Carnegie Mellon University

CARNEGIE INSTITUTE OF TECHNOLOGY

THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF Master of Science

TITLE **Cryptographic Applications of Learning with Errors**

PRESENTED BY **Ameer Mohammed**

ACCEPTED BY THE DEPARTMENT OF

Information Networking Institute

THESIS ADVISOR — DATE

ACADEMIC ADVISOR — DATE

DEPARTMENT HEAD — DATE

APPROVED BY THE COLLEGE COUNCIL

DEAN — DATE

Cryptographic Applications of Learning with Errors

Submitted in partial fulfillment of the requirements for

the degree of

Master of Science

in

Information Security Technology Management

Ameer A. Mohammed

B.S. Computer Engineering, Kuwait University

Carnegie Mellon University
Pittsburgh, PA

April, 2013

Copyright © 2013 by Ameer A. Mohammed
All rights reserved except the rights granted by the
Creative Commons Attribution-Noncommercial Licence

Acknowledgements

I would like to thank my thesis advisor, Avrim Blum, for his invaluable guidance and for giving me this singular opportunity to work in this fascinating sub-domain of cryptography. I would also like to express my sincere gratitude to Virgil Gligor for providing helpful comments and suggestions for corrections regarding the thesis draft. In addition, I would like to give credits to my academic advisor, Nicolas Christin, who supplied the equipment capable of running the computationally-intensive experiments documented in this work, and Tim Vidas who assisted with setting up an account for the simulation machine.

I would like to express my gratitude towards my family and friends for their unyielding support and words of encouragement. Finally, I would like to thank Kuwait University for sponsoring my research and supporting my studies at Carnegie Mellon University.

Abstract

Many practical applications rely on the strength of cryptographic schemes to protect the security of data communication and storage. Several such schemes are based on lattice and Learning with Errors (LWE) problems, which are presumed to be hard to solve in the worst-case. In this work, we explore the relationship between lattice-based problems and LWE, and discuss their applications in modern cryptography including public-key cryptosystems and cryptographic primitives. Furthermore, the Learning Parity with Noise (LPN) problem, a sub-class of the LWE problem, will be more thoroughly studied. We examine and contrast the various algorithms that were designed to solve the LPN problem and suggest improvements to some of them. We analyze the feasibility of each algorithm in terms of performance, query complexity, and memory utilization, and corroborate the analysis with experimental results using different parameters.

Table of Contents

Acknowledgements **ii**

Abstract **iii**

List of Tables **viii**

List of Figures **ix**

1 Introduction **1**

1.1 Notation 2
1.2 Problem Reductions 3
1.3 Classes of Cryptographic Systems 4

2 Lattice-Based Cryptography **6**

2.1 Lattices 6
2.2 Gram-Schmidt Orthogonalization 10
2.3 Lattice Problems 11
2.3.1 Shortest Independent Vector Problem 11
2.3.2 Shortest Vector Problem 12
2.3.3 Unique Shortest Vector Problem 13
2.3.4 Promise SVP 13
2.3.5 Small Integer Solutions 14
2.3.6 Closest Vector Problem 15
2.3.7 Bounded Distance Decoding 15
2.4 Problem Relations 17

2.5 Worst-case and Average-case Hardness . . . 17
2.6 Cryptographic Applications . . . 18
2.6.1 Ajtai-Dwork . . . 18
2.6.2 GGH Cryptosystem . . . 21

3 Learning with Errors **24**

3.1 The General LWE Problem . . . 24
3.2 Hardness Results . . . 25
3.2.1 Classical-Quantum Reduction . . . 25
3.2.2 Full Classical Reduction . . . 27
3.2.3 Search-LWE to Decision-LWE Reduction . . . 29
3.2.4 Worst-case to Average-case Reduction . . . 29
3.3 Known Solutions . . . 30
3.4 Cryptographic Applications . . . 31
3.5 Learning Parity with Noise . . . 32
3.5.1 Pseudorandom Generator . . . 33
3.5.2 HB Protocol . . . 33
3.5.3 The LPN-C Encryption Scheme . . . 36

4 Cryptanalytic Tools **38**

4.1 Brute Force Approaches . . . 38
4.2 Birthday Attacks . . . 39
4.2.1 Birthday-based Algorithms . . . 40
4.2.2 Generalized Birthday Problem . . . 41
4.3 The Walsh-Hadamard Transform . . . 43
4.3.1 Fast Walsh Transform . . . 44
4.3.2 Linear Correlation . . . 45
4.4 Lattice-Based Cryptanalysis . . . 48
4.4.1 Gauss' Algorithm . . . 49
4.4.2 LLL Algorithm . . . 49
4.4.3 Nearest Plane Algorithm . . . 52

5 LPN Algorithms **53**

5.1 Adversary Definition . . . 53
5.2 BKW Algorithm . . . 55
5.2.1 Noise Amplification . . . 55
5.2.2 Sample Definitions . . . 56
5.2.3 Design . . . 57
5.2.4 Analysis . . . 59
5.3 Hash Collision . . . 61
5.3.1 Design . . . 62
5.3.2 Analysis . . . 63
5.4 Fast Correlation Attacks . . . 64
5.4.1 FMICM Algorithm . . . 65
5.4.2 The Cube Root Method . . . 69
5.5 LF2 Algorithm . . . 72
5.5.1 Design . . . 72
5.5.2 Analysis . . . 73
5.6 Random Sampling . . . 74
5.6.1 Design . . . 74
5.6.2 Analysis . . . 75
5.7 Information Set Decoding . . . 76
5.7.1 Computational Syndrome Decoding . . . 77
5.7.2 ISD Algorithm . . . 79
5.7.3 Design . . . 80
5.7.4 Analysis . . . 81
5.7.5 ISD Improvements . . . 83
5.8 Hybrid BKW Algorithm . . . 85
5.8.1 BKW-S Hybrid . . . 86
5.8.2 BKW-H Hybrid . . . 90
5.9 Experimental Tests . . . 90
5.9.1 Setup . . . 91

5.9.2 BKW Parameters 91
5.9.3 Results 92

6 Conclusion **94**

Bibliography **96**

Appendix A Theorems **106**

Appendix B Sub-Procedures **108**

List of Tables

Table 4.1 Truth Table for a 3-input Affine Boolean Function Family . . . 47

Table 5.1 Time Complexity for the Hashing Algorithm using $\eta = 0.45$ . . 64

List of Figures

Figure 2.1 Lattice in $\mathbb{R}^2$. 8
Figure 2.2 Gram Schmidt Orthogonalization 10
Figure 2.3 Approximate and Exact SVP and CVP 12
Figure 2.4 The $\text{GapSVP}_{\zeta,\gamma}$ Problem 14
Figure 2.5 Lattice Problems and Relationships 16

Figure 3.1 Quantum Reduction of LWE 26

Figure 4.1 Generalized Birthday Problem for 8-lists 43

Figure 5.1 Non-linear Pseudo-random Keystream Generation 65
Figure 5.2 Optimum Sample Size for Random Selection Algorithm 77
Figure 5.3 Optimum Sample Size for ISD Algorithm 83
Figure 5.4 Optimum Subweight for ISD Algorithm 84
Figure 5.5 The Fast Walsh Transform for 16-bit Hypothesis 88
Figure 5.6 Experimental Performance of BKW-S for $k = 40$ 89
Figure 5.7 Theoretical Performance of BKW-S for $k = 80$ 89
Figure 5.8 Optimal Parameters for the BKW Algorithm 91
Figure 5.9 Comparative Experimental Results for $k = 32$ 93
Figure 5.10 Comparative Experimental Results for $k = 60$ 93

1

Introduction

The goal of cryptography in information security is to protect private data from being accessed and/or modified by unauthorized users while it is transmitted across an open network or stored within a (possibly remote) storage device. This ensures that data *confidentiality* and *integrity* (and possibly *authenticity*) are satisfactorily maintained. In this work, we will mainly focus on cryptographic applications and schemes that aim to achieve data confidentiality.

There exists various practical methods for implementing cryptography, and all involve transforming the plaintext data into ciphertext (unreadable text) in a process that is known as *encryption*. The operation of recovering the plaintext from ciphertext is known as *decryption* and only authorized users with the right valid key should be able to successfully decrypt a message that is intended to be received by them. One way to evaluate the strength of an encryption system is on how difficult it is to learn one or more bits of the plaintext data or some information that is related to that data. If it is computationally infeasible to deduce any information about the encrypted text without the key, then the encryption protocol is deemed to be secure. Thus, some cryptographic primitives are based on hard mathematical problems such

as integer factorization, discrete logarithm problem, lattice-based problems, learning parity with noise (LPN), and (more generally) learning with errors (LWE). Interestingly, lattices can be used to construct cryptographic functions that are as hard to break as the worst-case instance problem of approximating certain lattice problems, such as the shortest vector (SVP) and closest vector (CVP) problems.

The contribution of this work focuses on exploring possible improvements to some of the existing LPN algorithms as well as finding practical and feasible parameters for these algorithms when solving certain instances of the LPN problem. The analysis of these algorithms will cover implementation-specific details using concrete models, which will be compared against asymptotic performance and theoretical bounds. Comparisons will be made, where necessary, between the expected and actual results to convey the performance of the studied algorithms in terms of the time taken and the number of queries issued.

The outline of this thesis is as follows. In Section 2, the principles and fundamentals of lattice-based cryptography will be presented along with some of the cryptographic schemes that were developed on the assumption of some worst-case hard lattice problems. Section 3 will delve deeper into the LWE problem and its provable hardness assumptions. The LPN problem will also be introduced as one of the more useful hard problems on which several cryptographic primitives and schemes were built. Section 4 will discuss some of the tools used in cryptanalysis, specifically as useful sub-procedures for recovering one or more bits of the key. In Section 5, the new and existing LPN adversary algorithms will be discussed, analyzed, and compared followed by experimental results and discussion.

1.1 Notation

Throughout this work and unless otherwise specified, the following general rules for notation will be used to denote specific entities and/or data structures. Lower-

case italicized Roman symbols (e.g. n) denote scalar values. Bold-faced lower-case symbols, such as $\mathbf{x}$, denote vectors of some given length. A vector $\mathbf{x}$ can also be represented as a string of its constituents like so $(x_1, ..., x_n)$. Bold-faced upper-case symbols, such as $\mathbf{B}$ denote matrices with some arbitrary dimensions. A matrix $\mathbf{B}$ can also be represented as a combination of its column vectors like so $(\mathbf{b}_1, ..., \mathbf{b}_m)$. We will also denote the i^{th} element of some arbitrary vector $\mathbf{x}$ as x_i or $\mathbf{x}_j(i)$, if $\mathbf{x}$ is the j^{th} column vector in matrix $\mathbf{X}$.

We denote the concatenation of two vectors $\mathbf{x}, \mathbf{y}$ as $(\mathbf{x}||\mathbf{y})$. The term $\langle \mathbf{x}, \mathbf{y} \rangle$ indicates the inner product of the equally-sized vectors $\mathbf{x}$ and $\mathbf{y}$.

We say that a function $g(n) = poly(n)$ is a polynomial function if $g(n) = O(n^c)$ for some $c > 0$. We say that a function $g(n) = \text{negl}(n)$ is a negligible function if $g(n) = O(1/poly(n))$. We let $g(n) = \tilde{O}(f(n))$ denote that the function is $g(n) = O(f(n) \log^c n)$ for some $c > 0$. We say that a function $g(n)$ is sub-exponential in n if $g(n) \leq 2^{cn} \; \forall \; c > 0$ or $g(n) = 2^{o(n)}$.

1.2 Problem Reductions

We denote P as the class of problems that can be solved in polynomial time. We denote NP as the class of problems whose solutions can be verified in polynomial time. We will assume that $P \neq NP$ even though the question of whether $P = NP$ is still open. We will often refer to algorithms as *efficient* if they solve their respective problems in polynomial time complexity.

To prove the NP-hardness (or NP-completeness) of a specific problem, we would often have to resort to transforming a previously known hard problem into the new problem that we are investigating so that we can prove the new problem's hardness via a *reduction proof*.

Let A be the problem we are trying to prove to be hard and B be a problem that we already know (from previous proofs and reductions) is hard (e.g. an NP-complete

problem). Thus, if we can transform any instance of B into any instance of A in *polynomial* time, we can say that A is at least as hard as problem B. We will write $B \leq_p A$ to denote the fact that B is polynomially reducible to A. This implies that if there exists an algorithm that efficiently solves A, we will also be able to efficiently solve B. Equivalently, if there is no such efficient algorithm for B, then there is no efficient solution for A (i.e. if B is deemed hard then so is A).

1.3 Classes of Cryptographic Systems

Cryptographic schemes are often classified as either public-key or private-key (a.k.a. symmetric) cryptographic systems. In *public-key* systems, each participating user usually possesses two different, yet mathematically related, keys: a public key PK and a private key SK. The public key is published for everyone to see, while the private key is known only to the user that owns it. When a message is encrypted using PK, only the corresponding private key SK can successfully and correctly decrypt the encrypted message, thus ensuring that only the intended recipient is able to decipher this message. Most public-key schemes allow for encrypting and signing of messages. While the primary goal of encryption is confidentiality, the aim of signing messages is mainly to ensure the authenticity of the transmitted data and ascertain source non-repudiation. An example of such a scheme is RSA.

In *private-key* systems, each participating entity has one secret key SK that is shared with other entities. A message that is encrypted with SK can only be decrypted by any user that has SK. Unlike public-key systems, secure key distribution is often seen as a vital concern that must be addressed to avoid unauthorized acquisition of the key. These systems are further categorized into stream cipher-based and block cipher-based schemes. Stream ciphers (e.g. RC4) encrypt each bit of the message individually using a pseudorandom keystream generator. Block ciphers (e.g. DES and AES) encrypt messages by transforming fixed-size blocks of plaintext into

ciphertext of the same size. Block ciphers are normally used as building blocks for encryption modes such as Chained-Block Cipher (CBC) and Counter (CTR) modes.

2

Lattice-Based Cryptography

Lattice-based cryptographic constructions have increasingly gained popularity during the past few years due to their relatively efficient implementations and strong security foundation that is based around the worst-case hardness of certain lattice problems. While lattices were originally found to be quite potent as tools for cryptanalysis, it was not until 1996 that Ajtai [2] had discovered their usefulness in the creation of cryptographic systems including, but not limited to, cryptographic primitives, such as one-way hash functions, and public-key systems with proven security bounds. In this section, we outline the fundamental concepts of lattice-based cryptography and briefly describe some of the influential cryptographic schemes that were developed by researchers in this area. The background and definitions in this chapter are based on the work in [58, 15]. We direct the reader to these sources for comprehensive proofs and more details.

2.1 Lattices

A *lattice* can be described as a set of coordinates in n-dimensional real space and is generally represented as all possible integer linear combinations of a set of linearly-

independent vectors. More formally, given a set of n-dimensional linearly-independent vectors $\mathbf{B} = [\mathbf{b}_1, ..., \mathbf{b}_m]$, a lattice $\mathcal{L}$ is a discrete additive subgroup of $\mathbb{R}^n$ and is defined as:

$$\mathcal{L}(\mathbf{B}) = \mathcal{L}(\mathbf{b}_1, ..., \mathbf{b}_m) = \left\{ \sum_{i=1}^{m} x_i \mathbf{b}_i : x_i \in \mathbb{Z} \right\} = \{\mathbf{Bx} : \mathbf{x} \in \mathbb{Z}^m\} \tag{2.1}$$

When the basis is represented as the matrix $\mathbf{B}$, the columns of the matrix will indicate the basis vectors:

$$\mathbf{B} = \begin{pmatrix} | & | & & | \\ | & | & & | \\ \mathbf{b}_1 & \mathbf{b}_2 & \cdots & \mathbf{b}_m \\ | & | & & | \\ | & | & & | \end{pmatrix}$$

It follows from the definition of (2.1) that any lattice $\mathcal{L}$ is an additive group (closed under subtraction):

$$\forall\ \mathbf{v}, \mathbf{w} \in \mathcal{L}, \mathbf{v} - \mathbf{w} \in \mathcal{L} \text{ where } \mathbf{v} \neq \mathbf{w}$$

and that any lattice $\mathcal{L}$ is discrete with some fixed minimum distance between any two lattice points (or equivalently, lattice vectors):

$$\exists\ \delta > 0 : \forall\ \mathbf{v}, \mathbf{w} \in \mathcal{L}, \|\mathbf{v} - \mathbf{w}\| \geq \delta \text{ where } \mathbf{v} \neq \mathbf{w}$$

The matrix $\mathbf{B} \in \mathbb{R}^{nxm}$ is called the *basis* of the lattice, and it is not unique to that specific lattice. In fact, one may find two different bases $\mathbf{B}$ and $\mathbf{B'} = \mathbf{BU}$, for any unimodular (i.e. determinant is +1 or -1) matrix $\mathbf{U} \in \mathbb{Z}^{mxm}$, that will yield the same lattice. In other words, we can find two (or more) distinct bases that may result in creating the same set of discrete points on the hyperplane of dimension n. Because of that, we may sometimes omit the basis from the lattice definition and simply say that, for any one lattice $\Lambda = \mathcal{L}(\mathbf{B})$ regardless of the basis used to generate

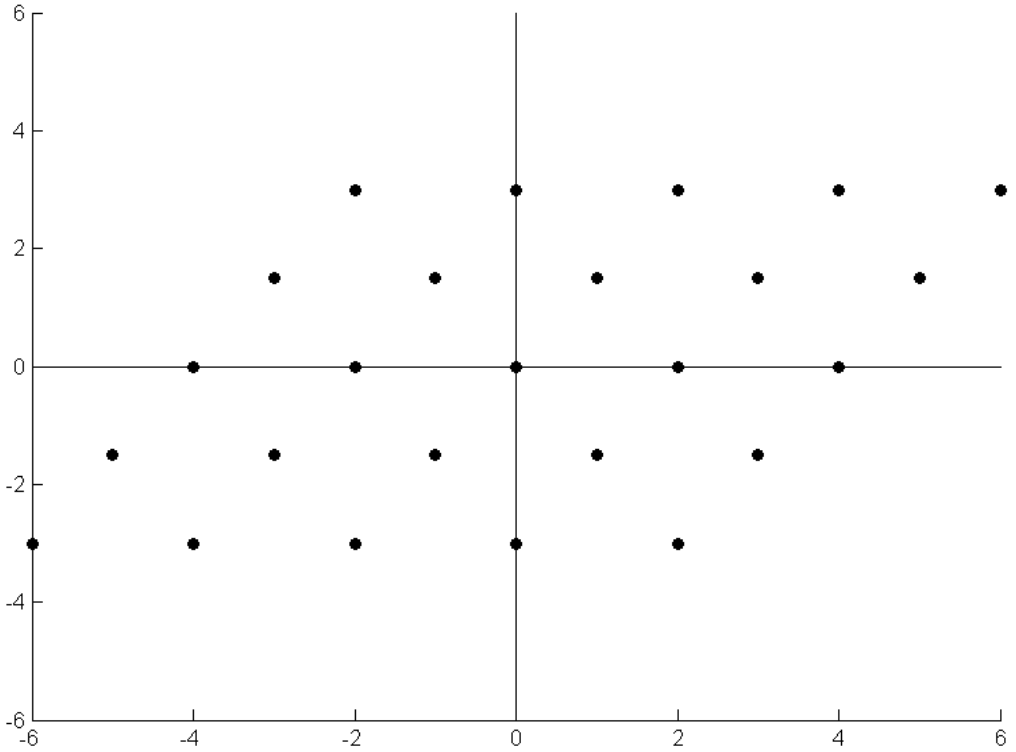

Figure 2.1: An example of a lattice in $\mathbb{R}^2$ with two different possible set of basis vectors

it. The length n of any basis vector $\mathbf{b}_i$ is called the *dimension* of the lattice, and the number of vectors m that make up the basis is called the lattice *rank*. A *full rank* lattice is one where $n = m$. Figure 2.1 shows how a full rank 2-dimensional lattice can be represented using two different bases: $\mathbf{B} = (\mathbf{b}_1, \mathbf{b}_2) = ([1, 0], [0.7, 1.1])$ and $\mathbf{B'} = (\mathbf{b'}_1, \mathbf{b'}_2) = ([-0.4, -2.2], [-0.7, -1.1])$.

The *span* of a lattice is defined as the continuous n-dimensional space occupied by the lattice vectors, and is independent of the basis:

$$\text{span}(\mathbf{B}) = \text{span}(\Lambda) = \{\mathbf{Bx} : \mathbf{x} \in \mathbb{R}^m\} \tag{2.2}$$

The *fundamental domain* of the lattice is defined as the space of points occupied by the parallelepiped that is formed by the basis vectors:

$$\mathcal{P}(\mathbf{B}) = \{\mathbf{Bx} : \forall\ i\ x_i \in [0, 1) \in \mathbb{R}\} \tag{2.3}$$

The width of $\mathcal{P}(\mathbf{B})$ is the minimum Euclidean distance between any one basis vector $\mathbf{b}_i$ and the span of the other basis vectors $\mathbf{b}_j$, $i \neq j$. The shaded regions in Figure 2.1 illustrate $\mathcal{P}(\mathbf{B})$ for the two different bases. A basis **B'** can be considered a basis for a lattice Λ if and only if $\mathcal{P}(\mathbf{B'})$ does not contain any lattice points from Λ other than the origin.

The *determinant* $\det(\Lambda)$ of a lattice is the n-dimensional volume of the parallelepiped $\mathcal{P}(\mathbf{B})$, and it is a lattice invariant that is independent of the basis:

$$\det(\Lambda) = \sqrt{|\mathbf{B}^T\mathbf{B}|} \tag{2.4}$$

We define the *dual* of a lattice Λ as:

$$\Lambda^* = \{\mathbf{y} \in \text{span}(\Lambda) : \langle \mathbf{x}, \mathbf{y} \rangle \in \mathbb{Z}\ \forall\ \mathbf{x} \in \Lambda\} \tag{2.5}$$

For any lattice with basis $\mathbf{B}$, the basis of the dual of the lattice is $\mathbf{B}^* = \mathbf{B}(\mathbf{B}^T\mathbf{B})^{-1}$. If the lattice is full rank, the dual basis is $\mathbf{B}^* = (\mathbf{B}^T)^{-1}$.

