciphers,« *Journal of Cryptology*, vol. 1, no. 3, pp. 159–176, 1989.

[MTSB13] R. Misoczki, J.-P. Tillich, N. Sendrier, and P. S. L. M. Barreto, »MDPC-McEliece: New McEliece variants from Moderate Density Parity-Check codes,« in *Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on*, July 2013, pp. 2069–2073.

[Nie86] H. Niederreiter, »Knapsack-type crytosystems and algebraic coding theory,« *Problems of Control and Information Theory*, vol. 15, no. 2, pp. 157–166, 1986.

[Pea88] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988.

[Pet11] C. Peters, »Curves, codes, and cryptography,« Ph.D. dissertation, Technische Universitet, Eindhoven, 2011.

[Pra57] E. Prange, »Cyclic Error-Correcting Codes in Two Symbols,« *AFCRC-TN-57-103*, September 1957.

[Pra62] E. Prange, »The use of information sets in decoding cyclic codes,« in *IRE Transactions on Information Theory IT*, 1962, pp. 5–9.

[RSA78] R. Rivest, A. Shamir, and L. Adleman, »A method for obtaining digital signatures and public-key cryptosystems,« *Communications of the ACM*, vol. 21, pp. 120–126, 1978.

[Sel08] A. A. Selçuk, »On Probability of Success in Linear and Differential Cryptanalysis,« *Journal of Cryptology*, vol. 21, no. 1, pp. 131–147, 2008.

[Sen00] N. Sendrier, »Finding the permutation between equivalent linear codes: the support splitting algorithm,« vol. 46, no. 4, Jul 2000, pp. 1193–1203.

[Sen11] N. Sendrier, »Decoding One Out of Many,« in *Post-Quantum Cryptography*, ser. Lecture Notes in Computer Science, B. Y. Yang, Ed., vol. 7071. Springer-Verlag, 2011, pp. 51–67.

[Sha48] C. Shannon, »A mathematical theory of communication,« *Bell System Technical Journal*, vol. 27, pp. 623–656, 1948.

[Sha98] R. L. Shackleford, *Introduction to Computing and Algorithms*. Addison-Wesley, 1998.

[Sho94] P. W. Shor, »Algorithms for quantum computation: Discrete logarithms and factoring,« in *35th Annual Symposium on Foundations of Computer Science, 20-22 November 1994, Santa Fe, New Mexico, USA*. IEEE Press, 1994, pp. 124–134.

[Sma03] N. Smart, *Cryptography: An Introduction*. McGraw-Hill Education, 2003.

[Spi96] D. A. Spielman, »Linear-time encodable and decodable error-correcting codes,« *IEEE Transactions on Information Theory*, vol. 42, no. 6, pp. 1723–1731, November 1996.

[SS92] V. M. Sidelnikov and S. O. Shestakov, »On the insecurity of cryptosystems based on generalized Reed-Solomon codes,« *Discrete Mathematics and Applications 2 (4)*, pp. 439–444, 1992.

[SS13] N. Sendrier and D. E. Simos, »How easy is code equivalence over $\mathbb{F}_q$?« *WCC 2013 - International Workshop on Coding and Cryptography*, April 2013.

[Ste89] J. Stern, »A method for finding codewords of small weight,« in *Coding theory and applications*, ser. Lecture Notes in Computer Science, J. W. G. D. Cohen, Ed. Springer-Verlag, 1989, pp. 106–113.

[Sun98] H. M. Sun, »Improving the security of the McEliece public-key cryptosystem,« in *Advances in Cryptology—ASIACRYPT'98*, ser. Lecture Notes in Computer Science, vol. 1514, 1998, pp. 200–213.

[Vit67] A. J. Viterbi, »Error bounds for convolutional codes and an asymptotically optimum decoding algorithm,« *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, April 1967.

[Wag02] D. Wagner, »A generalized birthday problem,« in *Advances in Cryptology—CRYPTO 2002*, ser. Lecture Notes in Computer Science, M. Yung, Ed., vol. 2442. Springer-Verlag, 2002, pp. 288–303.