Every lattice with rank $m \geq 1$ has a set of m *successive minima* $\boldsymbol{\lambda}(\Lambda) = (\lambda_1, ..., \lambda_m) \in \mathbb{R}^n$ where each component λ_i represents the length of the shortest non-zero vector (between any two lattice points) for rank i with respect to any given norm. One may also think of $\lambda_i(\Lambda)$ as the radius of the smallest sphere centered around the origin containing i linearly-independent lattice vectors in Λ. Normally, most applications are interested in the *first* successive minimum $\lambda_1(\Lambda) = \min_{\mathbf{x} \in \Lambda \setminus 0} ||\mathbf{x}||_2$. We can upper bound the value of $\lambda_1(\Lambda)$ in the l_2 norm using Minkowski's Theorem (see Appendix A), where Λ is n-dimensional and full-rank:

$$\lambda_1(\Lambda) \leq \sqrt{n}\det(\Lambda)^{1/n} \tag{2.6}$$

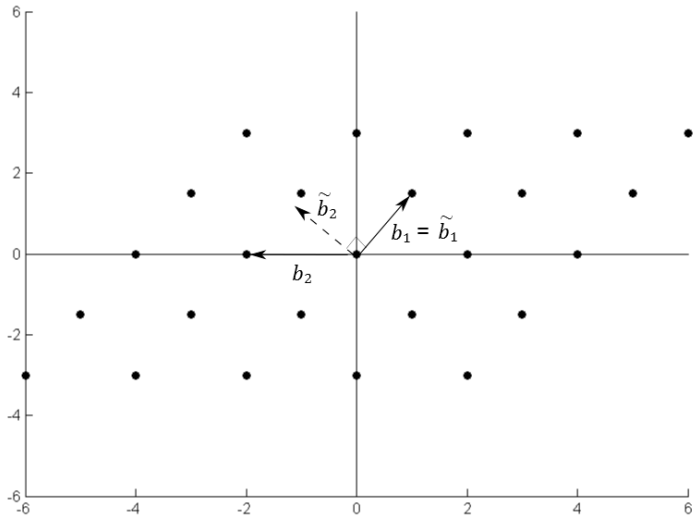

Figure 2.2: The normalized GSO vectors $\tilde{\mathbf{b}}_1$ and $\tilde{\mathbf{b}}_2$ for basis vectors $\mathbf{b}_1$ and $\mathbf{b}_2$

This can also be generalized for the product of higher rank successive minima:

$$\left(\prod_{i=1}^{n} \lambda_i(\Lambda)\right)^{1/n} \leq \sqrt{n}\det(\Lambda)^{1/n} \tag{2.7}$$

2.2 Gram-Schmidt Orthogonalization

Given a lattice Λ with basis $\mathbf{B}$, the *orthogonal* lattice is defined as:

$$\Lambda^{\perp} = \{\mathbf{y} \in \mathbb{R}^n : \langle \mathbf{x}, \mathbf{y} \rangle = 0 \; \forall \; \mathbf{x} \in \Lambda\} \tag{2.8}$$

The *Gram-Schmidt Orthogonalization* (GSO) of n linearly independent vectors $\mathbf{b}_1, ..., \mathbf{b}_n$ is defined as:

$$\tilde{\mathbf{b}}_i = \mathbf{b}_i - \sum_{j=1}^{i-1} \sigma_{i,j} \tilde{\mathbf{b}}_j \; \forall \; 1 \leq i \leq n \tag{2.9}$$

where for any $i \geq 1$ and $j > i$, $\sigma_{i,j} = \frac{\langle \mathbf{b}_i, \tilde{\mathbf{b}}_j \rangle}{\langle \tilde{\mathbf{b}}_j, \tilde{\mathbf{b}}_j \rangle}$. The result of the GSO process is a set of vectors $(\tilde{\mathbf{b}}_1, ..., \tilde{\mathbf{b}}_n)$ where, for all $j > i$, vector $\tilde{\mathbf{b}}_j$ is orthogonal to vector $\tilde{\mathbf{b}}_i$. The

vectors can be made orthonormal by scaling each vector by its size ($\mathbf{u}_i = \tilde{\mathbf{b}}_i / ||\tilde{\mathbf{b}}_i||$) and obtaining a normalized set. Furthermore, if we compute the GSO of a set of vectors that form some lattice $\mathcal{L}(\mathbf{B})$, the resultant vectors $\tilde{\mathbf{b}}_i$ are not guaranteed to be part of the lattice. Figure 2.2 shows an example of a lattice along with its basis vectors and GSO orthonormal vectors.

2.3 Lattice Problems

Generally, algebraic lattice problems are easier to solve than geometric problems. For example, finding the basis of a lattice's dual or checking if two bases generate the same lattice can both be solved in deterministic polynomial time (see Section 2.1), whereas finding a polynomial approximation of the shortest vector in a lattice is known to be NP-hard. Consequently, most of the interesting and potentially useful problems are of the geometric nature, hence motivating their discussion in this section.

2.3.1 Shortest Independent Vector Problem

Given a basis $\mathbf{B}$ for an n-dimensional lattice Λ of rank m, the exact Shortest Independent Vector Problem, SIVP, can be solved by finding a set of m linearly independent vectors $(\mathbf{v}_1, ..., \mathbf{v}_m)$ where $\mathbf{v}_i \in \Lambda$ and $max_i ||\mathbf{v}_i|| \leq \lambda_m(\Lambda)$. The *approximate* version, SIVP_γ, can be solved by finding a set $(\mathbf{v}_1, ..., \mathbf{v}_m) \in \Lambda$ such that:

$$max_i ||\mathbf{v}_i|| \leq \gamma \lambda_n(\Lambda) \tag{2.10}$$

where $\gamma > 1$ is an approximation factor. It was shown in [17] that SIVP is NP-hard for $\gamma = 2^{\log^{1-\epsilon} n}$ for some arbitrary $\epsilon > 0$.

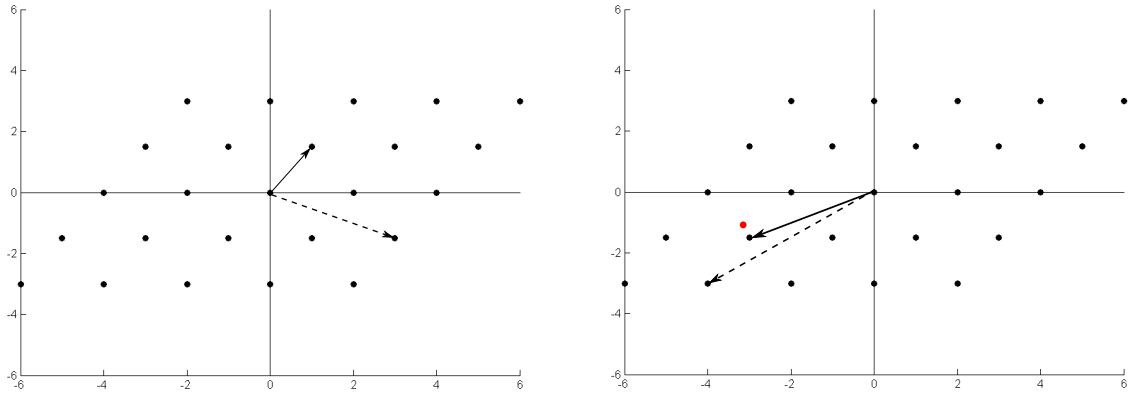

Figure 2.3: The left figure shows one solution for the exact (solid) and approximate (dashed) SVP. The right figure shows one solution for the exact (solid) and approximate (dashed) CVP.

2.3.2 Shortest Vector Problem

Given a basis $\mathbf{B}$ for a lattice Λ, the exact search version of the Shortest Vector Problem, SVP, can be solved by finding a lattice vector $\mathbf{v} \in \Lambda \backslash 0^n$ such that:

$$||\mathbf{v}|| \leq \lambda_1(\Lambda) \tag{2.11}$$

It is possible that there may be multiple linearly independent shortest vectors, but finding only one would suffice as a solution to the problem. It was shown that this version of SVP is NP-hard for the l_∞ norm by [73]. Later on, Ajtai [3] proved this hardness result for the l_2 norm with randomized reductions. All known lattice reduction algorithms (see Section 4.4) solve the problem in time exponential in the dimension n of the lattice. Since this is not always desirable, we are often content with a faster, preferably polynomial time, algorithm that can approximately solve this problem. The *approximate* version of the Short Vector Problem, SVP_γ, can be solved by finding a lattice vector $\mathbf{v} \in \Lambda \backslash 0^n$ such that:

$$||\mathbf{v}|| \leq \gamma \lambda_1(\Lambda) \tag{2.12}$$

where and $\gamma > 1$ is an approximation factor. It was initially shown in [57] that approximating the shortest vector within a constant factor of $\gamma < \sqrt{2}$ is NP-hard, but this hardness result was later improved in [48] to apply for $\gamma = 2^{(\log n)^{1/2-\epsilon}}$ under the l_p norm, for arbitrarily small $\epsilon > 0$ and $p > 1$. An example of a solution for SVP is shown in the left part of Figure 2.3.

2.3.3 Unique Shortest Vector Problem

Given a basis $\mathbf{B}$ for a lattice Λ and a gap factor $\gamma \geq 1$, the Unique Shortest Vector Problem, USVP, can be solved if we find a vector $\mathbf{x} \in \Lambda \backslash 0$ with $||\mathbf{x}|| = \lambda_1(\Lambda)$ (i.e. shortest) such that there exists no other vector $\mathbf{y} \in \Lambda$ where $\mathbf{y}$ is a non-multiple of $\mathbf{x}$ and $||\mathbf{x}|| \leq ||\mathbf{y}|| \leq \gamma||\mathbf{x}||$ (i.e. $\mathbf{x}$ is unique within gap γ). It has been shown in [2] that worst-case USVP can be reduced, in polynomial time, to average-case SVP.

2.3.4 Promise SVP

Given a basis $\mathbf{B}$ for a lattice Λ, a gap factor $\gamma > 1$, and a distance $d > 0$, the decisional promise variant of SVP, GapSVP_γ, will return a 1 (YES) if $\lambda_1(\Lambda) \leq d$, will return a 0 (NO) if $\lambda_1(\Lambda) > \gamma d$, and will return an undefined output otherwise. The best known algorithm [5] that solves GapSVP_γ for $\gamma = poly(n)$ does so in exponential time.

The *general* version of this problem, $\text{GapSVP}_{\zeta,\gamma}$, incorporates additional conditions on the given basis $\mathbf{B}$ by having $\lambda_1(\Lambda) \leq \zeta$ and $1 \leq \gamma d \leq \zeta$ for some $\zeta \geq \gamma \geq 1$. For $\zeta \geq 2^n$, we arrive back to the original GapSVP_γ problem. The conditions for the YES and NO instances of this problem are unchanged. Figure 2.4 shows how the $\text{GapSVP}_{\zeta,\gamma}$ problem can be illustrated as a set of concentric spheres where the radii ζ and γ can be resized to define the problem that needs to be solved. We notice from the figure that the problem will return a YES for lattice Λ and a NO for lattice Λ' provided that we have been promised that λ_1 lies below ζ. This problem can be

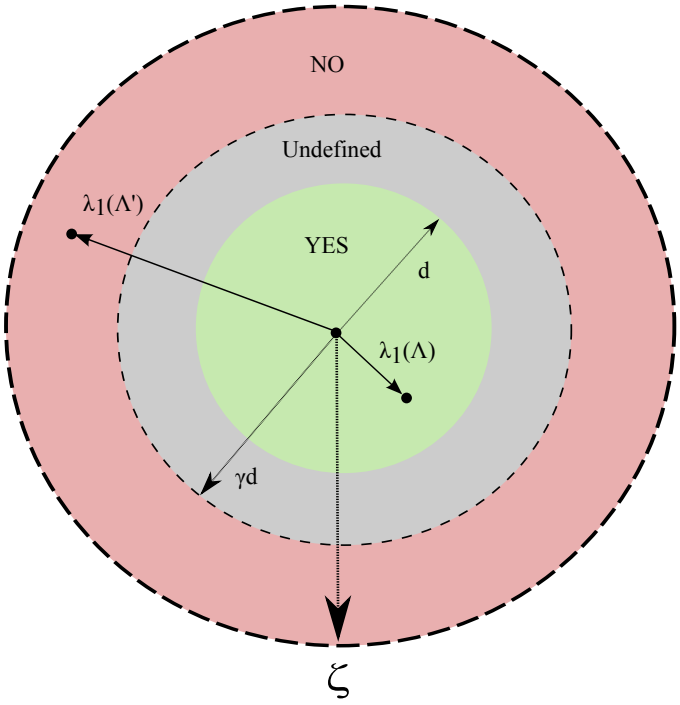

Figure 2.4: The $\text{GapSVP}_{\zeta,\gamma}$ decisional problem solved for two different n-dimensional lattices: Λ and Λ' for some given $d, \gamma,$ and ζ.

used to prove the hardness of LWE in the classical sense (see Section 3.2.2).

2.3.5 Small Integer Solutions

Given a modulus $q \geq 2$, an integer $k \geq n$, a matrix $\mathbf{A} \in \mathbb{Z}_q^{nxk}$ and a threshold real value $v < q$, the Small Integer Solution, SIS, problem can be solved by finding $\mathbf{y} \in \mathbb{Z}_q^k$, a solution to the linear system of equations $\mathbf{Ay} = 0 \bmod q$, subject to the constraint that $||\mathbf{y}|| \leq v$ (i.e. length of $\mathbf{y}$ must be less than v). This problem is very similar to classical SVP. To explain further, we should first consider the following k-dimensional q-ary lattices, originally defined by Ajtai in [2]:

$$\Lambda_q(\mathbf{A}) = \left\{\mathbf{x} \in \mathbb{Z}_q^k : \mathbf{x} = \mathbf{zA} \bmod q \text{ for some } \mathbf{z} \in \mathbb{Z}_q^n\right\} \tag{2.13}$$

$$\Lambda_q^{\perp}(\mathbf{A}) = \left\{\mathbf{x} \in \mathbb{Z}_q^k : \mathbf{Ax} = 0 \bmod q\right\} \tag{2.14}$$

We stress here that $\mathbf{A}$ is not a basis for the aforementioned lattices, but more of a parameter on which to erect the lattices. In fact, we can consider the role of $\mathbf{A}$ in $\Lambda_q(\mathbf{A})$ as a generator for the linear code that is the lattice itself, and as a parity matrix when used in $\Lambda_q^{\perp}(\mathbf{A})$. Also, we note that $\Lambda_q^{\perp}(\mathbf{A})$ is the orthogonal lattice of $\Lambda_q(\mathbf{A})$. Finding a solution to the SIS problem for $\mathbf{A}$ involves finding a "short" vector $\mathbf{y} \in \Lambda_q^{\perp}(\mathbf{A})$ such that $||\mathbf{y}|| \leq v$. It was shown in [59] that the average-case version of SIS is as hard as worst-case SIVP.

2.3.6 Closest Vector Problem

Given a basis $\mathbf{B}$ for a lattice Λ and a target vector $\mathbf{t} \in \mathbb{R}^n$, the exact search version of the Closest Vector Problem (CVP) can be solved by finding a lattice vector $\mathbf{v} \in \Lambda$ such that:

$$||\mathbf{v} - \mathbf{t}|| \leq ||\mathbf{x} - \mathbf{t}|| \; \forall \; \mathbf{x} \in \Lambda \tag{2.15}$$

In other words, one must find a lattice vector that is the closest to the given target vector. The *approximate* version of the Closest Vector Problem, CVP_γ, can be solved by finding a lattice vector $\mathbf{v} \in \Lambda$ such that:

$$||\mathbf{v} - \mathbf{t}|| \leq \gamma ||\mathbf{x} - \mathbf{t}|| \; \forall \; \mathbf{x} \in \Lambda \tag{2.16}$$

where $\gamma > 1$ is an approximation factor. It was shown in [28] that CVP is NP-hard for $\gamma \leq 2^{\log^{1-\epsilon} n}$. Furthermore, based on the work in [42, 38], CVP also appears to be harder than SVP. An example of a solution for CVP is shown in the right part of Figure 2.3.

2.3.7 Bounded Distance Decoding

Given a basis $\mathbf{B}$ for a lattice Λ, a target vector $\mathbf{t} \in \mathbb{R}^n$ that is known to be at most a distance of d from lattice Λ, the Bounded Distance Decoding problem, BDD_d, can

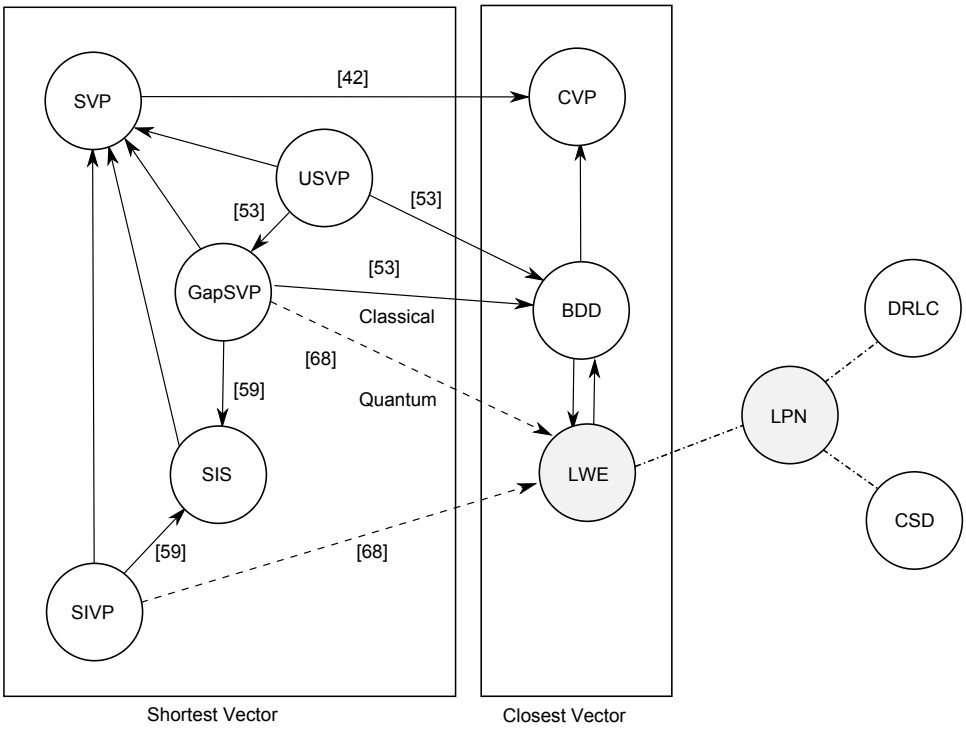

Figure 2.5: The relationships between the different discussed lattice problems. A directional arrow from problem A to problem B implies that A $\leq_p$ B

be solved by finding the exact closest lattice point to $\mathbf{t}$ such that:

$$||\mathbf{v} - \mathbf{t}|| \leq ||\mathbf{x} - \mathbf{t}|| \; \forall \; \mathbf{x} \in \Lambda \tag{2.17}$$

Notice that the only difference between BDD_d and CVP is the fact that in BDD_d we are given additional information about the target vector $\mathbf{t}$, namely the proximity between it and any one of the lattice points. Additionally, if we are given that $d < \lambda_1(\Lambda)/2$ then we are guaranteed a unique solution because there is only one lattice vector that is closest to the target point at that distance. It was shown in [52] that, in the l_p norm, the BDD_d problem is NP-hard for $d \geq \dfrac{\lambda_1(\Lambda)}{\sqrt[p]{2}}$. The work presented in [53] also demonstrates how the BDD problem is as hard as the USVP problem by showing a reduction from USVP to BDD with $d = \lambda_1(\Lambda)/poly(n)$.

2.4 Problem Relations

Inspired by [72], a customized sub-compilation of the problems and their relationships with each other is illustrated in Figure 2.5. Non-trivial relationships are annotated with the appropriate reference for more details on the reduction. The lattice problems can be generally divided into two categories, based on whether the goal is to find shortest vectors in a lattice or to find lattice vectors that are closest to some target vector. While there are many more lattice problems, we present only those that are related to this work. In particular, we will be focusing on the Learning with Errors (LWE) problem and its special case, the Learning Parity with Noise (LPN) problem, both which will be defined in Chapter 3. The LPN problem is also related to other non-lattice problems: the decoding of random linear codes (DRLC) and the computational syndrome decoding (CSD) problem, which is described in more detail in Section 5.7.1.

2.5 Worst-case and Average-case Hardness

It is desirable to have cryptographic schemes be based on problems that have average-case hardness assumptions. An average-case hard problem ensures us that, with high probability, drawing some random instance of the problem from a suitable distribution and using it as part of a cryptographic protocol does not in any way diminish the strength of the procedure. These problems also need to be in NP since we want the decryption process to be polynomially verifiable by the authorized key-holding party. For example, the well-known public-key cryptographic system, RSA, is considered secure assuming that the integer factorization problem is hard *on average*, and the Diffie-Hellman key exchange protocol is considered secure assuming that the Discrete Logarithm problem and the Computational (and Decisional) Diffie-Hellman problem are hard to solve *on average*.

The downside of having cryptographic systems based on average-case hard problems is that we need to make sure that we choose a suitable distribution from which we draw the random instances such that we avoid drawing (with high probability) easily solvable instances. For example, for RSA, choosing only small even numbers as factors for the public key is clearly not a good choice. Thus average-case hardness is necessary, but not sufficient for ensuring security.

On the other hand, if a cryptographic scheme was based on a worst-case hard problem, then breaking this scheme would imply that we can solve *any* instance of the problem. Consequently, we need not worry about how we generate the problem's instance. Several lattice problems have been shown to have provable relationships and reductions between their worst-case and average-case versions. In his seminal paper, Ajtai [2] established a reduction from worst-case USVP to average case approximate SVP. Furthermore, we will present in Section 3.2.4 a result obtained in [65] proving that the worst-case decisional LWE problem can be reduced to its average-case variant, which is considered more useful for cryptographic applications but still as hard. In the next section, we describe a select number of schemes that are built on worst-case lattice problems.

2.6 Cryptographic Applications

2.6.1 Ajtai-Dwork

Though not as efficient or as secure as more recently developed lattice-based systems, the Ajtai-Dwork public-key cryptographic scheme [4] is, nevertheless, one of the first cryptosystems that was built on the basis of worst-case hard lattice problems. In particular, the scheme is based on worst-case USVP (see Section 2.3.3) using a gap factor $\gamma = poly(n)$, and was later improved in [39] to reduce the number of decryption errors.

The scheme is parametrized with n which determines the strength of the system.

Given a message M composed of d bits, the encryption process encrypts each bit $b_i \in \{0,1\}$ of M individually to create ciphertext $\mathbf{C} = (\mathbf{c}_1, ..., \mathbf{c}_d)$ where $\mathbf{c}_i = E(b_i) \in \mathbb{R}^n$.

Definitions

We define the following parameters and distributions that will be used in the public-key cryptographic scheme depicted in Algorithm 2.6.1:

$\mathbf{s} \in \mathbb{R}^n$: The vector that represents the private key.

$m = n^3$: The number of perturbation vectors

$\mathcal{U}_n = \{\mathbf{x} \in \mathbb{R}^n : ||\mathbf{x}|| \leq 1\}$: The n-dimensional sphere of radius 1

$\mathcal{B}_n = \{\mathbf{x} \in \mathbb{R}^n : |x_i| \leq 2^{n \log n} \; \forall \; i \in [1, n]\}$: An n-dimensional cube

$S_n = \{\mathbf{x} \in \mathbb{R}^n : ||\mathbf{x}|| \leq n^{-p}\}$: An n-dimensional sphere of radius n^{-p} where $p \geq 1$. In the original construction, p is set to 8.

$\mathcal{H}(\mathbf{s})$: The distribution of points in $\mathcal{B}_n$ induced by private key $\mathbf{s}$. This random variable outputs a vector $\mathbf{v} = \mathbf{a} + \sum_i^n \mathbf{y}_i$ where $\mathbf{a}$ is drawn uniformly at random from the set $\{\mathbf{x} \in \mathcal{B}_n : \langle \mathbf{x}, \mathbf{s} \rangle \in \mathbb{Z}\}$ and the perturbation vectors $\mathbf{y}_1, ..., \mathbf{y}_n$ are drawn uniformly at random from S_n.

Analysis

The developers of this scheme proved that if one was able to successfully distinguish between the encryption of a 0 and the encryption of a 1, then one can also solve for any instance of USVP, which at first glance seems to be satisfactory for a security guarantee. However, the scheme has been shown by Nguyen and Stern [64] to be insecure against a heuristic attack that allows an adversary to distinguish between encryptions of 0 and 1 and recover the private key using only the corresponding public key. The attack is based on a proof that reduced the problem of distinguishing between encryptions to the problem of approximating, within a polynomial factor,

Algorithm 2.6.1 Ajtai-Dwork's Public Key Cryptosystem

PRIVATE KEY: Select $\mathbf{s}$ from $\mathcal{B}_n$ uniformly at random

PUBLIC KEY: Select vectors $\mathbf{w}_1, ..., \mathbf{w}_n$ and $\mathbf{v}_1, ..., \mathbf{v}_m$ at random from $\mathcal{H}(\mathbf{s})$ subject to the constraint that the width of $\mathcal{P}(\mathbf{w}_1, ..., \mathbf{w}_n) = \mathcal{P}(\mathbf{W})$ is less than $2^{n\log n}/n^2$

ENCRYPTION:

for each bit b_i in M **do**

if $b_i = 0$ **then**

Select $k_1, ..., k_m$ uniformly at random from {0,1}

Select vector $\mathbf{x} \in \mathcal{P}(\mathbf{W})$ such that $\mathbf{x} = \sum_i^m k_i \mathbf{v}_i + \sum_j^n r_j \mathbf{w}_j$ for any $r_j \in \mathbb{Z}$.

else

Select vector $\mathbf{x}$ uniformly at random from $\mathcal{P}(\mathbf{W})$

end if

Set $\mathbf{c}_i = \mathbf{x}$

end for

Output $\mathbf{C} = (c_1, ..., c_d)$

DECRYPTION:

for each ciphertext $\mathbf{c}_i$ in $\mathbf{C}$ **do**

if $(z - \langle \mathbf{c}_i, \mathbf{s} \rangle) \leq 1/n$ for some $z \in \mathbb{Z}$ **then**

Set $b_i = 0$

else

Set $b_i = 1$

end if

end for

Output $M = (b_1, ..., b_d)$

the shortest (or closest) vector for specific lattices, which can be efficiently solved using basis reduction algorithms (see Section 4.4). Furthermore, the size of the public key in this scheme (the number of perturbation vectors) is relatively large, and especially so when the size of the key is increased to counter the private key recovery attack, leading to high memory utilization.

2.6.2 GGH Cryptosystem

The Goldreich-Goldwasser-Halevi (GGH) cryptosystem [40] is constructed via a trapdoor one-way function primitive that is based on the hardness of CVP. It was introduced as a public key cryptographic scheme as well as a digital signature scheme, which makes it suitable substitute for RSA. Unlike the Ajtai-Dwork scheme, GGH makes explicit use of lattices to encrypt and decrypt.

The scheme is parametrized with n which determines the dimension, and subsequently, the strength of the scheme. At a high level, the given plaintext message M will be represented as an n-dimensional vector and encoded as a lattice point $\mathbf{Bm}$ in some lattice with randomly generated basis $\mathbf{B}$. The ciphertext $\mathbf{c} = E(M)$ will be represented as some point that is arbitrarily close to the lattice point $\mathbf{Bm}$. The decryption process involves applying Babai's nearest plane algorithm (see Section 4.4.3) to solve an instance of the approximate closest vector problem using a different private basis $\mathbf{D}$. That is, given the (private) basis and the target vector $\mathbf{c}$, one should find the lattice vector closest to $\mathbf{c}$ to obtain the plaintext message.

Definitions

We define the following parameters that will be used in the public-key cryptographic scheme depicted in Algorithm 2.6.2:

$\mathbf{D} \in \mathbb{Z}^{nxn}$: An integral matrix that represents the private key. It is to be generated randomly as a "good" basis (i.e. basis vectors are as orthogonal as possible) for a

full-rank lattice.

$\mathbf{B} \in \mathbb{Z}^{nxn}$: An integral matrix that represents the public key. It is considered as a "poor" basis (i.e. basis vectors are not very orthogonal) for the full-rank lattice that has basis $\mathbf{D}$.

$\mathbf{m} \in \mathbb{Z}^n$: The message to encrypt encoded as an integral vector.

η : A public real value representing the error

$\mathbf{e} \in \mathbb{Z}^n$: The error vector used in the encryption process

---

Algorithm 2.6.2 GGH Public Key Cryptosystem

PRIVATE KEY: Generate at random basis $\mathbf{D}$ with relatively small integers to build lattice Λ

PUBLIC KEY: Find a non-reduced basis $\mathbf{B}$ such that $\mathbf{B} = \mathbf{DU}$ where $\mathbf{U} \in \mathbb{Z}^{nxn}$ is unimodular. This basis will also build lattice Λ

ENCRYPTION:

Encode M as $\mathbf{m} \in \mathbb{Z}^n$

for each e_i in $\mathbf{e}$ **do**

    Assign e_i a value from $\{-\eta, \eta\}$ uniformly at random

end for

Set $\mathbf{c} = \mathbf{Bm} + \mathbf{e}$

Output $\mathbf{c}$

DECRYPTION:

Using Babai's Algorithm, compute $\mathbf{y} = \mathbf{D}^{-1}\mathbf{c} = \mathbf{D}^{-1}(\mathbf{Bm} + \mathbf{e}) = \mathbf{D}^{-1}\mathbf{Bm}$
Set $\mathbf{m} = \mathbf{B}^{-1}\mathbf{D}(\mathbf{y})$
Output $\mathbf{m}$ ▷ Decryption successful only if $\mathbf{D}^{-1}\mathbf{e} = 0^n$

Analysis

To prevent unauthorized entities from decrypting the ciphertext, the public key $\mathbf{B}$ is generated such that it represents a "poor" basis for lattice $\mathcal{L}(\mathbf{D})$. This way, any entity using the public key as input to the nearest plane algorithm will not obtain

the correct message. Additionally, it is not trivial to obtain a good basis using any of the known lattice basis reduction methods since doing so would imply that it is easy to solve the approximate shortest vector problem. Thus, the decryption task would be only be successful for the entity that holds the private key, the good basis $\mathbf{D}$.

It should also be noted that as we increase the error η, we are making the problem harder since we are widening the distance between the message lattice point $\mathbf{Bm}$ and the ciphertext lattice point $\mathbf{c}$. While that may help in debilitating the adversary's efforts, it may also lead to an increase in decryption errors when the authorized entity tries to decrypt the ciphertext.

Compared to the Ajtai-Dwork scheme, the size of the public key representation is several orders lower. However, Nguyen [63] has demonstrated an attack on this scheme for practical values of $n \leq 350$ by exploiting a vulnerability that allowed for the problem to be reduced to an easier instance of CVP. For an adversary with a known-plaintext attack capability, this exploit presented an opportunity for a successful decryption of a given ciphertext.

3

Learning with Errors

The Learning with Errors (LWE) problem was originally, albeit implicitly, introduced by Ajtai and Dwork [4] with the construction of a public-key cryptosystem based on the unique-SVP lattice problem, which could be reduced to LWE as means of ascertaining the problem's hardness. After being formally defined in 2005 by Regev [68], the LWE problem has since been employed as a flexible and fundamental basis problem for several cryptographic applications. As we will demonstrate in this chapter, the LWE problem is known to be as hard as some worst-case lattice problems, thus ensuring that cryptographic primitives based on the LWE problem are deemed to be sufficiently and provably secure (computationally infeasible to break).

3.1 The General LWE Problem

Given some positive integer $n \geq 1$, a modulus $q \geq 2$, an error (typically Gaussian) distribution χ, and a randomly chosen vector $\mathbf{s} \in \mathbb{Z}_q^n$, we define $\Theta_{\mathbf{s},\chi}$ to be an oracle that, when queried, will supply a tuple of the form $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e \bmod q)$ where $\mathbf{a} \in \mathbb{Z}_q^n$ is a vector chosen uniformly at random and $e \in \mathbb{Z}_q$ is selected from χ. After m

queries to $\Theta_{\mathbf{s},\chi}$, we will have a set of examples encoded as a matrix $\mathbf{A} \in \mathbb{Z}_q^{nxm}$ and a corresponding vector $\mathbf{z} \in \mathbb{Z}_q^m$ of noisy labels, where $\mathbf{z} = \mathbf{A}^T\mathbf{s} + \mathbf{e} \bmod q$ and $\mathbf{e} \in \mathbb{Z}_q^m$ is called the noise vector. We will denote this set of examples and corresponding labels as the oracle set $(\mathbf{A}, \mathbf{z})$.

To solve the search $\text{LWE}_{n,q,\chi}$ problem, one should recover, with high probability $(1 - \text{negl}(n))$, the unknown vector $\mathbf{s} \in \mathbb{Z}_q^n$ given $\mathbf{A}$ and $\mathbf{z}$. To solve the decisional $\text{LWE}_{n,q,\chi}$ problem, one should correctly distinguish between the uniform distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_q$ and the oracle set $(\mathbf{A}, \mathbf{z})$.

3.2 Hardness Results

The hardness results for the LWE problem was first proved by Regev [68] by proposing a quantum reduction from approximating worst-case GapSVP_γ and SIVP to LWE. Later on, Peikert [65] proposed a more stringent, fully classical reduction from the worst-case $\text{GapSVP}_{\zeta,\gamma}$ to LWE. It should be noted, however, that $\text{GapSVP}_{\zeta,\gamma}$ is only equivalent to GapSVP_γ for $\zeta \geq 2^n$, so if one desires to reduce to GapSVP_γ instead, q is required to be exponential in n since, as we shall see in Theorem 3.2.2, q is a proportionally related to ζ.

3.2.1 Classical-Quantum Reduction

Let us define $\alpha \in (0, 1)$, $q \geq 2$, $n > 1$, and $\mu(\Lambda) \leq \lambda_n(\Lambda)\sqrt{\omega(\log n)}$ to be a smoothing parameter. Furthermore, let $N_{\Lambda,r}$ be a discrete Gaussian distribution over lattice Λ with standard deviation r and χ_α be a discrete Gaussian distribution over $\mathbb{Z}_q$ with standard deviation αq.

Theorem 3.2.1 (LWE Quantum Hardness [68])**.** *Given an $\text{LWE}_{n,q,\chi_\alpha}$, a lattice Λ, and poly(n) samples from $N_{\Lambda,r}$ one can approximately solve, using an efficient quantum algorithm, the worst-case GapSVP_γ and SIVP within a factor of $\tilde{O}(n/\alpha)$ given that $\alpha q > 2\sqrt{n}$ and $r \geq \sqrt{2}q\mu(\Lambda)$.*

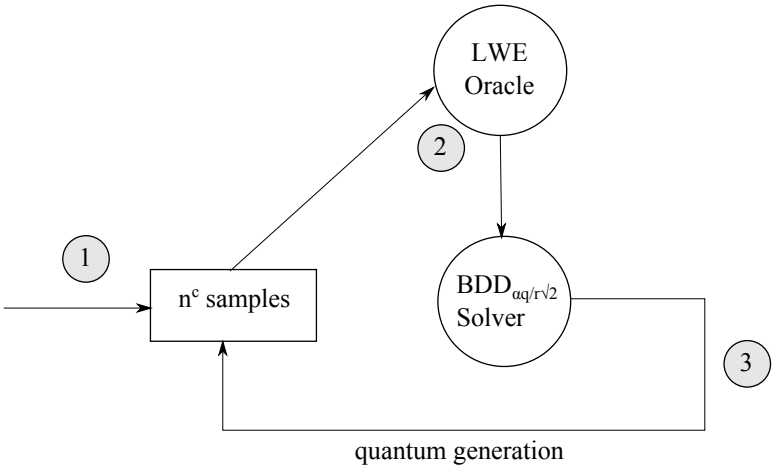

Figure 3.1: Given a LWE oracle, we can construct the BDD solver and quantumly solve the DGS problem

The reduction involves running an iterative procedure where each iteration is composed of two main steps: one classical and the other quantum. The main idea is to use this iterative algorithm to solve the Discrete Gaussian Sampling (DGS) problem, which is proven to be as hard as approximately solving GapSVP_γ and SIVP within some factor (that is GapSVP_γ, SIVP $\leq_p$ DGS). To achieve this, we will construct a Bounded Distance Decoding (BDD) oracle given the LWE oracle to solve the DGS problem.

Discrete Gaussian Sampling

Given an n dimensional lattice Λ and a real $r \geq \sqrt{2n}\mu(\Lambda)/\alpha$, the goal of the DGS problem is to find a sample in $N_{\Lambda,r}$. By having r be close to the lower bound, we can find relatively short vectors, with a norm of around $r\sqrt{n}$. Consequently, this leads to an algorithm that uses DGS to solve SIVP: by calling the DGS algorithm enough number of times we obtain with high probability n short linearly independent vectors.

Classical Part

The input to the iterative algorithm is a real number $r_0 \geq \sqrt{2n}\mu(\Lambda)/\alpha$ and a lattice Λ. We shall express r_i as $r_0(\alpha q/\sqrt{n})^i$ and initialize $i = 3n$. At the start of the procedure, and only for $i = 3n$, we can efficiently generate $poly(n)$ number of samples S_{3n} from $N_{\Lambda,r_{3n}}$ using the LLL algorithm since $r_{3n} > 2^{2n}\lambda_n(\Lambda)$ (Step 1 of Figure 3.1).

For each iteration $i = 3n, 3n-1, ..., 1$, the samples S_i are fed into the LWE oracle, whose output will then be used by the $\text{BDD}_{\alpha q/r\sqrt{2}}$ solver on the dual lattice Λ^* (Step 2 of Figure 3.1). The BDD solver can construct LWE samples of the form ($\mathbf{A}$,$\mathbf{z}$) given the distribution N_{Λ,r_i} and a target vector $\mathbf{t}$ that is within a distance of $\alpha q/(r\sqrt{2})$ from Λ^*. Each vector $\mathbf{a} \in \mathbf{A}$ is a sample from N_{Λ,r_i}, and each label $z \in \mathbf{z}$ is set to be $(\langle \mathbf{a}, \mathbf{t} \rangle \bmod q)$. The samples are submitted to the LWE oracle, which will solve for secret $\mathbf{s} = \mathbf{v} \bmod q$ where $\mathbf{v}$ is the output of the BDD solver (the lattice point closest to $\mathbf{t}$).

Quantum Part

The BDD solver (for the dual lattice Λ^*) is responsible for generating the next batch of samples for $i < 3n$ to be used in the next iteration. The generation process (Step 3) involves constructing a quantum state using the BDD solver than applying a quantum Fourier transform of N_{Λ,r_i} to obtain a sample from the new distribution. We note here that the output of this phase is only one sample. To generate $poly(n)$ samples and set up S_i, we need to execute Step 3 $poly(n)$ number of times per iteration i. After $3n$ iterations, we get the final sample set S_0, which contains the samples we need that solve the DGS problem for r_0.

3.2.2 Full Classical Reduction

Theorem 3.2.2 (LWE Classical Hardness [65])**.** *Let us define* $\alpha \in (0,1)$, $\gamma \geq n/(\alpha\sqrt{\log n})$, $\zeta \geq \gamma$, *and* $q \geq (\zeta/\sqrt{n})\omega(\sqrt{\log n})$. *Given an* LWE_{n,q,χ_α}, *a lattice*

Λ, *and poly*(n) *samples, one can solve, using an efficient algorithm, the worst-case* $GapSVP_{\zeta,\gamma}$.

To achieve this, we first reduce $\text{GapSVP}_{\zeta,\gamma}$ to BBD on lattice Λ, then reduce BDD to LWE using the same notion as Regev's classical part of the iterative procedure (but with classic generation of samples). Given a lattice basis $\mathbf{B}$ such that $\lambda_1(\mathcal{L}(\mathbf{B})) \leq \zeta$ and a number d where $1 \leq \gamma d \leq \zeta$, we perform the following procedure *poly*(n) times:

1. $\text{GapSVP}_{\zeta,\gamma} \leq_p$ BDD: We select a random lattice point $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ and generate the perturbed vector $\mathbf{x} = \mathbf{v} + \mathbf{w} \bmod q$ where $\mathbf{w}$ is drawn uniformly at random from an n-dimensional ball with radius $d\sqrt{n/(4 \log n)}$

2. BDD $\leq_p$ LWE: Call the BDD Solver (the same one used by Regev's classical reduction) with inputs $\mathbf{B}$, $\mathbf{x}$, and $r = q\sqrt{2n}/\gamma d$. To satisfy the preconditions of the BDD Solver, we must select r and γ such that the vector is perturbed by a factor $d' = ||\mathbf{w}|| \leq \alpha q/(\sqrt{2}r)$. The samples from $D_{\Lambda^*,r}$ are generated using a classical method that is described in [34]. The output of the BDD solver will be the lattice vector $\mathbf{y}$ that is closest to $\mathbf{x}$.

3. Feedback to $\text{GapSVP}_{\zeta,\gamma}$: If $\mathbf{y} \neq \mathbf{v}$, output YES (we found another vector that is closer to $\mathbf{x}$ than $\mathbf{v}$).

After all iterations complete, if the procedure never outputs YES, output NO. If a NO is returned, then we know that $\lambda_1(\Lambda) \geq \gamma d$, which implies that the minimum distance between any two points is large when compared with d. This intuitively leads to the BDD Solver to return the same vector $\mathbf{v}$ from which the perturbed vector $\mathbf{x}$ was extended. However, with sufficiently large number of iterations, we are guaranteed that, with probability $1 - 1/poly(n)$, we will produce a vector $\mathbf{x}$ that

allows the BDD to return a lattice vector $\mathbf{y} \neq \mathbf{v}$ that is closer to $\mathbf{x}$ than $\mathbf{v}$ is to $\mathbf{x}$. This is a YES instance (i.e. $\lambda_1(\Lambda) < d$).

3.2.3 Search-LWE to Decision-LWE Reduction

It has been shown in [68] that the search version of LWE can be reduced to the decision version of LWE, which implies that decision LWE is just as hard. Cryptographic applications often rely on the decisional variant of LWE since their security is formally based on the ability of an adversary to distinguish between true and random encryptions.

Lemma 3.2.3 (Search-LWE $\leq_p$ Decision-LWE [68])**.** *Assume we are given a decision $LWE_{n,q,\chi}$ oracle, where $n \geq 1$, $q \geq poly(n)$ is prime, and χ is some distribution on $\mathbb{Z}_q$. Then, there exists an efficient algorithm that can solve the search $LWE_{n,q,\chi}$ given samples $(\boldsymbol{A}, \boldsymbol{z})$ that are generated by oracle $\Theta_{\boldsymbol{s},\chi}$ with some secret $\boldsymbol{s}$.*

Given a LWE sample pair $(\mathbf{a}, z)$, we provide as input into the decision-LWE oracle the pair $(\mathbf{a} + d\mathbf{u}_i, z + dk)$ where, for every $k \in \mathbb{Z}_q$, $\mathbf{u}_i$ is the unit vector with a value of 1 for the element at position i and d is chosen uniformly at random from $\mathbb{Z}_q$. Let s_i be the i^{th} component of $\mathbf{s}$. For any i, if $k = s_i$, then clearly, the pair $(\mathbf{a} + d\mathbf{u}_i, z + ds_i)$ will appear to be from the distribution generated by $\Theta_{\mathbf{s},\chi}$. If $k \neq s_i$, then the pair will instead appear to be from the uniform random distribution. Thus, for each bit s_i, we use the decision-LWE to test if $s_i = k$ by trying all possible $k = q < poly(n)$ and recover $\mathbf{s}$. This takes at most $qn = O(poly(n))$ queries to the decision-LWE oracle.

3.2.4 Worst-case to Average-case Reduction

It has been shown in [68] that the worst-case version of LWE can be reduced to the average-case version of LWE, which implies that the average-case LWE problem is just as hard. This is a useful result for cryptographic applications since this implies

that we need not concern ourselves with drawing strong (hard to solve) instances of the problem for cryptographic primitives that are based on such assumptions. In fact, it has been shown in [41] that we can choose $\mathbf{s}$ to be from *any* distribution assuming that $q = \omega(n^c)$ for all c (i.e. superpolynomial in n). This, however, incurs a manageable increase in the dimension of $\mathbf{s}$.

Lemma 3.2.4 (Worst-case $\leq_p$ Average-case [68])**.** *Given $n \geq 1$, $q \geq 1$, and χ a distribution on $\mathbb{Z}_q$, and decision $LWE_{n,q,\chi}$ oracle that can distinguish between samples of $\Theta_{\boldsymbol{s},\chi}$ and samples from the uniform distribution for a non-negligible fraction of all possible $\boldsymbol{s}$. Then, there exists an efficient algorithm that can, for all possible $\boldsymbol{s}$, distinguish between samples of $\Theta_{\boldsymbol{s},\chi}$ and samples from the uniform distribution.*

Given a decisional average-case $\text{LWE}_{n,q,\chi}$, an input pair $(\mathbf{a}, z)$ and $\mathbf{t} \in \mathbb{Z}_q^n$, we can construct a new pair $(\mathbf{a}, z + \langle \mathbf{a}, \mathbf{t} \rangle)$ that is either a sample from the oracle $\Theta_{\mathbf{s}+\mathbf{t},\chi}$ or a sample from the uniform random distribution. We can then distinguish for *any* $\mathbf{s}$ by calling the average-case oracle with different possible $\mathbf{t}$ until we discover the vector $\mathbf{s} + \mathbf{t}$ that allows us to identify the samples from the $\Theta_{\mathbf{s}+\mathbf{t},\chi}$ oracle.

3.3 Known Solutions

While there still has yet to be an algorithm that could efficiently (in polynomial time) solve the LWE problem, there are still some efforts that attempt to solve this problem in at least not much more than an exponential amount of time.

One simple algorithm is to draw samples from the LWE oracle expecting to find $poly(n)$ equations where each $\mathbf{a}_i$ has one element with a value of 1 and $n-1$ elements with a value of 0. This allows us to individually recover each bit of $\mathbf{s}$ in $2^{O(n \log n)}$ time and number of samples. Another similarly straightforward solution is the maximum likelihood method which uses around $O(n)$ samples (since $q = poly(n)$) and $2^{O(n \log n)}$ time to find the vector $\mathbf{s}$' that satisfies the given equations (in that case the actual $\mathbf{s}$'

$= \mathbf{s}$ with high probability). The BKW algorithm [19] (see Section 5.2) seems to be the best known method to solve the general LWE problem in $2^{O(n)}$ time and number of queries.

3.4 Cryptographic Applications

The versatility and provable hardness of the LWE problem has led to the rise of several innovative and secure cryptographic primitives and systems. Moreover, the inherent structure of the problem aids in constructing efficient and practical implementations that are resilient against polynomially-bounded adversaries and quantum attacks.

One of the first IND-CPA secure public-key cryptosystems based on the LWE problem was devised by Regev [68]. A similar, more efficient scheme was designed in [66] where some of the randomness can be amortized across different encryptions without affecting the asymptotic complexity. Some IND-CCA secure public-key cryptosystems were also proposed in [65, 67].

More interestingly, fully homomorphic encryption (FHE) schemes were also developed based on the hardness of the LWE problem [33, 22, 20, 21]. These schemes allow ciphertext to be modified, with a predictable change in the underlying enciphered message, without the need to decrypt the data first. A *fully*, as opposed to a partial, homomorphic scheme is one where ciphertext modifications can include both additions or multiplications.

Other cryptographic applications include oblivious transfer protocols [66], leakage-resilient encryption schemes [6, 7, 41], identity-based encryption [34, 1, 27], secure constructions of pseudorandom functions [9], and key distribution schemes [27]. Furthermore, studies are under way to determine the types of lattices (ideal, general, etc.) that preserve the hardness of the LWE problem, and can thus be safely used as elements in a scheme.

3.5 Learning Parity with Noise

Given some positive integer $n \geq 1$, an error probability $\eta \in [0, 1/2)$, and a randomly chosen vector $\mathbf{s} \in \mathbb{Z}_2^n$, we define $\Pi_{\mathbf{s},\eta}$ to be an oracle that, when queried, will supply a tuple of the form $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle \oplus e)$ where $\mathbf{a} \in \mathbb{Z}_2^n$ is a vector chosen uniformly at random and $e \in \mathbb{Z}_2$ is selected from the distribution $Ber(\eta)$. After m queries to $\Pi_{\mathbf{s},\eta}$, we will have a set of examples encoded as a matrix $\mathbf{A} \in \mathbb{Z}_2^{n \mathrm{x} m}$ and a corresponding vector $\mathbf{z} \in \mathbb{Z}_2^m$ of noisy labels, where $\mathbf{z} = \mathbf{sA} \oplus \mathbf{e}$ and $\mathbf{e} \in \mathbb{Z}_2^m$ is called the noise vector. To solve the $\mathrm{LPN}_{n,\eta}$ problem, one must recover the secret vector $\mathbf{s}$ given $(\mathbf{A}, \mathbf{z})$.

This is a special case of the LWE problem when $q = 2$. It is interesting to note that, as of yet, there are no hardness relations between LWE and LPN (the hardness proof of LWE only works for a modulus $q > 2\sqrt{n}$). A structurally similar problem, the efficient decoding of random linear codes was shown to be NP-hard [14], which strongly supports, but does not decisively prove, the average-case hardness of LPN. While not as widely used as LWE for encryption schemes, the LPN problem is nevertheless a foundation for several developed cryptographic applications. In this section, we will discuss in more detail three cryptographic systems that could be regarded as representative of the different applications that use the hardness assumption of LPN as a basis for their security.

The apparent significance of the LPN problem in the field of cryptography further lends weight to the importance of ensuring that schemes relying on this problem are made secure against determined attackers. We shall thus revisit this problem in Chapter 5, the main focus of this thesis, where we will examine and devise adversarial algorithms that try to solve the LPN problem as efficiently as possible.

3.5.1 Pseudorandom Generator

A Pseudorandom Generator (PRG) is a function $G : \{0,1\}^l \rightarrow \{0,1\}^n$ that takes as input a seed of size l and outputs a string of size $n > l$ that is indistinguishable from a random string of bits of size n. For some seed d, a PRG G is considered $\epsilon-$secure (ϵ is negligible) against an adversary $A : \{0,1\}^n \rightarrow \{0,1\}$ if:

$$|Pr[A(G(d)) = 1] - Pr[A(U_n) = 1]| \leq \epsilon \tag{3.1}$$

Here, U_n is the uniform distribution of strings in $\{0,1\}^n$. The adversary outputs 1 if it determines that the given string is from G and 0 otherwise.

One of the earlier cryptographic applications of LPN involved the construction of a simple PRG [18]. A full proof on the strength of this primitive is described in the cited paper. Let us assume that an error probability of η is used. For any $\mathbf{A} \in \mathbb{Z}_2^{m \times n}$, $\mathbf{s} \in \mathbb{Z}_2^n$, and random seed d, we define a function $G(\mathbf{A}, \mathbf{s}, d) = (\mathbf{A}, \mathbf{As} + \mathbf{e}(d))$, where $\mathbf{e}$(d) is a function that randomly samples an error vector from $Ber^m(\eta)$ given seed d. To ensure that G has an output that is greater than the size of the seed d, the size of the seed must be at least $H(\eta)m$ where H is the binary entropy function.

An efficient PRG was later constructed based on the syndrome decoding problem [29], which is quite similar to LPN. Moreover, an LPN-based PRG was later developed in [7] that allows for a linear-stretch function in quasi-linear time.

3.5.2 HB Protocol

The Hopper-Blum (HB) protocol [43] and its successor, HB+ [46], are lightweight authentication schemes based on the LPN problem. They were primarily designed to work on devices that possess low processing power. The goal of any authentication protocol is to allow two parties to prove their identities to each other by exchanging protocol messages over a possibly insecure channel. The challenges when designing such a protocol include preventing unauthorized entities from eavesdropping on an

ongoing communication and thwarting the efforts of any malicious third parties that attempt to impersonate one legitimate entity to another during (or after) connection establishment.

Assume that two parties, A and B, wish to authenticate each other with a pre-shared k-bit key $\mathbf{x}$. We define η to be the known probability of error within the range (0,1/2) and q the number of required rounds. The protocol works as follows (initiated by A):

Algorithm 3.5.1 HB Authentication Scheme

---

for i = 1 to q **do**

A selects uniformly at random vector $\mathbf{a}_i \in \{0,1\}^k$ and sends it to B.

B calculates $z_i = \langle \mathbf{a}_i, \mathbf{x} \rangle \oplus e_i$ where $e_i \in \{0,1\}$ is 1 with probability η.

B sends z_i to A.

A calculates $z_i' = \langle \mathbf{a}_i, \mathbf{x} \rangle$ and compares it against the received z_i.

end for

A accepts the authentication if $z_i = z_i'$ for more than $q\eta$ rounds.

Analysis

The HB protocol is considered secure against a probabilistic polynomial time *passive* adversary whose goal is to recover the pre-shared key $\mathbf{x}$. A passive adversary possesses the sole capability of mounting known plaintext attacks. By only observing protocol messages exchanged between the two legitimate parties, a passive adversary may construct the set of noisy linear equations $\mathbf{Ax} = \mathbf{z}$ where $\mathbf{A} = (\mathbf{a}_i, ..., \mathbf{a}_q)$ obtained from A's messages, $\mathbf{z} = (z_1, ..., z_q)$ obtained from B's messages, and $\mathbf{x}$ is the key to recover. This happens to be equivalent to solving the hard LPN problem. In fact, the protocol designer can set parameters k and η to determine the resistance factor of the scheme against any adversaries that try to attack the underlying LPN

problem. A passive attack is one of the main goals of the BKW (Section 5.2.1) and other similar cryptanalysis algorithms.

It was shown, however, that the protocol is not secure against an active adversary: one with the capability of choosing the vectors $\mathbf{a}$ at will. An adversary with this capability can easily recover the key. For example, the attacker can send q unit vectors of value (1,0,...,0) to B to obtain $z_1 = x_1$. A majority vote would determine the actual value of x_1, the first bit of the key. This can be repeated to obtain all the bits of the key.

The improved version of this protocol, HB+, was designed in an effort to defend against such attacks. Now we assume that A and B share two k-bit keys: $\mathbf{x}$ and $\mathbf{y}$. The protocol works as follows:

Algorithm 3.5.2 HB+ Authentication Scheme

---

for i = 1 to q **do**

A selects uniformly at random vector $\mathbf{a}_i \in \{0,1\}^k$ and sends it to B.
B selects uniformly at random vector $\mathbf{b}_i \in \{0,1\}^k$ and sends it to A.

B calculates $z_i = \langle \mathbf{a}_i, \mathbf{x} \rangle \oplus \langle \mathbf{b}_i, \mathbf{y} \rangle \oplus e_i$ where $e_i \in \{0,1\}$ is 1 with probability η.

B sends z_i to A.

A calculates $z_i' = \langle \mathbf{a}_i, \mathbf{x} \rangle \oplus \langle \mathbf{b}_i, \mathbf{y} \rangle$ and compares it against the received z_i.

end for

A accepts the authentication if $z_i = z_i'$ for more than $q\eta$ rounds.

Analysis

The protocol was designed and proved to be secure against active adversaries that observe the response at the receiver's end (i.e. party B). However, an active attack was devised in [37] where the adversary can recover the key by modifying the messages of A and the observing the initiator (A). In particular, the adversary can change A's messages $\mathbf{a}_i$ by adding (mod 2) the unit vector $\mathbf{u}_j$ to each $\mathbf{a}_i$ and observing whether

A accepts or refuses the transaction. If it accepts the transaction then bit $\mathbf{x}_j$ of the key is 0, else it is 1.

Several other variants of HB were developed in the hopes of defending against passive and active attacks. Most recent is the HB# [36] and Trusted-HB [23] which offer light-weight security against several Man-in-the-Middle attacks.

3.5.3 The LPN-C Encryption Scheme

The LPN-C [35] is a probabilistic private-key (symmetric) encryption scheme whose hardness is based on the LPN problem. Unlike traditional symmetric encryption modes whose security rely on the underlying block cipher, LPN-C builds a scheme without using any block ciphers, and instead opting to directly use the hardness of the LPN problem to establish the security of encryption. Regardless, messages are encrypted in blocks (and padding is necessary if the last block is short of the block size).

Definitions

$k \in \mathbb{Z}$ is the security parameter

$r \in \mathbb{Z}$ is the size of a block of message.

$H : \{0,1\}^r \to \{0,1\}^m$ a public error correcting code with correction capacity $\lfloor \frac{d-1}{2} \rfloor$.

$\mathbf{D} \in \mathbb{Z}_2^{k \times m}$ is a shared secret key.

$\mathbf{m} \in \mathbb{Z}_2^r$ is the message to encrypt.

$C \in \mathbb{Z}_2^k \times \mathbb{Z}_2^m$ is the ciphertext.

$\eta \in (0, 1/2)$ is the probability of error.

$\mathbf{e} \in \mathbb{Z}_2^m$ is the error vector where each $e_i \leftarrow Ber(\eta)$.

Analysis

During encryption, it is vital that the Hamming weight of the error vector $\mathbf{e}$ is less than the correction capacity to prevent any decoding errors when H^{-1} is called. We

Algorithm 3.5.3 LPN-C Private-Key Cryptosystem

SECRET KEY: Generate key $\mathbf{D}$ uniformly at random and distribute securely

ENCRYPTION:

Select uniformly at random vector $\mathbf{a} \in \mathbb{Z}_2^k$
Choose at random $\mathbf{e}$ such that Hamming weight of $\mathbf{e} < \lfloor \frac{d-1}{2} \rfloor$
Set $\mathbf{y} = H(\mathbf{m}) \oplus (\mathbf{aD}) \oplus \mathbf{e}$

Set $C = (\mathbf{a}, \mathbf{y})$

Output C

DECRYPTION:

Compute $\mathbf{m'} = \mathbf{y} \oplus (\mathbf{aD})$

Set $\mathbf{m} = H^{-1}(\mathbf{m'})$

Output $\mathbf{m}$

will repeatedly draw new $\mathbf{e}$ until we satisfy this condition. Furthermore, if for any reason, the H^{-1} could not successfully decode the message, then an error flag is set to notify the receiver.

The scheme described in Algorithm 3.5.3 was shown to be secure against adaptive chosen plaintext attacks (CPA) by showing that breaking LPN-C (with a CPA capability) is as hard as distinguishing samples of $\Pi_{\mathbf{s},\eta}$ from random. This should suffice to prove the security of this scheme since it has been shown in [47] that the the hardness of LPN implies that the oracle $\Pi_{\mathbf{s},\eta}$ is sufficiently pseudorandom. To equip the scheme with security against adaptive chosen ciphertext attacks, the authors adopt the Encrypt-then-MAC paradigm by adding a MAC of the ciphertexts to the transmitted messages. A closely related, but more efficient, variant of this scheme was proposed in [7].

4

Cryptanalytic Tools

This chapter is devoted to explaining some of the cryptanalytic techniques which an adversary might use to attack a scheme and recover the key. The significance of these attacks relates to their potential adaptability as (sub-)solutions to the LPN problem. Consequently, the analysis of these tools will aid the discussion of the adversaries in Chapter 5 where we will be using some of the techniques described in this section as components for devising procedures for solving the LPN problem. To simplify the demonstration of the cryptanalytic tools, we will be mostly be concerned with symmetric key cryptosystems that employ a single shared secret key.

4.1 Brute Force Approaches

While not very practical, brute force attacks against cryptographic schemes present a necessary extreme and non-tight bound for an exhaustive search of the secret key. Most of these techniques are usually applied on schemes which have no known exploitable structural weaknesses, and can thus be the best attack an adversary can mount against such schemes. For example, the 56-bit DES block-cipher was shown to be easily broken with an exhaustive key search due to its relatively small key

space. Today's cryptographic security standards [10] recommend a symmetric key size of 112 bits to ensure that such attacks are deemed infeasible.

4.2 Birthday Attacks

Birthday attacks are a sub-class of brute-force attacks that exploit the birthday problem (also known as the birthday paradox) to find collisions between sets of random items. These attacks are quite generic and are usually independent of the scheme or cryptographic primitive. Hence, there might exist other practical and more efficient attacks which are specifically crafted towards specific schemes. Nevertheless, birthday attacks are often seen as a better alternative for an adversary to use than other brute-force approaches.

Theorem 4.2.1 (Birthday Paradox [13])**.** *Given a list* $\boldsymbol{L}$ *containing* N *k-bit distinct pairwise independent elements, let* $q \leq N$ *be the number of elements that we uniformly select (with replacement) at random from* $\boldsymbol{L}$*. Furthermore, let the probability of at least one collision be* $C(N, q)$*, where a collision implies that we selected two elements* $x, y \in \boldsymbol{L}$ *and* $x = y$*. This probability will then be given by* $C(N, q) = O\left(\frac{q(q-1)}{2N}\right)$.

Proof. Let E_i be the event which indicates that the i^{th} item selected from $\mathbf{L}$ is equal to one of the previously selected $(i-1)$ items. Then $Pr[E_i]$ would be at most $\frac{i-1}{N}$ since the i^{th} item that was selected may be equal to at most $\frac{i-1}{N}$ possibly different items (for example, the previous selected items might have been all equal). Hence,

using Theorem A.0.2, we get the following:

$$\begin{aligned} C(n,q) &= Pr[E_1 \vee ... \vee E_q] \\ &\leq \sum_{i=1}^{q} Pr[E_i] \\ &\leq \frac{0}{N} + \frac{1}{N} + ... + \frac{q-1}{N} = \frac{\sum_{i=1}^{q-1} i}{N} \\ &= \frac{q(q-1)}{2N} \end{aligned}$$

□

Corollary 4.2.2 ([13]). *Given a list* $\boldsymbol{L}$ *of* N *items, choosing* $q = O(\sqrt{N})$ *items from* $\boldsymbol{L}$ *uniformly at random will yield* $C(N,q) \leq \frac{1}{2}$.

The birthday paradox has seen many theoretical applications in cryptanalysis with various levels of efficiency. In the case of modes of operations for block ciphers, birthday attacks are used as means for proving the upper and lower bounds for the security of the mode by defining the probability of collisions in the output of an ideal block cipher. Furthermore, the output of hash functions can be analyzed by determining the upper bound on the probability of collisions thus providing a suitable measure of collision resistance.

4.2.1 Birthday-based Algorithms

Ideally, one would want to find collisions in time $O(N)$ where N is the number of items in the list. However, it is not known how this is achieved in the general case. The brute force approach for finding a collision is to try every possible $\binom{N}{2}$ combination of items and test if they are equal.

Alternatively, one may pre-process the list by first sorting it using an efficient sorting algorithm such as merge sort or quicksort, which takes time $\Theta(N \log N)$. The sorted items will easily indicate whether two consecutive items are equal, and

thus, form a collision. This method requires that all the relevant elements must be present in the list before the search for collisions begins.

Hash functions may also be used to find collisions. Unlike sorting, finding collisions using this method can be done at runtime so all the elements need not be available at the start of the procedure. Assume that each element is of size b bits. Furthermore, let $H : \{0,1\}^b \to \{0,1\}^s$ be a collision resistant hash function. We create a hash table T of size 2^s that is indexed using the output of H. Hence, when we are given an element x, we save it in $T[H(x)]$. If two elements x, y are equal, then we necessarily have $H(x) = H(y)$. So if, after hashing some element y, we find that location $T[H(y)]$ is not empty and that $y = T[H(y)]$, then we have found a collision between y and the element currently residing in $T[H(y)]$.

The hash function's collision resistance property is used here not for cryptographic reasons, but for performance reasons: to mitigate the effect of conflict resolution. It is possible that $x \neq y$, but $H(x) = H(y)$. Therefore, if $T[H(x)]$ is not empty, and we try to save y in the hash table, we will find that $y \neq T[H(y)]$ so we need to resolve this collision somehow, for example, using a linked list. A large number of such collisions may yield performance issues, since we now need to search the list of conflicts to find if a collision between equivalent items exists. Therefore, it is vital to set s appropriately to minimize performance degradation.

4.2.2 Generalized Birthday Problem

The birthday problem can be restated by having 2 lists $\mathbf{L}_1, \mathbf{L}_2$ of sizes N_1, N_2, respectively, constructed such that all elements in one list are distinct, and the goal is to find $x \in \mathbf{L}_1$, $y \in \mathbf{L}_2$ such that $x = y$. If each element is of size k-bits, then the expected number of collisions is $\frac{N_1 N_2}{2^k}$ because we have $N_1 N_2$ possible pairs and each pair may be equal with probability $1/2^k$. Hence, we can find on average 1 collision

given $N_1 = N_2 = 2^{k/2}$ samples in $O(2^{k/2})$ time using a merge or hash-join algorithm. The problem was formulated more generally for t-lists by Wagner in [74].

Theorem 4.2.3 (Generalized Birthday Problem [74])**.** *Given a set of t lists $\boldsymbol{L}_1, ..., \boldsymbol{L}_t$ each containing about $2^{k/(1+\log_2 t)}$ elements, the birthday problem can be solved in time and space $O(t.2^{k/(1+\log_2 t)})$.*

Proof. For any k-bit element x in a list, we denote x_l to be the l least significant bits of x. Let t be a power of 2 and 2^l be the size of each list, where l is to be determined. For each pair of lists $\mathbf{L}_i, \mathbf{L}_{i+1}$, where i is odd, we find all pairs of elements $x \in \mathbf{L}_i, y \in \mathbf{L}_{i+1}$ such that $x_l = y_l$, and store $(x \oplus y)$ in a new list $\mathbf{L}_{i,i+1}$, making sure to keep pointers to x and y. Thus, each element in the new lists will have their l least significant bits equal to zero. The expected size of each of the new $t/2$ lists $\mathbf{L}_{i,i+1}$ is given as $\frac{|\mathbf{L}_i| \times |\mathbf{L}_{i+1}|}{2^l} = \frac{2^l \times 2^l}{2^l} = 2^l$.

We repeat the same joining procedure on each pair of lists $\mathbf{L}_{i,i+1}$ and $\mathbf{L}_{i+2,i+3}$ except this time we find elements $x \in \mathbf{L}_{i,i+1}$ and $y \in \mathbf{L}_{i+2,i+3}$ such that $x_{2l} = y_{2l}$ and store $(x \oplus y)$ in the new list $\mathbf{L}_{i,...,i+3}$. Again, each of the new $t/4$ lists contain on average 2^l elements.

This procedure is iterated until we end up with 2 lists where each of the elements have zeros in their $(\log_2 t - 1)l$ least significant bits. We can then finally join the two lists by finding matches between these two lists, where the probability of a match is $1/2^{k-(\log_2 t-1)l}$. Any match indicates that we have found a collision: two elements from the original lists that agree on all their k bits. The expected size of the final combined list is equal to the expected number of collisions between the two remaining lists, which is given as $\frac{2^l \times 2^l}{2^{k-(\log_2 t-1)l}} = \frac{2^{l+l\log_2 t}}{2^k} = 2^{l+l\log_2 t-k}$. This value is at least 1 if $l + l\log_2 t - k \geq 0$ or $l \geq \frac{k}{1+\log_2 t}$. Since we started with t lists each of size 2^l, the time and space complexity is thus $O(t.2^l) = O(t.2^{k/(1+\log_2 t)})$. □

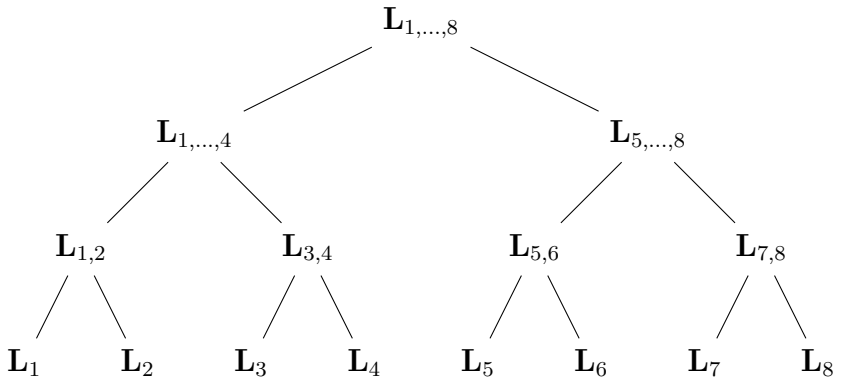

Figure 4.1: Solving the generalized birthday problem for 8 lists

To better clarify how this procedure work, Figure 4.1 illustrates the process for 8-lists using a bottom-up complete binary tree of depth 3. Each intermediate list at height h contains elements whose p least significant bits are all zeros, where $p = \dfrac{hk}{1+\log_2 t} = \dfrac{hk}{4}$. For example, after joining lists $\mathbf{L}_1$ and $\mathbf{L}_2$, all the elements in $\mathbf{L}_{1,2}$ will have their $k/4$ least significant bits equal to zero.

While this technique presents us with a collision between a given set of elements, sometimes we require more than one collision. This method can be easily extended to find $m^{1+\log_2 t}$ collisions (or solutions to the birthday problem) simply by setting $l = k/(1+\log_2 t)+\log_2 m$ such that the starting size of each list is now $2^l = m2^{k/(1+\log_2 t)}$. The only restriction is that $m \leq 2^{k/(\log_2 t+(log_2 t)^2)}$.

4.3 The Walsh-Hadamard Transform

The Walsh-Hadamard Transform (WHT), denoted here as F, is a type of discrete Fourier transform with practical applications in various fields including data compression, signal processing, quantum computations, and more relevantly, cryptanalysis, as we shall demonstrate in this section.

Definition 4.3.1 (Walsh-Hadamard Transform)**.** *For some given function* $f(\boldsymbol{x})$:

$\mathbb{Z}_2^n \to \mathbb{Z}$, *the WHT of* f *is defined as follows:*

$$F(\boldsymbol{w}) = \sum_{\forall\, \boldsymbol{x} \in \mathbb{Z}_2^n} f(\boldsymbol{x})(-1)^{\langle \boldsymbol{w}, \boldsymbol{x} \rangle} \;\; \textit{where } \boldsymbol{w} \in \mathbb{Z}_2^n \tag{4.1}$$

If f is a Boolean function $f(\mathbf{x}) : \mathbb{Z}_2^n \to \mathbb{Z}_2$, then the WHT of f can also be defined as:

$$F(\mathbf{w}) = |\{\mathbf{x} \in \mathbb{Z}_2^n : f(\mathbf{x}) = 1, \langle \mathbf{w}, \mathbf{x} \rangle = 0\}| - |\{\mathbf{x} \in \mathbb{Z}_2^n : f(\mathbf{x}) = 1, \langle \mathbf{w}, \mathbf{x} \rangle = 1\}| \tag{4.2}$$

The Walsh Transform (WT), denoted as F^*, is a slightly different kind of transformation. It applies the WHT on $f^*(\mathbf{x}) = (-1)^{f(\mathbf{x})}$, which has a range that lies in {-1,1}.

$$F^*(\mathbf{w}) = \sum_{\forall\, \mathbf{x} \in \mathbb{Z}_2^n} (-1)^{f(\mathbf{x}) \oplus \langle \mathbf{w}, \mathbf{x} \rangle} \;\; \text{where } \mathbf{w} \in \mathbb{Z}_2^n \tag{4.3}$$

$$F^*(\mathbf{w}) = |\{\mathbf{x} \in \mathbb{Z}_2^n : f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle\}| - |\{\mathbf{x} \in \mathbb{Z}_2^n : f(\mathbf{x}) \neq \langle \mathbf{w}, \mathbf{x} \rangle\}| \tag{4.4}$$

Alternatively, the WT of a Boolean function $f(\mathbf{x}) : \mathbb{Z}_2^n \to \mathbb{Z}_2$ can be derived from the WHT of f as follows:

$$F^*(\mathbf{w}) = -2F(\mathbf{w}) + 2^n \delta(\mathbf{w}) \;\; \text{where } \delta(0) = 1, \delta(\mathbf{w}) = 0 \; \forall \; \mathbf{w} \neq 0 \tag{4.5}$$

The naive time complexity of evaluating the WT for a function is $O(2^{2n})$ since for each of the 2^n values that $\mathbf{w}$ takes on, we need to evaluate the summation over all the possible 2^n values of $\mathbf{x}$. However, we can improve this upper bound using a faster algorithm, which we describe in the next section.

4.3.1 Fast Walsh Transform

The Fast Walsh-Hadamard Transform (FWHT), is an efficient algorithm that computes F^* for $f(\mathbf{x}) : \mathbb{Z}_2^n \to \mathbb{Z}_2$ in time $O(n2^n)$ and can be either performed iteratively or recursively. The procedure in its iterative form is shown in Algorithm 4.3.1 as adapted from [45].

The algorithm accepts the truth table $\mathbf{T}$ for function $f : \mathbb{Z}_2^n \rightarrow \mathbb{Z}_2$, where $T[\mathbf{x}] = f(\mathbf{x}) \; \forall \; \mathbf{x} \in [0, 2^n - 1]$. Using this truth table, the first for-loop evaluates $F(\mathbf{w})$ for f. This takes time $O(n2^n)$ since the outer for-loop is executed $\log_2 2^n = n$ times, and the inner while-loop performs $O(2^n)$ addition/subtraction operations. The second external for-loop simply obtains $F^*(\mathbf{w})$ by implementing Equation 4.5. Note that all the computations are in-place such that the table $\mathbf{T}$ will be replaced with the evaluated $F^*(\mathbf{w})$, and is returned as the output of this algorithm.

Algorithm 4.3.1 Fast Walsh Transform

```
1:  procedure FAST WALSH TRANSFORM(T, n)
2:      for i = 0 to n − 1 do                          ▷ Obtain WHT of function
3:          u ← 2^i
4:          p ← 0
5:          while p < 2^n do
6:              for j = 0 to u − 1 do
7:                  t1 ← T[p + j] + T[p + u + j]
8:                  t2 ← T[p + j] − T[p + u + j]
9:                  T[p + j] ← t1
10:                 T[p + u + j] ← t2
11:             end for
12:             p ← p + 2u
13:         end while
14:     end for
15:
16:     for i = 0 to 2^n − 1 do                        ▷ Obtain WT out of WHT
17:         T[i] ← −2T[i]
18:         if i = 0 then
19:             T[i] ← T[i] + 2^n
20:         end if
21:     end for
22:     return T
23: end procedure
```

4.3.2 Linear Correlation

Assume that we have a system of q linear equations consisting of n Boolean variables and represented as $\mathbf{z} = \mathbf{sA} \oplus \mathbf{e}$ where $\mathbf{A} \in \mathbb{Z}_2^{n \times q}$ is the coefficient matrix, $\mathbf{s} \in \mathbb{Z}_2^n$ is the unknown vector, and $\mathbf{e}$ is an error vector with $e_i = 1$ for some probability $\eta \in [0, 1/2)$. If $\eta = 0$, then we can use simple Gaussian elimination with $q = n$ independent equations to retrieve $\mathbf{s}$. However, for $\eta > 0$, this task becomes much

harder (see Section 3.5). Nevertheless, using q sufficient equations, we can discover vector $\mathbf{s}$ with high probability if we use the FWHT. This is achieved by having the FWHT provide the most probable value of $\mathbf{s}$; that which satisfies the most number of q equations. We describe how this is done after defining some necessary concepts.

Definitions

Definition 4.3.2 (Affine Boolean Function)**.** *An affine Boolean function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ is a degree-1 function that can be represented in the following format:*

$$f(x_1, ..., x_n) = s_n x_n \oplus s_{n-1} x_{n-1} \oplus ... \oplus s_1 x_1 \oplus s_0 \tag{4.6}$$

where $s_0, ..., s_n$ are Boolean coefficients. f is called a linear function if $s_0 = 0$.

The constant s_0 is added to differentiate between a specific function and its complement. We further define a family of n-bit affine Boolean functions as the set of all different possible affine functions that accept n Boolean variables as input. Each function in the family is obtained by setting the tuple of coefficients $(s_1, ..., s_n)$ to one of 2^n different possible combinations. We disregard the value of the constant s_0 as we shall be treating a function and its complement as the same affine function. Table 4.1 shows the truth table for such a family of functions that accept 3 Boolean variables.

We also define the non-linearity of a function, which measures how far a given function is from an affine function.

Definition 4.3.3 (Non-Linearity)**.** *Let H_n be the family of n-input affine Boolean functions. The non-linearity, ϕ, of a Boolean function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ is given by:*

$$\phi(f) = \min_i dist(f, h_i \in H_n) \tag{4.7}$$

where dist(f,g) is the Hamming distance between functions f and g.

f	$f(000)$	$f(001)$	$f(010)$	$f(011)$	$f(100)$	$f(101)$	$f(110)$	$f(111)$
0	0	0	0	0	0	0	0	0
x_0	0	1	0	1	0	1	0	1
x_1	0	0	1	1	0	0	1	1
$x_1 \oplus x_0$	0	1	1	0	0	1	1	0
x_2	0	0	0	0	1	1	1	1
$x_2 \oplus x_0$	0	1	0	1	1	0	1	0
$x_2 \oplus x_1$	0	0	1	1	1	1	0	0
$x_2 \oplus x_1 \oplus x_0$	0	1	1	0	1	0	0	1

Table 4.1: Truth Table for a 3-input Affine Boolean Function Family

Functions with high non-linearity are specifically desirable for stream-cipher based encryption mechanisms that use linear feedback shift registers (see Section 5.4), since linear correlations between the registers' states and their combined output will be harder to discover. The FWHT can be used to find the non-linearity of a function as follows:

$$\phi(f) = \frac{2^n - \max_{\mathbf{w}}|F^*(\mathbf{w})|}{2} \tag{4.8}$$

Hypothesis Testing

Finding the most likely $\mathbf{s}$ given $\mathbf{z}$ and $\mathbf{A}$ is akin to finding the closest affine function to that system of equations. Thus, this problem can also be that of finding the coefficients of an unknown affine function. By constructing the truth table of the unknown function using the given equations, we can compare this table of values against all affine functions in a family to discover the one that yields the minimum non-linearity and, therefore, the maximum similarity.

To construct the truth table for $f(\mathbf{x})$, we find all possible unique patterns (columns) of $\mathbf{A}$ and store their corresponding label (obtained from $\mathbf{z}$) in the table. In other words, for each $\mathbf{a}_i \in \mathbf{A}$, we set $f(\mathbf{a}_i) = z_i$. If their exists multiple $\mathbf{a}_i$ with the same pattern, but with different labels, we store the label that occurred most frequently. A specific pattern might not exist in $\mathbf{A}$ leading to an empty entry in the truth table.

Nevertheless, if the number of missing patterns is minimal then this should not deter from obtaining the correct result.

Once the truth table, $f(\mathbf{x})$, is constructed and submitted to the FWHT procedure, it will be compared efficiently to every affine function g in the family of affine functions of the same input size. For example, for a 3-input function, if $\mathbf{w} = (101)$, $f(\mathbf{x})$ will be compared with $g(\mathbf{x}) = x_2 \oplus x_0$. The value of $\mathbf{w}$ that yields the highest $F^*(\mathbf{w})$ (see Equation 4.5), and consequently, the lowest non-linearity, will be the one that provides the most probable coefficients $\mathbf{s} = \mathbf{w}$ for the unknown function's representation (see Equation 4.6). Incidentally, these coefficients are also the desired unknown vector that we are trying to find.

Based on the analysis of hypothesis testing in [25], it was shown that, given q parity equations ($\mathbf{A}$ has q columns) to hypothesize b bits of the unknown vector s, where $b \geq log_2 q$, the running time of the FWHT is $O(2^b \log_2 q)$. We notice that if $b = \log_2 q$ then this complexity reduces to $O(b2^b)$ as was shown earlier in this section.

4.4 Lattice-Based Cryptanalysis

Lattice reduction algorithms have seen wide use in attacking cryptosystems based on hard problems. Applications range from solving number theoretic problems to approximating solutions of hard lattice problems such as the shortest vector problem. It is therefore helpful to look at how these algorithms work and examine their usefulness as tools for cryptanalysis. In this section, we discuss some of the more relevant procedures that fall under this class. The definitions and algorithms are based on the material in [45]. For a more in-depth analysis on lattice-based cryptanalysis, we refer the reader to the cited source.

The main goal of the lattice-reduction algorithms is to process a given lattice basis and produce a reduced basis for the same lattice. The reduced basis would contain vectors that are shorter than the ones in the initial basis. The quality of the

resultant basis (how orthogonal the basis vectors are to each other) can be measured using the Gram-Schmidt Orthogonalization procedure (see Section 2.2).

Definition 4.4.1 (Reduced Basis)**.** *A lattice basis* $\boldsymbol{B} = (\boldsymbol{b}_1, ..., \boldsymbol{b}_n)$ *is considered reduced if both of the following conditions are satisfied:*

a) $\sigma_{i,j} = \frac{\langle \boldsymbol{b}_i, \tilde{\boldsymbol{b}_j} \rangle}{\langle \tilde{\boldsymbol{b}_j}, \tilde{\boldsymbol{b}_j} \rangle} \leq 1/2 \quad \forall\, 1 \leq i < j \leq n$

b) $||\tilde{\boldsymbol{b}_i}|| \geq \left(\delta - \sigma_{i,i-1}^2\right) ||\tilde{\boldsymbol{b}}_{i-1}||^2 \quad \forall\, 1 < i \leq n$ *and for some* $\frac{1}{4} < \delta \leq 1$

4.4.1 Gauss' Algorithm

Gauss' Algorithm [32] is one of the earliest algorithms used to solve the exact SVP for lattices with dimension $n = 2$. It essentially reduces, in polynomial time, a basis of a two-dimensional lattice in the l_2 norm to find the two successive minima (the two shortest vectors in the lattice).

Algorithm 4.4.1 presents the procedure. The algorithm takes as input two linearly independent vectors $\mathbf{a}, \mathbf{b}$, which represent components of a basis $\mathbf{B}$ for some lattice $\mathcal{L}(\mathbf{B})$, and returns a reduced basis $\mathbf{a}', \mathbf{b}'$ for the lattice. The algorithm iteratively replaces the length of the longer vector in the basis with another shorter basis vector in the lattice. The process halts when this vector becomes shorter than the second basis vector, or equivalently, when $\sigma \leq 1/2$. Note also that a 2-dimensional basis $\mathbf{B} = (\mathbf{a}, \mathbf{b})$ is considered reduced if $||\mathbf{a} + \mathbf{b}|| \geq ||\mathbf{a}||, ||\mathbf{b}||$.

4.4.2 LLL Algorithm

The Lenstra-Lenstra-Lovász (LLL) algorithm [49] is a polynomial time lattice basis reduction algorithm whose goal is to approximately solve the shortest vector problem (SVP_γ) for $\gamma = (2/\sqrt{3})^n$.

Algorithm 4.4.2 shows the procedure for integer lattices and can be extended, if necessary, to work with rational lattices. The input to the algorithm is a non-reduced

Algorithm 4.4.1 Gauss' Algorithm

```
1: procedure GAUSS-REDUCTION(a, b)
2:     if ||a|| < ||b|| then                          ▷ a is the longer vector
3:         Swap a and b
4:     end if
5:     while ||a|| > ||b|| do
6:         σ = ⟨a, b⟩ / ⟨b, b⟩                        ▷ Compute σ that minimizes ||a − σb||
7:         a ← a − σb
8:         Swap a and b
9:     end while
10:    return (a, b)
11: end procedure
```

basis $\mathbf{B} = (\mathbf{b}_1, ..., \mathbf{b}_n)$ of an n-dimensional full rank lattice and a parameter δ that ranges from $\frac{1}{4}$ to 1. The output of this algorithm is a δ-LLL reduced basis.

The LLL algorithm is at its core a relaxed application of Gauss' algorithm for higher dimensions. Whereas in Gauss' algorithm we ensure that the second condition of Definition 4.4.1 is satisfied for $\delta = 1$ (i.e. exactly), it would take far too long to reach this desired result for higher dimensions. Thus, δ acts as a relaxation parameter that can be reduced to accommodate a polynomial running time at the cost of accuracy.

Each loop of the LLL algorithm contains three stages, which are repeatedly executed until every two consecutive basis vectors satisfy the second condition of Definition 4.4.1. In each loop, we find the GSO vectors of the basis vectors, replace the basis vectors with shorter ones (just like the Gauss' Algorithm), and test the condition to check if they pass as reduced basis vectors (using their GSO counterparts). While the worst-case running time of the algorithm is exponential in the dimension of the lattice, it has been shown [31] that the time complexity of the procedure for most lattices, while still exponential, is nevertheless below the conjectured worst-case.

The applications of the LLL algorithm are numerous and diverse. Apart from being able to solve approximate SVP, the algorithm was also found to be useful in factoring integer and rational polynomials (as was intended by the authors of LLL),

Algorithm 4.4.2 LLL Algorithm

```
1: procedure LLL(B, δ)
2:     reduced ← false
3:
4:     while reduced = false do
5:         for i = 1 to n do                          ▷ Gram-Schmidt Orthogonalization
6:             b̃_i = b_i
7:             for j = 1 to i − 1 do
8:                 b̃_i = b̃_i − σ_{i,j} b̃_j            ▷ σ_{i,j} = ⌈ ⟨b_i, b̃_j⟩ / ⟨b̃_j, b̃_j⟩ ⌋
9:             end for
10:        end for
11:
12:        for i = 2 to n do                          ▷ Basis Reduction
13:            for j = i − 1 down to 1 do
14:                b_i = b_i − σ_{i,j} b_j
15:            end for
16:        end for
17:
18:        reduced ← true
19:        for i = 2 to n do                          ▷ Test and Exchange
20:            if ||b̃_i|| < (δ − σ²_{i,i−1}) ||b̃_{i−1}||² then
21:                Swap b_i and b_{i−1}
22:                reduced ← false
23:                Exit loop
24:            end if
25:        end for
26:    end while
27:
28:    return B
29: end procedure
```

performing integer linear programming in bounded dimensions, finding integer relations for a set of real numbers, and solving a variety of number theoretic problems [70] including finding solutions to quadratic equations and computing greatest common divisors. Moreover, we shall demonstrate in the next section how the LLL is used as a sub-procedure to solve the approximate closest vector problem.

More importantly, in this context, the LLL algorithm has also be used as a cryptanalytic tool. In particular, it can be used to break knapsack-based cryptosystems as demonstrated, for example, in [69] where an attack is mounted against the Merkle-Hellman public-key knapsack-based scheme. It can also be used to break an RSA scheme that uses low public exponents [26].

4.4.3 Nearest Plane Algorithm

Proposed by Babai [8], the nearest plane algorithm approximately solves, in polynomial time, the closest vector problem (CVP_γ) with an approximation factor γ that is exponential in the lattice dimension n.

Algorithm 4.4.3 shows the procedure for $\gamma = 2^{n/2}$. The algorithm accepts a lattice basis $\mathbf{B}$ and a target vector $\mathbf{t}$. The output is a vector $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ such that $||\mathbf{v} - \mathbf{t}|| \leq 2^{n/2}||\mathbf{x} - \mathbf{t}|| \; \forall \; \mathbf{x} \in \mathcal{L}(\mathbf{B})$. A tighter approximation factor of $\gamma = (2/\sqrt{3})^n$ can be achieved by appropriate modification of the parameters.

Algorithm 4.4.3 Babai's Nearest Plane Algorithm

procedure NEAREST PLANE($\mathbf{B}, \mathbf{t}$)
  $\mathbf{B} \leftarrow LLL(\mathbf{B}, 3/4)$
  for $i = 1$ to n **do** ▷ Gram-Schmidt Orthogonalization
    $\tilde{\mathbf{b}}_i = \mathbf{b}_i$
    for $j = 1$ to $i - 1$ **do**
      $\tilde{\mathbf{b}}_i = \tilde{\mathbf{b}}_i - \sigma_{i,j}\tilde{\mathbf{b}}_j$ ▷ $\sigma_{i,j} = \left\lceil \frac{\langle \mathbf{b}_i, \tilde{\mathbf{b}}_j \rangle}{\langle \tilde{\mathbf{b}}_j, \tilde{\mathbf{b}}_j \rangle} \right\rfloor$
    end for
  end for

  $\mathbf{u} \leftarrow \mathbf{t}$
  for $i = n$ down to 1 **do**
    $\mathbf{u} \leftarrow \mathbf{u} - \lambda_i \mathbf{b}_i$ ▷ $\lambda_i = \left\lceil \frac{\langle \mathbf{u}, \tilde{\mathbf{b}}_j \rangle}{\langle \tilde{\mathbf{b}}_j, \tilde{\mathbf{b}}_j \rangle} \right\rfloor$
  end for
  $\mathbf{v} \leftarrow \mathbf{t} - \mathbf{u}$
  return v
end procedure

5

LPN Algorithms

In this section, we discuss the design and implementation of some of the existing solutions to the LPN problem and suggest hybrid algorithms that are built on ideas from several previous efforts that tackle this problem. While the main focus of this chapter is to propose improvements to the BKW algorithm, we also examine other possible independent, but related, solutions that have potentially useful practical applications. In hopes of building upon previous work, we strive to enhance our understanding of the underlying complexity of the studied solutions by contrasting the algorithms' individual and comparative performance using different parameters. Comparisons shall be made between the different algorithms in terms of time complexity and number of queries by providing experimental data to support the analysis.

5.1 Adversary Definition

We will examine and analyze the security of the LPN problem (formally defined in Section 3.5) in terms of the computational power of a given adversary. Therefore, we first fully define the adversary's goals and capabilities. This will give us a clear vision of the algorithms' privileges and limitations and enable us to establish a more suitable

and balanced comparison between the different techniques that we will discuss.

An $LPN_{k,\eta}$ adversary, A, in this setting, is one whose goal is to break the scheme by recovering $\mathbf{s} \in \mathbb{Z}_2^k$ given the set of uniformly random examples $\mathbf{A} = (\mathbf{a}_1, ..., \mathbf{a}_n)$ and their corresponding noisy labels $\mathbf{z} = (z_1, ..., z_n)$ supplied by oracle $\Pi_{\mathbf{s},\eta}$. Each example $\mathbf{a}_i$ is represented as a column vector in $\mathbf{A}$. Since we are trying to solve an NP-hard problem (and assuming P $\neq$ NP), A is permitted to exercise oracle $\Pi_{\mathbf{s},\eta}$ in an experiment using sub-exponential number of queries q in sub-exponential amount of time t. At the end of the experiment, A is expected to present its guess of $\mathbf{s}$ regardless of the decisiveness of this guess.

An adversary, A, that runs in time t, makes q queries to oracle $\Pi_{\mathbf{s},\eta}$, and uses at most μ amount of memory is said to (t, q, μ, ϵ)-solve an instance of the $LPN_{k,\eta}$ problem if:

$$Pr[\mathbf{s} \leftarrow \{0,1\}^k : A^{\Pi_{\mathbf{s},\eta}}(1^k) = \mathbf{s}] \geq \epsilon(k) \tag{5.1}$$

When discussing the algorithms in this section, we will denote $\mathbf{x}$ as being the adversary's output or "guess" of $\mathbf{s}$. In order to solve the LPN problem, A should output $\mathbf{x}$ such that it is equal to $\mathbf{s}$ with high probability. Given that η is the probability of error, we will refer to $\sigma = \frac{1}{2} - \eta$ as the *bias*. Therefore, since the probability of error can be written as $\eta = \frac{1}{2} - \sigma$, we can also write the probability of correctness as $(1 - \eta) = \frac{1}{2} + \sigma$. Using Hoeffding Bounds, we can determine that the lower bound on the number of equations needed to correctly find $\mathbf{s}$ with high probability is $n = O(k/\sigma^2)$ [19]. Clearly, the adversary's task of recovering $\mathbf{s}$ becomes easier as the bias increases (η decreases). In fact, if $\eta = 0$, the problem reduces to solving a system of $n = O(k)$ linear equations using Gaussian elimination, a poly-time solution.

While most of the algorithms discussed in this chapter have comparable asymptotic performance, we will experimentally demonstrate how some offer better practical results than others in a concrete setting. For example, $2^{O(k/20)}$ is a better bound

than $2^{O(k/\log k)}$ for certain values of k especially if the hidden constants manifest as significant contributing factors.

5.2 BKW Algorithm

The BKW algorithm [19] was the first algorithm designed to solve the LPN problem in sub-exponential time and number of queries: $2^{O(k/\log k)}$, where k is the size of the unknown vector $\mathbf{s}$. In this section, we discuss the underlying concepts used in the design of the algorithm to achieve the aforementioned complexity.

5.2.1 Noise Amplification

The BKW algorithm makes use of linear combinations as a means to reduce the number of equations and obtain the target unit vectors. This leads us to consider how the error is amplified as we add several noisy examples.

For any i, let $\mathbf{a}_i \in \mathbb{Z}_2^k$ be an example and $b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle$ represent its corresponding label. We will let $(\mathbf{a}_i, z_i)$ denote a tuple that represents an example and its corresponding noisy label (i.e. $z_i = b_i \oplus 1$ with some error probability η). When we apply the addition operator to any two tuples $(\mathbf{a}_i, z_i)$ and $(\mathbf{a}_j, z_j)$, we linearly combine them to get the resultant tuple $(\mathbf{a}_r, z_r) = (\mathbf{a}_i \oplus \mathbf{a}_j, z_i \oplus z_j)$. In this case, z_r will be the true label of both combined equations if either both z_i and z_j are corrupted or both are correct. The following lemma provides the probability of correctness in the general case (for n examples).

Lemma 5.2.1 (Noise Amplification [19])**.** *Given n examples $\boldsymbol{a}_1, ..., \boldsymbol{a}_n$ and their corresponding labels $z_1, ..., z_n$, where each $z_i = \langle \boldsymbol{a}_i, \boldsymbol{s} \rangle \oplus 1$ with probability η, the probability that the linear combination of the examples is correct is given as $Pr[\langle \boldsymbol{a}_1 \oplus ... \oplus \boldsymbol{a}_n, \boldsymbol{s} \rangle = z_1 \ \oplus ... \oplus \ z_n] = \frac{1}{2} + \frac{1}{2}(1-2\eta)^n$.*

Proof. The base case, $n = 1$, results in probability of correctness $Pr[\langle \mathbf{a}_1, \mathbf{s} \rangle = z_1] = \frac{1}{2} + \frac{1}{2}(1-2\eta) = 1 - \eta$, as expected. Since the examples $\mathbf{a}_1, ..., \mathbf{a}_n$ are uniformly drawn

at random, they can be considered independent of each other. By induction, we show the following:

$$
\begin{aligned}
&Pr[\langle \mathbf{a}_1 \oplus ... \oplus \mathbf{a}_n, \mathbf{s} \rangle = z_1 \ \oplus ... \oplus \ z_n] \\
&= Pr[\langle \mathbf{a}_1 \oplus ... \oplus \mathbf{a}_{n-1}, \mathbf{s} \rangle = z_1 \ \oplus ... \oplus \ z_{n-1}].Pr[\langle \mathbf{a}_n, \mathbf{s} \rangle = z_n] \\
&+ Pr[\langle \mathbf{a}_1 \oplus ... \oplus \mathbf{a}_{n-1}, \mathbf{s} \rangle \neq z_1 \ \oplus ... \oplus \ z_{n-1}].Pr[\langle \mathbf{a}_n, \mathbf{s} \rangle \neq z_n] \\
&= \left(\frac{1}{2} + \frac{1}{2}(1-2\eta)^{n-1}\right)\left(\frac{1}{2} + \frac{1}{2}(1-2\eta)\right) \\
&+ \left(1 - \frac{1}{2} - \frac{1}{2}(1-2\eta)^{n-1}\right)\left(1 - \frac{1}{2} - \frac{1}{2}(1-2\eta)\right) \\
&= \frac{1}{4}\left(1 + (1-2\eta)^{n-1} + (1-2\eta) + (1-2\eta)^n\right) \\
&+ \frac{1}{4}\left(1 - (1-2\eta)^{n-1} - (1-2\eta) + (1-2\eta)^n\right) \\
&= \frac{1}{4}\left(2 + 2(1-2\eta)^n\right) = \frac{1}{2} + \frac{1}{2}(1-2\eta)^n
\end{aligned}
$$

□

5.2.2 Sample Definitions

In the context of the BKW algorithm, we will be treating each labeled example $\mathbf{a}$ drawn from oracle $\Pi_{\mathbf{s},\eta}$ as a set of a blocks, each of size b bits, where $a = \lceil k/b \rceil$. Thus, for each example, group 1 contains bits 1 to b, group 2 contains bits $b+1$ to $2b$, and so on. As it is, the probability that $\mathbf{a}$'s label is incorrect is η.

Definition 5.2.1 (Sample Set)**.** *Let a, b be some positive integers such that $k = ab$. An (n, k, i, p)-sample set is a set of n k-bit examples whose last ib bits are guaranteed to be all zeros and whose labels are corrupted with probability p.*

The following Lemma is derived from Lemma 5 in [19].

Lemma 5.2.2 (Sample Set Derivation)**.** *Given an (n, k, i, p)-sample set, we can construct in time $O(n)$ an $(n', k, (i+1), p')$-sample set where each example in the set has zeros for its last $(i+1)b$. The size of the group is $n' \geq n - 2^b$, and the probability of error is given by $p' = \frac{1}{2} - \frac{1}{2}(1-2p)^2$.*

Proof. Let $\{(\mathbf{a}_1, z_1), ..., (\mathbf{a}_n, z_n)\}$ be a (n, k, i, p)-sample where each example $\mathbf{a}_j$ has zeros in its last ib bits (i.e. bit groups $[a - i + 1, a]$ for $i > 0$), and its corresponding label z_j is corrupt with probability p.

The examples can be categorized into separate partitions, in time $O(n)$, based on the bit pattern in bit group $(a - i)$. Since we are categorizing on b bits, the number of partitions is at most 2^b, each partition containing on average $n/2^b$ examples.

Next, in every partition, we select at random an example $\mathbf{a}_r$, add it (mod 2) to every other example in the partition (including the labels), then discard it. Since all the examples in one partition have the same bit pattern in group $(a - i)$, this process will zero out the bits in that group: bits $[(a - i - 1)b + 1, (a - i)b]$. The examples will then have all zeros in their last $(i + 1)b$ bits. Furthermore, since we are discarding at most one example per partition, the total number of examples n' after this process is at least $n - 2^b$. Using Lemma 5.2.1, the error probability of a resultant vector after adding two noisy examples (having error probability p) is $1 - (\frac{1}{2} + \frac{1}{2}(1-2p)^2) = \frac{1}{2} - \frac{1}{2}(1-2p)^2$. □

5.2.3 Design

The main idea of the algorithm is to start with a $(m, k, 0, \eta)$-sample from oracle $\Pi_{\mathbf{s},\eta}$ and derive a $(m - (a-1)2^b, k, (a-1), p')$-sample $\mathbf{A}'$ by iterating Lemma 5.2.2 $(a - 1)$ times. Each example $\mathbf{a}'$ in $\mathbf{A}$ is a result of combining 2^a examples using 2^{a-1} linear combinations. Consequently, the labels will be incorrect with probability $p' = \frac{1}{2} - \frac{1}{2}(1-2p)^{2^{a-1}}$. The labeled examples in $\mathbf{A}'$ will have zeros in the last $(a-1)b$ bits and arbitrary values in the first b bits. Within $\mathbf{A}$', we search for an example

$\mathbf{a}'_j = \mathbf{u}_j$, where $\mathbf{u}_j$ is a unit vector with a 1 in position j, and estimate bit j of $\mathbf{x}$ by setting $x_j = z_j$. The number of vectors $\mathbf{u}_j$, for any j, that will exist in $\mathbf{A}'$ is on average $\frac{1}{2^b} \times (m - (a-1)2^b)$, so m must be selected such that this number is at least 1. By repeating this sample reduction process $q = poly((1-2\eta)^{-2^a}, b)$ times with new set of labeled examples each time, we can recover the first b bits of $\mathbf{x}$ with high probability by taking the majority vote. To recover the rest of the bits of $\mathbf{x}$, we run the procedure q times for groups $\alpha = (2, ..., a)$ of $\mathbf{x}$. The only difference is that, for each example, we zero out the bits in all the other groups except group α.

Algorithm 5.2.1 illustrates an implementation-specific version of the BKW algorithm which includes lower-level details that will aid in understanding the subtle performance issues that may arise in practice. Furthermore, we note that concrete values of parameters a, b, and q are specified. These specific values will be used to demonstrate how the algorithm works in detail. While not necessarily optimal in all cases, these values were found to be sufficiently representative of the average performance of the algorithm during experimentation. Nevertheless, to ensure completeness, we also present tests for various values of b in Section 5.9.3.

After setting the values of a, b, and q, we start by selecting the index of the target group α (the set of b bits of $\mathbf{s}$ to estimate) at the beginning of the target loop (line 6). The estimation loop (line 10) is repeated at least q times per target group to acquire the votes necessary to estimate group α of $\mathbf{x}$ with high probability. Each estimation loop consists of drawing $m = a2^b$ new labeled examples from $\mathbf{A}$ and storing them in $\mathbf{A}'$. This is the initial $(m, k, 0, \eta)$-sample. We then successively apply Lemma 5.2.2 on every group (categorize and merge) except the target group to obtain the desired $(2^b, k, (a-1), p')$-sample. At the end of an estimation loop, we estimate the target group based on $\mathbf{u}_j$ vectors and accumulate the votes for each bit j in the target group α (line 24). For bit j of group α, the number of votes for 1 are

recorded $\mathbf{v}_1(j)$ and the number of votes for 0 are recorded in $\mathbf{v}_0(j)$. After at least q estimates, we hypothesize the target group via a straightforward majority vote (line 30), so for each bit j, if $\mathbf{v}_1(j) > \mathbf{v}_0(j)$, x_j is set to 1 else it is set to 0.

Algorithm 5.2.1 BKW algorithm

```
1:  procedure BKW(A, z, η)
2:      a ← (1/2) log_2 k
3:      b ← ⌈k/a⌉
4:      q ← b · (1 − 2η)^{−2^a}
5:
6:      for α = 1 → a do                                   ▷ Target Loop
7:          Initialize votes array v
8:          for γ = 1 → q do                               ▷ Collect q votes
9:              estimated ← false
10:             while estimated is false do                ▷ Estimation Loop
11:                 m ← a2^b
12:                 A' ← subset(A, z, m)
13:                 c ← a
14:                 repeat                                 ▷ Combination Loop
15:                     if c ≠ α then
16:                         {Q_j}_{j=0}^{2^b−1} ← categorize(A', 1 + b(c − 1), bc)   ▷ see Alg. B.0.1
17:                         for j = 0 to 2^b − 1 do
18:                             Q_j ← merge(Q_j)           ▷ see Alg. B.0.2
19:                         end for
20:                         A' ← ⋃_{j=0}^{2^b−1} Q_j
21:                     end if
22:                     c ← c − 1
23:                 until c = 0
24:                 (v_0, v_1) ← estimate(A', α, b)        ▷ see Alg. B.0.3
25:                 if (v_0(i) + v_1(i)) ≥ γ ∀ i then
26:                     estimated ← true                   ▷ All votes accounted for
27:                 end if
28:             end while
29:         end for
30:         x_α = hypothesize(v_0, v_1, α)                 ▷ Hypothesize partial solution
31:     end for
32:
33:     x ← (x_1||...||x_a)                                ▷ Reconstruct key
34:     return x
35: end procedure
```

5.2.4 Analysis

We analyze Algorithm 5.2.1 to find an upper bound on the time complexity T, number of required queries to the oracle Q, and the amount of memory M required

to run the procedure.

Proposition 5.2.3. *For any fixed error rate η, the BKW algorithm solves the LPN problem with time complexity $2^{O(k/\log_2 k)}$.*

Proof. The number of estimation loops per target group is at least $q = b(1-2\eta)^{-2^a}$, each loop taking time $O(a2^b + 2^b) = O(a2^b)$ due to the categorization and merging procedure implied by the combination loops and the estimation process. We may need to repeat each individual estimation loop more than once if $estimated = false$ at the end of the loop. This happens when we cannot estimate bit j of the target group α due to the absence of vector $\mathbf{u}_j$ in the resultant sample set. Thus, each estimation loop is repeated at most a times to obtain the desired $\mathbf{u}_j$ vectors for each bit j of the target group. We repeat the q estimation loops for each of the a groups. Hence, the total time complexity is given by:

$$T(a, b, \eta) = O(a \times aq \times a2^b) = O\left(a^3 b 2^b (1-2\eta)^{-2^a}\right) \tag{5.2}$$

Setting $a = \frac{1}{2}\log_2 k$ and $b = 2k/\log_2 k$, we get the following result:

$$T(k, \eta) = O\left(2^{O(k/\log_2 k)} \times \frac{k}{4}(\log_2 k)^2 \times (1-2\eta)^{-\sqrt{k}}\right) \tag{5.3}$$

□

Proposition 5.2.4. *For any fixed error rate η, the BKW algorithm solves the LPN problem using $2^{O(k/\log_2 k)}$ number of queries.*

Proof. For each of the q estimation loops, we require $a2^b$ new examples from the oracle. Since we execute each estimation loop at most a times, the number of new queries needed per target group is $a^2 2^b q$. For all a groups, the number of queries needed is $a^3 2^b q$. Setting $a = \frac{1}{2}\log_2 k$ and $b = 2k/\log_2 k$,

$$Q(k, \eta) = O(a^3 b 2^b (1-2\eta)^{-2^a}) = O\left(2^{O(k/\log_2 k)} \times \frac{k}{4}(\log_2 k)^2 \times (1-2\eta)^{-\sqrt{k}}\right) \tag{5.4}$$

□

Proposition 5.2.5. *For any fixed error rate η, the BKW algorithm solves the LPN problem with $O(a2^b)$ amount of memory.*

Proof. The space complexity is dominated by the combined size of $\mathbf{A}$ and $\mathbf{z}$, which is $(k+1) \times n$. Knowing that we require $Q(k, \eta)$ examples, we can ask the oracle in advance to supply $n = Q(k, \eta)$, which leads to space complexity of $O(kn)$ or $O(ka^3 2^b (1-2\eta)^{-2^a})$. However, we can significantly reduce the amount of memory necessary by drawing examples on demand. That is, we ask the oracle directly each time we need a new set of $a2^b$ examples instead of having all the necessary examples at the start of the procedure. This reduces the space complexity to $O(a2^b)$. After the examples are categorized, combined, and used for estimation at the end of an estimation loop, they are discarded to make room for a new set of examples. □

5.3 Hash Collision

A slightly different, perhaps more general, approach to the BKW algorithm exploits the birthday paradox to find unit vectors among a series of linear combinations between any pair of vectors. To implement this solution, we will use a hash table (see Section 4.2.1) to store the examples.

The hash function $H : \{0,1\}^k \to \mathbb{Z}_m$ uses the bit pattern of the example as the key, and the hash value indicates the location $i \in [1, m]$ in the hash table to which the example will be inserted. We assume that H is modeled as a Random Oracle Model [12] and that collisions are rectified using a linked list. To avoid severe performance degradation due to long linked lists, one can use universal hash functions to implement a perfect hashing mechanism. One may also minimize the amortized cost by judiciously increasing the size of the hash table at regular intervals as the number of elements in the table increase. We consider the latter solution in our

implementation.

5.3.1 Design

Algorithm 5.3.1 presents the procedure in detail. We notice that the algorithm's behavior does not depend on the value of η. Instead, the error rate indirectly affects the complexity by influencing the size of the matrix $\mathbf{A}$ presented as an input to the algorithm; the higher the error rate, the more examples will be requested from the oracle. To begin, an augmented matrix $\mathbf{A}$' is created containing both the examples and their labels. We also initialize the set $\mathbf{Q}$, a hash table that will hold the examples after they are hashed using H.

The while-loop in lines (7-23) is repeated until we run out of examples in $\mathbf{A}'$. Within each loop, we select a random example $\mathbf{a}'_r$ from $\mathbf{A}'$, flip its first bit and check if this new vector is present in the hash table. If not, we flip the second bit of $\mathbf{a}'_r$ (the first bit is reverted back to its original value) and try again with this new vector. If none of the bit-flipped vector modifications provide a match in the hash table, we insert $\mathbf{a}'_r$ into $\mathbf{Q}$. However, if we happen to find a match for $\mathbf{a}'_r$ with bit j flipped, then this means we have found a vector $\mathbf{b}$ residing within $\mathbf{Q}$ whose Hamming distance to $\mathbf{a}'_r$ is exactly one. Linearly combining $\mathbf{a}'_r$ and $\mathbf{b}$ will result in unit vector with a 1 in position j. We estimate x_j as being equal to the combined labels of both $\mathbf{a}'_r$ and $\mathbf{b}$ before we delete vector $\mathbf{b}$ from $\mathbf{Q}$. To increase the probability of successfully recovering the $\mathbf{x}$, we find at least $q = (1 - 2\eta)^{-2}$ such estimates per bit of the key.

Algorithm 5.3.1 Hash Collisions

```
1: procedure HASH COLLISIONS(A, z)
2:     (k, n) ← Dim(A)
3:     Build (k + 1) × n matrix A′ = {a′_i}_{i=1}^n, where a′_i = (a_i||z_i)
4:     Q ← {}
5:     v_0(i) = 0 ∀ i = [1, k]
6:     v_1(i) = 0 ∀ i = [1, k]
7:     while |A′|> 0 do
8:         a′_r ← randomSelect(A′)
9:         A′ = A′ − a′_r
10:        for i = 1 to k do
11:            b ← a′_r ⊕ u_i                ▷ u_i is the unit vector with 1 in position i
12:            if H(b) ∈ Q then
13:                q ← a′_r ⊕ b
14:                if q_{k+1} = 0 then        ▷ Label of (a′_r ⊕ b)
15:                    v_0(i) = v_0(i) + 1
16:                else
17:                    v_1(i) = v_1(i) + 1
18:                end if
19:            end if
20:            Q ← Q − b
21:        end for
22:        Q ← Q ∪ a′_r
23:    end while
24:    x ← hypothesize(v_0, v_1)
25:    return x
26: end procedure
```

5.3.2 Analysis

Due to the birthday paradox, we begin encountering collisions between vectors with Hamming distance 1 with probability 0.5 when the number of elements in $\mathbf{Q}$ reaches approximately $2^{k/2}$. For each of the examined examples, we need to check at most k different possibilities for collisions. Since we need at least q more different estimates for each of the k bits, and assuming that the hashing operation takes $O(1)$ time, this results in the following running time:

$$T_1(k, \eta) = O(2^{k/2} + O(1) \times k^2 q) = O\left(2^{k/2} + k^2(1 - 2\eta)^2\right) \tag{5.5}$$

The space and query complexity is also the same. We can improve the time and space complexity by using the generalized birthday problem [74] solution described in Section 4.2.2. Using this extension, we can find kq collisions (or solutions to the

k	T_1	T_2
32	68736	6.8×10^7
64	4.3×10^9	1.3×10^9
80	1.1×10^{12}	4.6×10^9
128	1.8×10^{19}	1.1×10^{11}
256	3.4×10^{38}	1.2×10^{14}
512	1.2×10^{77}	1.9×10^{19}

Table 5.1: Time Complexity for the Hashing Algorithm using $\eta = 0.45$

t-sum problem) by incrementally constructing $t = 2^{k/\log_2(kq)}$ lists of examples drawn from the oracle, each of size $O\left(kq2^{k/(1+\log_2 t)}\right)$, in time and space $O(tkq \times 2^{k/\log_2 t})$, which for any fixed η is an asymptotically more preferable upper bound than the complexity in Equation 5.5. Thus, the updated time and space complexity would be:

$$T_2(k, \eta) = O(kq \times 2^{k/\log_2(kq)} \times 2^{\log_2 kq}) \tag{5.6}$$

Table 5.1 provides a numerical comparison of theoretical performance T_1 and T_2 for different values of k using $\eta = 0.45$ so $q = 100$.

5.4 Fast Correlation Attacks

The algorithms described in this section harvest their gain in speed by employing techniques commonly used in *fast correlation attacks* against stream ciphers whose pseudorandom keystream outputs are generated by a non-linear combination of linear feedback shift registers (LFSRs) (see Figure 5.1). These attacks attempt to find the linear correlation between the unknown internal state of the LFSRs and the known, but noisy, output of the non-linear combination generator. By discovering such a relationship, one can then find, with non-negligible bias, the output of the registers and even succeed in reconstructing the initial state of the LFSRs. Some of the most notable attacks are discussed in [60, 54, 61] along with their related performance and design improvements.

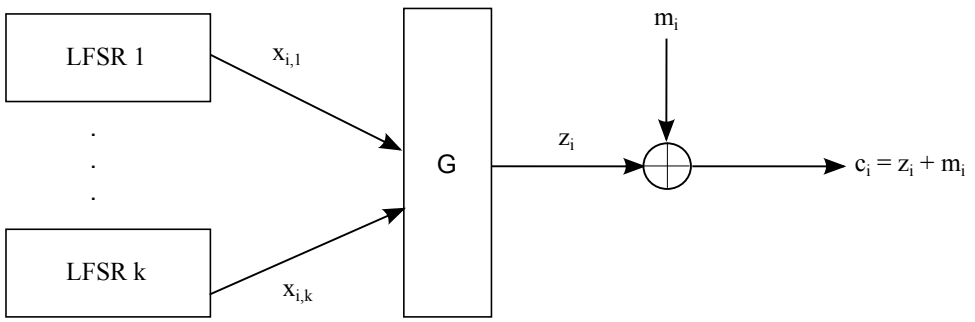

Figure 5.1: The outputs of the LFSRs $(x_1, ..., x_k)$ at state i is combined using some non-linear function G to create the next bit in the keystream

5.4.1 FMICM Algorithm

Proposed by Fossorier et al. [30], the FMICM algorithm (named after the original authors) solves the LPN problem by utilizing sub-components of an advanced fast correlation attack [62]. In particular, the algorithm employs sample decimation, linear combination, and hypothesis testing to find the bits of the key.

Design

The whole procedure is detailed in Algorithm 5.4.1. Before the process proper begins, the parameters $b \leq k, h \leq (k-b), w \geq 2$ must be specified. In the sample decimation phase, we search for examples in the given matrix $\mathbf{A}$ that have all zeros in their last b bits and store these examples in a decimated matrix $\mathbf{A}' = (\mathbf{a}'_1, ..., \mathbf{a}'_{n'})$ of size $(k+1) \times n'$ where the first row of $\mathbf{A}'$ contains the labels of the examples. Since the examples in $\mathbf{A}$ are drawn at random from a uniform distribution, the expected number of examples in $\mathbf{A}'$ is $n' = \dfrac{n}{2^b}$.

In the linear combination phase, we try all possible linear combinations of $c \leq w$ examples from $\mathbf{A}'$ to find a resultant vector with weight 1 in positions $[h+1, k-b]$ (i.e. only a single 1 in these positions). The vector, $\mathbf{p}$, produced by the *nextCombination(n′, c)* function provides the indices $(i_1, ..., i_c)$ of the next c examples out of n' examples to linearly combine. The function *linearCombination(**A′**, **p**)*

simply adds (mod 2) the examples (columns) in $\mathbf{A}'$ whose indices are given by $\mathbf{p}$ to produce the resultant vector $\mathbf{r} = \mathbf{a}'_{i_1} \oplus ... \oplus \mathbf{a}'_{i_c} = (r_0, ..., r_k)$. The bits $\mathbf{r}$ in position $[0, h]$ could take on any arbitrary value. For each $j \in [h+1, k-b]$, we locate and store in matrix $\mathbf{Q}_j$ at least q such resultant vectors whose j^{th} bit is equal to 1.

In the hypothesis phase, for each possible value of the first h bits of the unknown vector $\mathbf{x}$ we estimate the j^{th} bit of $\mathbf{x}$ based on the majority vote evaluation of $(1||\mathbf{x})\mathbf{Q}_j = 0$, where the 1 is prefixed to $\mathbf{x}$ so that the label is included. When performing this evaluation, we note that the last b bits of the examples (columns) in $\mathbf{Q}_j$ are all zeros, and the bits in position $[h+1, k-b]$ have weight 1 (the location of that 1 being in position j). Thus, the bits $(x_{h+1}, ..., x_k) \backslash x_j$ are of no consequence in the evaluation of $(1||\mathbf{x})\mathbf{Q}_j$ since they will always be multiplied by a zero in the corresponding example bit. Only the hypothesized bits $(x_1, ..., x_h)$ will affect the estimation of x_j. To test if a given hypothesis $(x_1, ..., x_{k-b})$ is acceptably valid, we compute the Hamming weight of $(1||\mathbf{x})\mathbf{A}'$ and check if we get a value that is close to the expected weight of the error vector $\mathbf{e}$. If the check passes, we add this partially estimated $\mathbf{x}$ to the set $\mathbf{L}$ of most likely candidates for the true solution.

Once we have finished iterating over all possible hypotheses, we use the vectors in $\mathbf{L}$ as the starting point in estimating the last b bits of $\mathbf{x}$. We repeat Phase 2 to generate weight 1 vectors in positions $[k-b+1, k]$, repeat Phase 3 to estimate bits $(x_{k-b+1}, ..., x_k)$ using the same procedure that was performed for the previously estimated $(k-b)$ bits, and update $\mathbf{L}$ accordingly. The fully estimated $\mathbf{x}$ can be recovered by testing the most probable solutions and deciding which candidate solution yields a vector $\mathbf{y} = (\mathbf{xA} \oplus \mathbf{z})$ with a Hamming weight that is as close to zero as possible (i.e. most of the equations are correctly evaluated using this solution).

Algorithm 5.4.1 FMICM Algorithm

procedure FMICM($\mathbf{A}, \mathbf{z}, \eta$)
    Set algorithm parameters b, h, w
    $(k, n) \leftarrow Dim(\mathbf{A})$ ▷ Dimensions of $\mathbf{A}$
    $q \leftarrow (1 - 2\eta)^{-2w}$

    $\mathbf{A}' \leftarrow \{\}$ ▷ Phase 1: Decimation
    for $i = 1$ to n **do**
        if $\mathbf{a}_i(j) = 0 \; \forall \; j \in [k - b + 1, k]$ **then**
            $\mathbf{A}' \leftarrow \mathbf{A}' \cup (z_i || \mathbf{a}_i)$ ▷ Build $(k + 1) \times n'$ matrix $\mathbf{A}'$
        end if
    end for

    $\mathbf{Q}_j \leftarrow \{\} \; \forall \; j \in [h + 1, k - b]$ ▷ Phase 2: Linear Combination
    for $c = 2$ to w **do**
        while $(\mathbf{p} \leftarrow nextCombination(n', c)) \neq$ null **do**
            $\mathbf{r} \leftarrow linearCombination(\mathbf{A}', \mathbf{p})$
            if $\sum_{j=h+1}^{k-b} r_j = 1$ **then**
                $u \leftarrow findIndexOfOne(\mathbf{r}, h + 1, k - b)$
                $\mathbf{Q}_u \leftarrow \mathbf{Q}_u \cup \mathbf{r}$ ▷ Categorize results
            end if
        end while
    end for

    $\mathbf{L} \leftarrow \{\}$ ▷ Phase 3: Hypothesizing
    for $i = 0$ to $2^h - 1$ **do**
        for $j = (h + 1)$ to $(k - b)$ **do**
            $\mathbf{x}_h \leftarrow (x_1, ..., x_h) = i$
            $\mathbf{x} \leftarrow (\mathbf{x}_h || x_{h+1}, ..., x_k)$
            Estimate x_j by evaluating $(1||\mathbf{x})\mathbf{Q}_j = 0$
        end for
        if $\text{wt}((1||\mathbf{x})\mathbf{A}') < \eta n'$ **then**
            $\mathbf{L} \leftarrow \mathbf{L} \cup \mathbf{x}$
        end if
    end for

    Using $\mathbf{L}$, repeat Phases 2 and 3 to estimate bits x_j for $j = [k - b + 1, k]$
    $\mathbf{x} \leftarrow \min_{\mathbf{x} \in \mathbf{L}} \text{wt}(\mathbf{xA} \oplus \mathbf{z})$
    return $\mathbf{x}$
end procedure

Analysis

The decimation phase has a time complexity of $O(n)$ because we need to search all the examples and identify vectors with all zeros in their last b bits. While it is entirely possible that it could be the dominant factor when computing the total complexity,

the authors argue that the decimated matrix $\mathbf{A}'$ can be pre-computed during sample collection, and can thus be disregarded.

The linear combination phase has a complexity of $O\left(\binom{n'}{w}\right)$ per-bit due to the fact that the last iteration in the loop of line 14 ($c = w$) will be the dominant factor in this phase. However, using the approach in [25], we can reduce this time complexity to $O\left(\binom{n'/2}{w/2}\right)$ for each bit that must be estimated.

The naive time complexity of the hypothesis phase (shown in lines 24-33) is $O(2^h q)$ for each bit that must be estimated since we need to evaluate the equations in $\mathbf{Q}_j$ for each possible hypothesis of the first h bits, where $|\mathbf{Q}_j| = O(q)$ for any j. We can improve the efficiency of this phase by instead applying the Fast Walsh Transform (see Section 4.3) to evaluate the equations and find the best estimate of x_j, which effectively reduces the work to $O(2^h \log q)$. Since q is usually set to $(1-2\eta)^{-2w}$, we can write the complexity as $O(2^h w \log (1-2\eta)^{-2})$.

When we estimate the last b bits of $\mathbf{x}$, we merely repeat Phases 2 and 3 to solve a smaller variation of the same problem. Thus, the complexity (per-bit) of each phase remains the same for this second, and last, iteration of the algorithm. We can write the total complexity of the entire process as:

$$T(n,k,b,h,w,\eta) = k \times O\left(\binom{n/2^{b+1}}{w/2} + 2^h w \log (1-2\eta)^{-2}\right) \tag{5.7}$$

The original paper describes how one may find the optimum value of b and h to minimize the time and space complexity of the procedure. While w can be seen as a contributing factor in this determination, the authors have noted that the selection of w does not greatly affect the performance. In fact, even when we set $w = 2^{a-1}$ (where a is the number of groups, as is the case in the BKW algorithm), the FMICM algorithm exhibits a better theoretical performance.

5.4.2 The Cube Root Method

A similar, more recent approach was devised by U. Wagner [75], which results in a solution to the LPN problem in time $2^{O(k/3)}$ and $2^{O(k/\log k)}$ number of queries assuming that the bias $\sigma = 1/poly(k)$.

Design

Once again, the process is divided into a decimation phase, a linear combination phase and a hypothesis phase. The whole procedure is detailed in Algorithm 5.4.2. Very much like the FMICM algorithm, the parameter b is used to determine the number of low-order bits in the decimated examples that are all zeros (i.e. the last b bits of the $\mathbf{x}$ are not significant in evaluating the equations in the decimated matrix). The parameter h determines the number of bits of $\mathbf{x}$ that we wish to hypothesize in Phase 3. The parameter w is an even integer that indicates the number of linear combinations that must be performed.

In the sample decimation phase, we search for examples in the given matrix $\mathbf{A}$ that have all zeros in their last b bits and store these examples in a decimated matrix $\mathbf{A}' = (\mathbf{a}'_1, ..., \mathbf{a}'_{n'})$ of size $(k+1) \times n'$ where the first row of $\mathbf{A}'$ contains the labels of the examples. Since the examples in $\mathbf{A}$ are drawn at random from a uniform distribution, the expected number of examples in $\mathbf{A}'$ is $n' = \frac{n}{2^b}$. At the end of this stage, all examples in $\mathbf{A}'$ will have zeros for their last b bits.

The linear combination phase is divided into two parts. The first part (lines 13-17) finds all $w/2$ linear combinations in $\mathbf{A}'$ and stores the results in $\mathbf{G}$. The number of examples in $\mathbf{G}$ will be $\binom{n'}{w/2}$. The examples in G will then be categorized into groups based on the value of the bits $[k-(b+h)+1, k-b]$ using the same *categorize* function that was used in the BKW algorithm. This will result in 2^h groups, where each group $\mathbf{Q}_j$ contains on average $C = \frac{\binom{n'}{w/2}}{2^h}$ examples. The second part performs

linear combinations between each different pair of examples for every group, saving the results into a newly initialized $\mathbf{G}$. The expected number of examples in $\mathbf{G}$ would then be $2^h \times \binom{C}{2}$. At the end of this stage, all examples in $\mathbf{G}$ will have zeros for their last $(b+h)$ bits.

In the hypothesis phase, we need to try all possible combinations of the first $k-(b+h)$ bits of $\mathbf{x}$ in evaluating the equations in $\mathbf{G}$. We can accomplish this task efficiently by using the Fast Walsh Transform. Note that the hypothesis procedure is performed on a set of vectors that are generated by adding w noisy examples, which implies that the overall error rate has increased (see Lemma 5.2.1). After obtaining the partially estimated $\mathbf{x}$, we repeat the last two phases to recover the rest of $\mathbf{x}$.

Analysis

The decimation phase, while optional, has a time complexity of $O(n)$ because we need to search all the examples and identify vectors with all zeros in their last b bits. Assuming that this process is performed during sample collection, this overhead can be disregarded.

The first part of the linear combination phase requires $\binom{n'}{w/2} = O((n')^{w/2})$ operations to generate on average $C = \dfrac{\binom{n'}{w/2}}{2^h}$ examples in each of the 2^h groups. The second part requires on average $2^h \times \binom{C}{2} = 2^h \times O(C^2) = O\left(\dfrac{(n')^w}{2^h}\right)$ operations to generate as many examples. Thus, the total work in this phase is $O\left((n')^{w/2} + \dfrac{(n')^w}{2^h}\right)$.

Using the Fast Walsh Transform, hypothesizing $(k-b-h)$ bits using $\dfrac{(n')^w}{2^h}$ examples will yield a running time of $O\left(2^{k-b-h} \log \dfrac{(n')^w}{2^h}\right)$. We can write the total

Algorithm 5.4.2 Cube-Root Algorithm

procedure CUBE-ROOT ALGORITHM($\mathbf{A}, \mathbf{z}, \eta$)
  Set algorithm parameters w, b, h
  $(k, n) \leftarrow Dim(\mathbf{A})$

  $\mathbf{A}' \leftarrow \{\}$ ▷ Phase 1: Decimation
  for $i = 1$ to n **do**
    if $\mathbf{a}_i(j) = 0 \; \forall \; j \in [k - b + 1, k]$ **then**
      $\mathbf{A}' \leftarrow \mathbf{A}' \cup (z_i || \mathbf{a}_i)$ ▷ Build $(k + 1) \times n'$ matrix $\mathbf{A}'$
    end if
  end for

  $\mathbf{G} = \{\}$ ▷ Phase 2: Linear Combination
  while $(\mathbf{p} \leftarrow nextCombination(n', w/2)) \neq$ null **do**
    $\mathbf{r} \leftarrow linearCombination(\mathbf{A}', \mathbf{p})$
    $\mathbf{G} \leftarrow \mathbf{G} \cup \mathbf{r}$
  end while
  $\{\mathbf{Q}_j\}_{j=0}^{2^h-1} \leftarrow categorize(\mathbf{G}, k - b - h + 1, k - b)$
  $\mathbf{G} \leftarrow \{\}$
  for $j = 0$ to $2^h - 1$ **do**
    $n_j \leftarrow NoOfExamples(\mathbf{Q}_j)$
    while $(\mathbf{p} \leftarrow nextCombination(n_j, 2)) \neq$ null **do**
      $\mathbf{r} \leftarrow linearCombination(\mathbf{Q}_j, \mathbf{p})$
      $\mathbf{G} \leftarrow \mathbf{G} \cup \mathbf{r}$
    end while
  end for

  $\mathbf{L} = \{\}$ ▷ Phase 3: Hypothesis
  $\eta' = \frac{1}{2} - \frac{1}{2}(1 - 2\eta)^w$
  for $i = 0$ to $2^{k-b-h} - 1$ **do**
    $\mathbf{x}_h \leftarrow (x_1, ..., x_h) = i$
    $\mathbf{x} \leftarrow (\mathbf{x}_h || x_{k-b-h+1}, ..., x_k)$
    if $\text{wt}((1||\mathbf{x})\mathbf{G}) < \eta' n'$ **then**
      $\mathbf{L} \leftarrow \mathbf{L} \cup \mathbf{x}$
    end if
  end for

  Using $\mathbf{L}$, repeat Phases 2 and 3 to estimate bits x_j for $j = [k - b - h + 1, k]$
  $\mathbf{x} \leftarrow \min_{\mathbf{x} \in \mathbf{L}} \text{wt}(\mathbf{xA} \oplus \mathbf{z})$
  return $\mathbf{x}$
end procedure

complexity of the entire process as:

$$T(n,k,b,h,w,\eta) = O\left(\left(\frac{n}{2^b}\right)^{w/2} + \frac{\left(\frac{n}{2^b}\right)^w}{2^h} + 2^{k-b-h}\log\frac{\left(\frac{n}{2^b}\right)^w}{2^h}\right) \tag{5.8}$$

5.5 LF2 Algorithm

In [51], two different and enhanced LPN algorithms were proposed. The first algorithm, the LF1 algorithm modifies the last phase (hypothesis phase) of the BKW algorithm to achieve a more efficient solution, and shall be discussed in more detail in Section 5.8. The LF2 algorithm is a heuristic procedure based on the work in [74], exploiting the birthday problem to solve an LPN instance using much less queries than is required in BKW.

5.5.1 Design

The algorithm replaces the *merge*() function of Algorithm 5.2.1 with the procedure shown in Algorithm 5.5.1. Instead of combining one example with all the other examples in a group then discarding it, we linearly combine each different pair of examples within a group to generate new examples. This allows us to increase the number of examples we have for the *estimate*() function without increasing the noise rate in the final set, thus lowering the upper bound on the number of required queries to the oracle.

Algorithm 5.5.1 LF2 Combination Sub-Procedure

```
1: procedure LF2-COMBINE(A)
2:     A' ← {}
3:     (k, n) ← Dim(A)
4:     while (p ← nextCombination(n, 2)) ≠ null do
5:         r ← linearCombination(A, p)
6:         A' ← A' ∪ r
7:     end while
8:     return A'
9: end procedure
```

5.5.2 Analysis

Let us define the following Lemma, a modified version of Lemma 5.2.2 to aid in the discussion:

Lemma 5.5.1 (LF2 Sample Set Derivation)**.** *Given an (n, k, i, p)-sample set, we can construct in time $O(n^2/2^b)$ an $(n', k, (i+1), p')$-sample set where each example in the set has zeros for its last $(i+1)b$. The size of the group is $n' = O(n^2/2^b)$, and the probability of error is given by $p' = \frac{1}{2} - \frac{1}{2}(1-2p)^2$.*

Proof. Let $\{(\mathbf{a}_1, z_1), ..., (\mathbf{a}_n, z_n)\}$ be a (n, k, i, p)-sample where each example $\mathbf{a}_j$ has zeros in its last ib bits (i.e. bit groups $[a-i+1, a]$ for $i > 0$), and its corresponding label z_j is corrupt with probability p.

The examples can be categorized into separate partitions, in time $O(n)$, based on the bit pattern in bit group $(a-i)$. Since we are categorizing on b bits, the number of partitions is at most 2^b, each partition containing on average $n/2^b$ examples.

Next, in every partition, we find all $\binom{(n/2^b)}{2}$ different linear combinations in time and save them in a new partition, discarding the original vectors. This takes total time $O(n^2/2^b)$ for all partitions. Since all the examples in one partition have the same bit pattern in group $(a-i)$, this process will zero out the bits in that group: bits $[(a-i-1)b+1, (a-i)b]$. The examples will then have all zeros in their last $(i+1)b$ bits. The total number of examples n' after this process is equal to $\binom{(n/2^b)}{2} \times 2^b = O(n^2/2^b)$. Using Lemma 5.2.1, the error probability of a resultant vector after adding two noisy examples (having error probability p) is $1 - (\frac{1}{2} + \frac{1}{2}(1-2p)^2) = \frac{1}{2} - \frac{1}{2}(1-2p)^2$. □

Given an $(a2^b, k, 0, \eta)$-sample from oracle $\Pi_{\mathbf{s},\eta}$, we can derive a $(n', k, (a-1), p')$-sample by applying Lemma 5.5.1 $(a-1)$ times, where $n' = a^{2^{a-1}}2^b$ and $p' = \frac{1}{2} - \frac{1}{2}(1-2p)^{2^{a-1}}$. Next, we run the *estimate*() function of this new set to find the desired unit vectors. The number of vectors $\mathbf{u}_j$, for any j, in the new set is on

average $O\left(\frac{1}{2^b} \times a^{2^{a-1}} 2^b\right) = O(a^{2^{a-1}})$ as opposed to an average of 1 such unit vector in the classic BKW algorithm. Thus, the number of required estimates q is reduced by a factor of $a^{2^{a-1}}$, which leads to query complexity of:

$$Q(a, b, \eta) = O(a \times aq \times a2^b) = O\left(a^3 b 2^b \frac{(1-2\eta)^{2a}}{a^{2^{a-1}}}\right) \tag{5.9}$$

However, the time complexity remains about the same as the original BKW algorithm since the work per estimate has increased:

$$T(a, b, \eta) = O(a \times aq \times a^{2^{a-1}} 2^b) = O\left(a^2 b 2^b (1-2\eta)^{-2^a}\right) \tag{5.10}$$

5.6 Random Sampling

The previous algorithms were all deterministic in the sense that after some bounded number of queries to the oracle, the unknown vector $\mathbf{x}$ will be correctly recovered by the adversary. In 2008, Carrijo at al. [24] proposed a simple probabilistic and passive attack against the HB protocol. The attack tries to solve the underlying LPN problem by randomly sampling a fixed-size subset of examples from the set of all given noisy linear equations and then solves the system of linear equations to recover the secret vector. If the sampled subset of equations happens to be error-free (no corrupted labels), then we can successfully obtain the unknown vector. However, if it appears that we have solved for an incorrect $\mathbf{x}$, we try again with another round of random sampling.

5.6.1 Design

Algorithm 5.6.1 depicts the procedure in detail. We note that the algorithm accepts an integer $\lambda \geq 0$. The procedure is composed of three main parts, which are repeated until we are sure that the computed vector $\mathbf{x}$ is equal to $\mathbf{s}$ with high probability. The first part is contained in the function *randomSelect* (line 5), which chooses c random

columns from $\mathbf{A}$ with their associated labels from $\mathbf{z}$ and stores them in matrix $\mathbf{Q}$ and vector $\mathbf{y}$. To ensure that, with high probability, linearly independent columns are selected, we use λ to select slightly more columns than needed. The second part (line 6) will find an estimate of the secret vector via Gaussian elimination using the selected subset of columns and labels ($\mathbf{Q}$, $\mathbf{y}$). The *gaussianElimination*() function will return a *null* vector if there exists either no solution or many solutions. The third part (line 9) computes the Hamming distance between the set of all noisy labels and the labels generated by the correct candidate $\mathbf{x}$.

If $\mathbf{x} = \mathbf{s}$ then $\text{wt}(\mathbf{z}' \oplus \mathbf{z}) = \text{wt}(\mathbf{xA} \oplus (\mathbf{sA} \oplus \mathbf{e})) = \text{wt}(\mathbf{e})$. Since the expected weight of the error vector is $\eta.n$ and $\eta \in [0, 1/2)$, then we expect that $\mathbf{x}$ is correct with high probability if $\text{wt}(\mathbf{e}) < 0.4n$. Conversely, if $\mathbf{x} \neq \mathbf{s}$, then the expected value of $\text{wt}(\mathbf{z}' \oplus \mathbf{z})$ is $0.5n$, which fails the check.

Algorithm 5.6.1 Random Sampling

```
1: procedure RANDOM SAMPLING(A, z, λ)
2:     c ← k + λ
3:     x ← null
4:     while x is null do
5:         (Q, y) ← randomSelect(A, z, c)
6:         x ← gaussianElimination(Q, y)
7:         if x not null then
8:             z' ← xA
9:             if wt(z' ⊕ z) < 0.4n then
10:                return x
11:            end if
12:        end if
13:    end while
14: end procedure
```

5.6.2 Analysis

The amount of time that the algorithm takes is solely dominated by $\mathcal{P}$, the expected number of repetitions of the while-loop in line 4. This value can be derived from the probability that we choose k correct examples out of a set n examples. This

probability can be written as:

$$p_{correct} = \frac{\binom{(1-\eta)n}{k}}{\binom{n}{k}} \tag{5.11}$$

Within each loop, a cost is incurred for performing matrix multiplication on the whole sample set (line 8). Thus, the time complexity is given as:

$$T(n,k) = \mathcal{P} \times nk = \frac{1}{p_{correct}} \times nk = \frac{\binom{n}{k}}{\binom{(1-\eta)n}{k}} \times nk \tag{5.12}$$

The query complexity is given as $Q = O(n)$ and the memory required is $M = O(nk)$. The time and query complexity are intricately tied together; finding the required sample size is equivalent to finding n that minimizes T. Figure 5.2 shows the expected value of n for different noise rates η and k. Since this is a theoretical average, practical values of n were often observed to be higher to diminish the possibility of finding duplicate, yet incorrect solutions. This increase, however, is not more than a small polynomial factor of n.

5.7 Information Set Decoding

There exists other problems that have a structure similar to the LPN problem. In fact, some are even a simple transformation away from being formulated as an instance of LPN. The solutions developed for these problems can be considered as suitably adaptable to be used for solving the LPN problem assuming that the time invested for adaptation is polynomially bounded. One such closely related problem is the *computational syndrome decoding* (CSD) problem. We will examine a relatively efficient algorithm that solves the CSD problem and analyze the feasibility of adapting this algorithm to work on LPN instances.

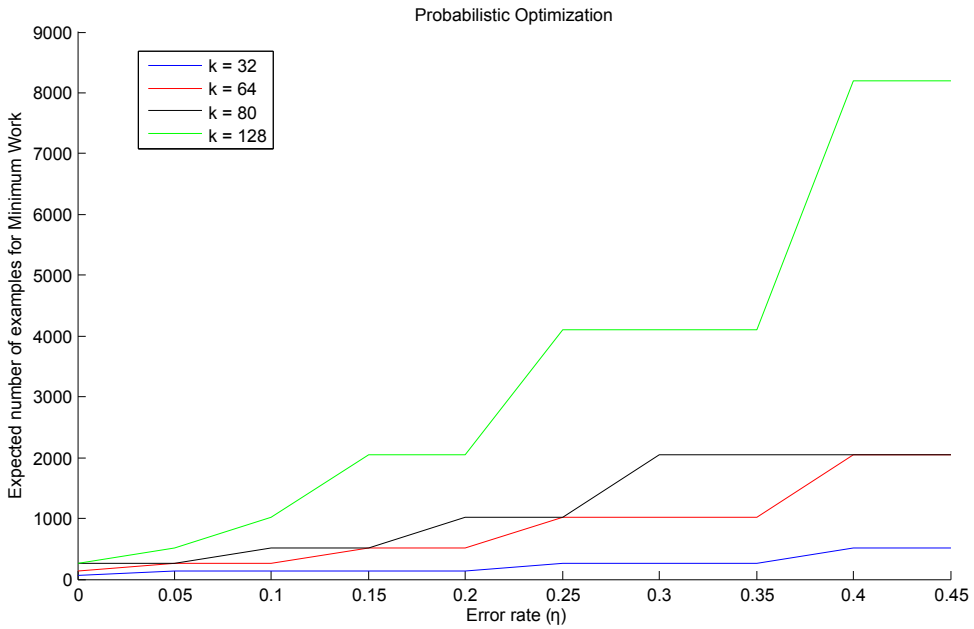

Figure 5.2: For $k = 32$, 64, 80, and 128 the figure illustrates the optimum value of the sample size n that minimizes the time complexity for a given error rate

5.7.1 Computational Syndrome Decoding

An $[n, k, d]$ binary linear code C is a subspace of the finite field $\mathbb{F}_2^n$ with dimension k, a code length of n bits, and a minimum Hamming distance of d between any two n-bit codewords $\mathbf{c}_1, \mathbf{c}_2 \in C$. A code is characterized by its *generator matrix* $G \in \mathbb{F}_2^{k \times n}$, a basis that can be used to enumerate all possible codewords, i.e. $C = \{\mathbf{xG} : \mathbf{x} \in \mathbb{F}_2^k\}$. Furthermore, we say that $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$ is a parity matrix for code C if $\mathbf{Hc} = 0 \; \forall \; \mathbf{c} \in C$. We can view the parity matrix as a way of detecting whether a given point $\mathbf{y}$ is a correct codeword or not, which leads us to the definition of the CSD problem.

Definition 5.7.1. *Given a parity matrix* $\boldsymbol{H} \in \mathbb{F}_2^{(n-k) \times n}$ *and a vector* $\boldsymbol{t} \in \mathbb{F}_2^{(n-k)}$*, the CSD problem can be solved if one can efficiently find a vector* $\boldsymbol{e} \in \mathbb{F}_2^n$ *such that the syndrome* $\boldsymbol{t} = \boldsymbol{He}$*.*

The vector $\mathbf{e}$ represents the error vector and indicates how far an arbitrary point

$\mathbf{y} \in \mathbb{F}_2^n$ is from a codeword in C. That is, $\mathbf{y} = \mathbf{c} \oplus \mathbf{e}$ for some $\mathbf{c}$. The syndrome can then be represented as $\mathbf{t} = \mathbf{H}(\mathbf{c} + \mathbf{e}) = \mathbf{He}$ since we already know that $\mathbf{Hc} = 0$. Notice that if $\mathbf{e}$ is zero (there is no error), then the syndrome is also zero. Once we obtain $\mathbf{e}$, we can decode $\mathbf{y}$ and obtain $\mathbf{c} = \mathbf{y} \oplus \mathbf{e}$. To ensure unique decoding of any point, we define the error correction capability of a code as $w = \lfloor \frac{d-1}{2} \rfloor$. Consequently, this implies that $wt(\mathbf{e})$, the weight of $\mathbf{e}$, must not exceed w. Thus, we shall assume that $wt(\mathbf{e}) = w$ in the worst case. We can easily see the parallelism between this problem and the bounded decoding problem (Section 2.3.7).

One might think of solving the CSD problem as finding the subset of columns of $\mathbf{H}$ whose linear combination results in the syndrome vector. For example, referring to the hypothetical scenario in Equation 5.13, we notice that an error vector of weight 3 set to the indicated value will select columns 1, 3, and 4 from the parity matrix and add them (mod 2) to get the desired result that is equal to the syndrome $\mathbf{t}$. Hence, this error vector is considered a solution to this problem.

$$\underset{\mathbf{H} \in \mathbb{F}_2^{5\times 7}}{\begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}} \quad \underset{\mathbf{e} \in \mathbb{F}_2^{7\times 1}}{\begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}} \quad = \quad \underset{\mathbf{t} \in \mathbb{F}_2^{5\times 1}}{\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}} \tag{5.13}$$

The brute force approach is to try all possible $\binom{n}{w}$ combinations of column additions (all possible weight-w $\mathbf{e}$ vectors). However, there are other algorithms that will yield better performance using minimal amount of memory. One particular algorithm is the *information set decoding* procedure, which was first introduced and used by McEliece [56].

5.7.2 ISD Algorithm

The algorithm is composed of two main steps. Given the parity matrix $\mathbf{H}$ and the syndrome vector $\mathbf{t}$, the first step is to represent $\mathbf{H}$ in a form that allows for the problem to be easier to solve. We achieve this task by first randomly permuting the columns of $\mathbf{H}$ (and, hence, the bits in $\mathbf{e}$), then applying Gaussian elimination on the permuted $\mathbf{H}$ (and $\mathbf{t}$) to mold it into its reduced row echelon format $(\mathbf{Q}|\mathbf{I}_{n-k})$, where $\mathbf{Q} \in \mathbb{F}_2^{(n-k)\times k}$ and $\mathbf{I}_{n-k}$ is the identity matrix of size $(n-k)$. We denote $\mathbf{t}'$ as the result of computing Gaussian elimination on $\mathbf{t}$, and $\mathbf{e}'$ the result of applying the permutation (the same permutation applied on $\mathbf{H}$) on $\mathbf{e}$. We thus have the following formatted matrix $\mathbf{H}'$ after completion of the first part of the algorithm:

$$\overbrace{\left(\begin{array}{c|c} \mathbf{Q} & \mathbf{I}_{n-k} \end{array}\right)}^{\mathbf{H}'} \underbrace{\qquad}_{k} \underbrace{\qquad}_{(n-k)} \left(\begin{array}{c} \\ \mathbf{e}' \\ \\ \end{array}\right) = \left(\begin{array}{c} \\ \mathbf{t}' \\ \\ \end{array}\right) \tag{5.14}$$

The second part of the procedure aims to search for the error vector $\mathbf{e}'$ that results in a solution to the problem $\mathbf{t}' = \mathbf{H}'\mathbf{e}'$. Knowing the weight w of the error vector, we set $p < w$, also referred as the subweight, to be some integer that represents the conjectured weight of the first k bits of $\mathbf{e}'$. The first k bits of $\mathbf{e}'$ are column selectors for $\mathbf{Q}$. If e'_j is 1 for some $j \leq k$ then this means that column $j \leq k$ of $\mathbf{H}'$ (i.e. column j of $\mathbf{Q}$) is selected for linear combination. The goal is to find a linear combination of p columns of $\mathbf{Q}$ where the resultant vector $\mathbf{r} = \mathbf{q}_{i_1} \oplus ... \oplus \mathbf{q}_{i_p}$ has a Hamming distance to $\mathbf{t}'$ of exactly $w - p$. If such an $\mathbf{r}$ exists, we set to 1 the bits $j > k$ of $\mathbf{e}'$ that correspond to the unit vectors (columns) in I_{n-k} such that when these unit vectors are added to $\mathbf{r}$, we obtain exactly $\mathbf{t}'$. Furthermore, since we now know that,

out of the first k bits of $\mathbf{e}'$, only bits i_1 to i_p are set to 1, we have found $\mathbf{e}'$.

If there is no such linear combination of p vectors that yield a resultant vector with Hamming distance $w - p$ to $\mathbf{t}'$, this implies that the weight of the first k bits of $\mathbf{e}'$ is not p, and that our conjecture is incorrect. Hence, we repeatedly apply the second part of the procedure with new permutations of $\mathbf{H}$ until we find an $\mathbf{e}'$ that satisfies the conditions. Once we find $\mathbf{e}'$ we can reverse the permutation to obtain $\mathbf{e}$.

5.7.3 Design

The ISD problem was chiefly studied in the context of a solution to the CSD problem. However, we notice that this solution can also be applied to the LPN problem due to LPN's structural similarity to CSD. In particular, instead of focusing on recovering $\mathbf{x}$ for some set of noisy linear equations, we can instead seek to discover the error vector that was added to the labeled examples. If the error vector is discovered then we can easily find $\mathbf{x}$ by applying Gaussian elimination to evaluate $\mathbf{xA} = (\mathbf{z} \oplus \mathbf{e})$.

To transform an instance of an LPN problem to an instance of a CSD problem, we have to find some matrix $\mathbf{H} \in \mathbb{Z}_2^{(n-k)\times n}$ such that $\mathbf{HA}^T = 0$, where $\mathbf{A}$ is the set of examples that are acquired by an LPN oracle. In other words, $\mathbf{H}$ is the null space of $\mathbf{A}^T$. The reasoning behind finding $\mathbf{H}$ is that when we multiply the labels $\mathbf{z}$ by $\mathbf{H}$, we get $\mathbf{Hz} = \mathbf{H}(\mathbf{A}^T\mathbf{x} + \mathbf{e}) = \mathbf{HA}^T\mathbf{x} + \mathbf{He} = \mathbf{He}$. The term $\mathbf{Hz}$ can then be referred to as $\mathbf{t}$, the syndrome, and the ISD algorithm can then be applied here.

Algorithm 5.7.1 describes the procedure in detail. Given the a set of labeled examples corrupted with some probability η, we first transform the LPN instance into a CSD instance by finding the corresponding $\mathbf{H}$ and $\mathbf{t}$ as described previously. We then repeatedly execute the initialization and search parts of ISD until we obtain the desired error vector.

Unlike in the CSD problem where we are given the weight of the error vector, in the LPN problem, the exact weight is unknown but we know that it is on average ηn.

Hence, when testing the Hamming distance of a resultant vector, we try a range of weights defined by parameters w_{min} and w_{max} centered around the average (expected) weight. Furthermore, the number of linear combinations p is varied between p_{min} and p_{max} to increase the probability of finding the partial weight of $\mathbf{e}'$. Once we find a resultant vector $\mathbf{r}$ that passes the Hamming weight check, we use $\mathbf{v}$, which contains the indices of the columns of $\mathbf{Q}$ used to create $\mathbf{r}$, to set the corresponding bits in the first part of $\mathbf{e}'$. The rest of $\mathbf{e}'$ is set using the indices of the columns of $\mathbf{I}_{n-k}$. The original $\mathbf{e}$ is recovered by reversing the permutation imposed on $\mathbf{H}$, and the error vector is then used to filter out the error in the labels $\mathbf{z}$.

5.7.4 Analysis

The LPN transformation consists of finding the parity matrix (null space) of $\mathbf{A}^T$, which takes $poly(n)$ time. The two main factors affecting the time complexity of the procedure are $\mathcal{P}$, the number of repetitions of the while-loop at line 8 and C, the number of combinations that are required to be tested within each loop. The former depends on the probability of obtaining p ones in the first k bits of $\mathbf{e}'$ after a permutation:

$$\frac{1}{\mathcal{P}} = \frac{\sum_{p=p_{min}}^{p_{max}} \binom{k}{p}\binom{n-k}{\eta n-p}}{\binom{n}{\eta n}} \tag{5.15}$$

Thus, the total time complexity is given by:

$$T(n,k,\eta,p) = \mathcal{P} \times \sum_{p=p_{min}}^{p_{max}} \binom{k}{p} = \frac{\binom{n}{\eta n}}{\sum_{p=p_{min}}^{p_{max}} \binom{k}{p}\binom{n-k}{\eta n-p}} \times \sum_{p=p_{min}}^{p_{max}} \binom{k}{p} \tag{5.16}$$

The query complexity is given by $Q = O(n)$, and the required memory is $M = O(nk)$. Given some k and η, T can be minimized by selecting optimum values for n and p. This will consequently give us the size of the required sample size. We will consider, for simplicity, that $p = p_{min} = p_{max}$. Referring to Figure 5.3, we notice

Algorithm 5.7.1 Adapted Information Set Decoding

```
1: procedure ADAPTED ISD(A, z, η)
2:     Set parameters w_min, w_max, p_min, p_max
3:     (k, n) ← Dim(A)
4:     w ← [w_min, w_max]
5:     (H, s) ← transformLPN(A, z)
6:
7:     e = null
8:     while e is null do
9:         (H, P) ← permute(H)                          ▷ Part 1 Initialization
10:        (H', s') ← rref(H, s)
11:        (Q, I_{n−k}) ← H'
12:
13:        for p = p_min to p_max do                    ▷ Part 2: Search and Test
14:            while (v ← nextCombination(k, p)) ≠ null do
15:                r ← linearCombination(Q, t)
16:                if hammingDistance(r, s') = (w − p) then
17:                    e'_j = 0 ∀ j ∈ [1, n]
18:                    for i = 1 to p do                ▷ Set first k bits of e'
19:                        e'_{t_i} = 1
20:                    end for
21:                    g ← r ⊕ s'
22:                    for i = (k + 1) to n do          ▷ Set the rest of the bits of e'
23:                        e'_i = g_i
24:                    end for
25:                    e = reversePermute(e', P)
26:                    z' = (z ⊕ e)
27:                    x = gaussianElimination(A, z')
28:                    return x
29:                end if
30:            end while
31:        end for
32:    end while
33:
34: end procedure
```

that, even for different values of p, the performance stabilizes at a minimum after $n = poly(k)$ number of examples. Very much like the Random Selection algorithm, a constant surplus of examples should be supplied in practice to avoid reaching duplicate solutions that do not match the true solution.

Figure 5.4 shows how the optimum value of p is determined. Based on the left figure ($1/\mathcal{P}$ vs. p), we state a couple of intuitively justifiable observations. Firstly, as the error rate is increased, the values of p that yield, *with non-negligible probability*, an error vector with weight p in its first k bits also increase. This is attributed to

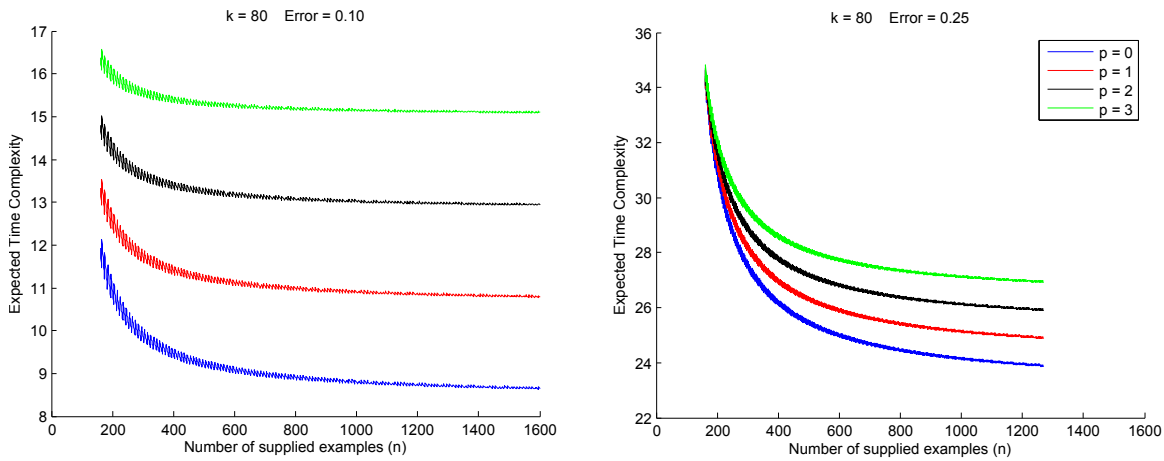

Figure 5.3: For $k = 80$ and error rates $\eta = 0.1$ (left) and $\eta = 0.25$ (right), the time complexity $\log T$ for $p = [0, 3]$ is computed using different values of n.

the fact that higher error rates produce large weight error vectors, so finding an error vector with a relatively small weight in its first k bits becomes highly unlikely. Secondly, as the error rate rises, the standard deviation of the normal probability distribution increases. This implies that high error rates tend to spread out the probability of acquiring the desired error vector across more subweights p, making it harder to find this vector. This second observation is corroborated in the right figure ($\log T$ vs. p) where higher error rates depreciate the importance of selecting the best p, since the time complexity becomes increasingly even across all p (line gradient approaches 0).

5.7.5 ISD Improvements

Several enhancements to the ISD algorithm were proposed mainly to lower the time complexity of finding the error vector. In [71, 50], the Search and Test exhaustive step was replaced with a more efficient Meet-in-the-Middle approach. This approach divides $\mathbf{Q}$ into two $l \times (k/2)$ sub-matrices $\mathbf{Q}_1$ and $\mathbf{Q}_2$, where $l < (n-k)$. We then find all $p/2$ linear combinations of the columns of $\mathbf{Q}_1$ and save them in list $\mathbf{L}_1$. Similarly, we find all $p/2$ linear combinations of the columns of $\mathbf{Q}_2$ and save them in list $\mathbf{L}_2$.

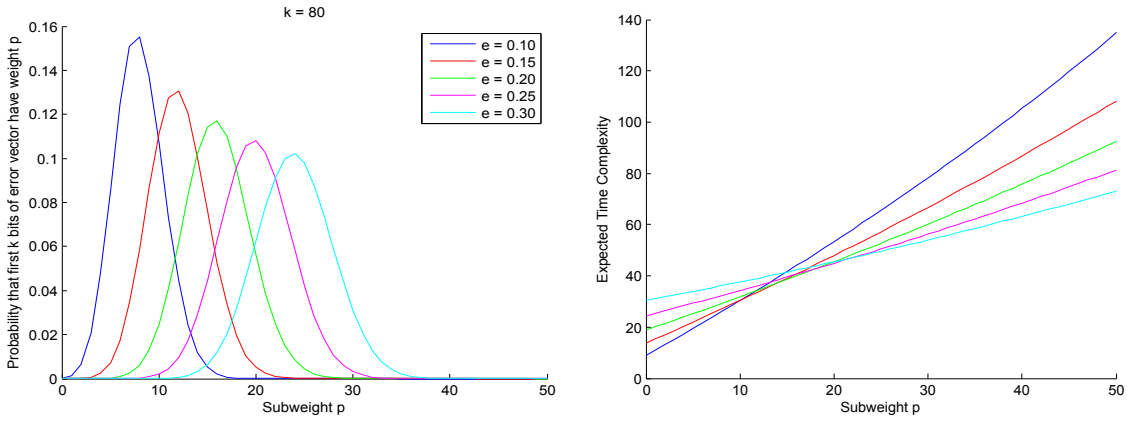

Figure 5.4: For $k = 80$ and different error rates e, the left figure illustrates the probability distribution of $1/\mathcal{P}$ over the subweight p. The right figure shows how $\log T$ is affected by p for $k = 80$ and various error rates e

Next, we search for any two elements $\mathbf{x} \in \mathbf{L}_1, \mathbf{y} \in \mathbf{L}_2$ such that $\mathbf{x} = \mathbf{y} \oplus \mathbf{t}'_l$, where $\mathbf{t}'_l$ is the first l bits of $\mathbf{t}'$. These pairs of elements specify the p chosen columns whose linear combination exactly match the first l bits of $\mathbf{t}'$. This matching process is called collision decoding. For each such pair $(\mathbf{x}, \mathbf{y})$, we test if their full-sized constituents (the columns that were used to create $\mathbf{x}$ and $\mathbf{y}$) add up to form a resultant vector that has a Hamming distance to $\mathbf{t}'$ of exactly $(w - p)$. Finally, if such a pair exists, then we simply add the appropriate unit vectors in $\mathbf{I}_{n-k}$ to the resultant vector to obtain an exact match to $\mathbf{t}'$, as seen previously in Algorithm 5.7.1.

An improvement over the Meet-in-the-Middle approach was proposed by Bernstein et al. in [16], where the ball collision decoding technique is used as an alternative to the collision decoding technique. Allowing for approximate matching between elements in the lists $\mathbf{L}_1$ and $\mathbf{L}_2$, we can find a solution in less time and memory accesses.

Another method, proved to be asymptotically better than the previous techniques, uses the subset sum representation approach described in [44] to solve the column match problem [55], which is equivalent to finding the set of p columns that equal $\mathbf{t}'$ in the first l bits. The method was further improved in [11] by increas-

ing the number of column representations, thus making the task of finding a linear combination that is equal to $\mathbf{t}'$ significantly faster.

5.8 Hybrid BKW Algorithm

In this section, we suggest a modular approach to improving the BKW algorithm by fusing it with some of the other algorithms discussed previously in this chapter. The basic structure and algorithm remains the same. However, we strive to minimize wasting many examples during each estimation loop of BKW by integrating more efficient procedures for hypothesizing the vector $\mathbf{s}$.

The structure of the hybrid procedure is shown in Algorithm 5.8.1. The main difference between this algorithm and the classic BKW is that the hybrid algorithm replaces the $estimate()$ function with an abstract placeholder, $estimateHybrid()$, for other functions to take its place. We discuss some of these functions in the following sub-sections. Furthermore, we introduce the fusion threshold g, which determines the maximum number of BKW linear combination loops allowed before the estimation procedure begins, and is set according to the given k and error rate η. Hence, instead of running the combination loop $(a-1)$ times to estimate a group of b bits, we run the combination loop $g \leq (a-1)$ to estimate a multiple of b-bit groups (specifically $k-gb$ bits). Note that if $g = (a-1)$ then this reduces to the normal BKW procedure.

In addition to the normal LPN parameters, the algorithm accepts a, b, q, and m as parameters to ensure that it is as flexible as possible for solving the problem. The variables a and b represent the number of groups each example is divided into and the size of each group (int bits), respectively. The variable q denotes the number of repetitions for the estimation loop (number of estimates), and m indicates the number of examples that must be selected for each estimation loop (to be reduced by the combination loop and then used to estimate part of $\mathbf{s}$).

We can interpret the time and query complexity more generally by analyzing each

individual sub-procedure to determine how they affect the overall performance. We denote the running time of each combination loop (which includes the *categorize*() and *merge*() functions) as T_c, and the running time of each estimation procedure, *estimateHybrid*(), as T_e. Furthermore, the number of target loops decreases from a to $T_a = \dfrac{a}{a-g}$ since, if $g > (a-1)$, the number of groups that are estimated at once increases.

Thus, using the same analysis that was used in Section 5.2, we can write the generic total time T and query Q complexities, and the memory utilization M as follows:

$$\begin{aligned} T &= O\left(T_a \times q \times (T_c + T_e)\right) \\ Q &= O\left(q \times T_c\right) \\ M &= O\left(T_c\right) \end{aligned} \tag{5.17}$$

5.8.1 BKW-S Hybrid

One of the main drawbacks of the BKW algorithm is that a lot of examples are unused and discarded after the combination stage since only vectors with weight 1 are of interest. As a more general form of LF1, which was proposed in [51], the algorithm discussed in this section implements the *estimateHybrid*() function such that it performs the Fast Walsh Transform (FWT) (see Section 4.3.1) on the entire set of examples leftover from the combination stage to estimate the targeted bits.

For simplicity, we will assume for now that $g = (a-1)$, so the process is the same as the BKW algorithm except in the last stage, we estimate the bits using the FWT. Within the *estimateHybrid*() function, we set up the truth table L containing the noisy labels for each of the possible 2^b inputs. For each example $\mathbf{a}_i$ from $\mathbf{A}'$, we set $L[\mathbf{a}_i] = z_i$. If there exists multiple equal examples but with different labels, we store the label that occurs the most number of times. Once we construct L, we

Algorithm 5.8.1 Hybrid BKW algorithm

```
1: procedure HYBRID BKW(A, z, η, a, b, q, m)
2:
3:     Set g = f(k, η)                                   ▷ Set fusion threshold
4:     α = 1
5:     while α ≤ a do                                    ▷ Target Loop
6:
7:         Initialize votes array v
8:         for γ = 1 → q do
9:             estimated ← false
10:            while estimated is false do               ▷ Estimation Loop
11:                A′ ← subset(A, z, m)
12:                c ← a
13:                repeat                                ▷ Combination Loop
14:                    if c ∉ [α, α + g − 1] then
15:                        {Q_j}_{j=0}^{2^b−1} ← categorize(A′, 1 + b(c − 1), bc)  ▷ see Alg. B.0.1
16:                        for j = 0 to 2^b − 1 do
17:                            Q_j ← merge(Q_j)          ▷ see Alg. B.0.2
18:                        end for
19:                        A′ ← ⋃_{j=0}^{2^b−1} Q_j
20:                    end if
21:                    c ← c − 1
22:                until c = 0
23:                (v_0, v_1) ← estimateHybrid(A′, α, b, g)
24:                if (v_0(i) + v_1(i)) ≥ γ ∀ i then
25:                    estimated ← true                  ▷ All votes accounted for
26:                end if
27:            end while
28:        end for
29:
30:        for i = 1 to (a − g) do                       ▷ Hypothesize partial x
31:            x_α = hypothesize(v_0, v_1, α)
32:            α ← α + 1
33:        end for
34:    end while
35:
36:    x ← (x_1||...||x_a)                               ▷ Reconstruct x
37:    return x
38: end procedure
```

submit it to the FWT procedure, which produces the most likely representation of the function (i.e. the b bits of $\mathbf{x}$ that were compatible with the highest number of examples). If we have m examples, and $b < \log_2 m$, then running the transform takes time $O(2^b \log_2 m)$. As an example, Figure 5.5 illustrates the result of the transform for a function of 16-bit inputs. There are 2^{16} possibilities for the 16-bit input, and the highest 'spike' above the acceptable threshold gives us the most probable one.

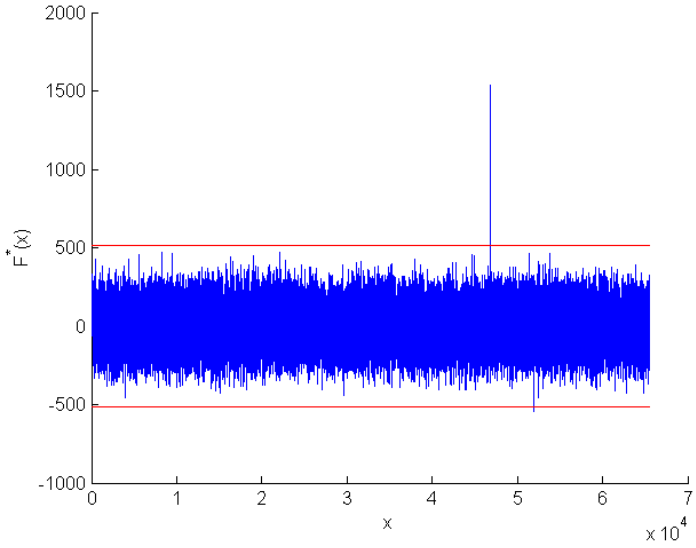

Figure 5.5: The Fast Walsh Transform is computed for a given function with 16-bit inputs. The highest absolute value of the transform above (or below) the threshold (the red lines) gives the most likely function representation.

Incorporating the fusion threshold g, we set $q = 1$ (only need one estimate) and instead use $m = g2^b + (a-g)b(1-2\eta)^{-2^{g+1}}$. This will provide us with an average of $(a-g)b(1-2\eta)^{-2^{g+1}}$ parity equations with which to build the truth table for the FWT. Thus $T_c = O(g2^b + (a-g)b(1-2\eta)^{-2^{g+1}})$, and $T_e = 2^{(a-g)b}\log_2(a-g)b(1-2\eta)^{-2^{g+1}}$. The complexity of T_e represents the running time of using the Fast Walsh Transform (see Section 4.3.1) to estimate $(a-g)b$ bits over $(a-g)b(1-2\eta)^{-2^{g+1}}$ equations.

Figure 5.6 shows how the experimental time complexity differs with g and b. We notice that, when $k = 40$, reducing the fusion threshold under the same value of b decreases the time complexity for medium to high error rates. However, the rate of reduction deteriorates rapidly with higher k for the same error rate. For example, looking at the expected theoretical graphs of BKW-S for $k = 80$ in Figure 5.7, we notice that reducing g for $b = 20$ *increases* the time. In fact, for larger error rates ($\eta > 0.35$), there seems to be little to no difference between using $b = 40, g = 1$ and $b = 20, g = 2$. Nevertheless, if one would want to choose between one of those two

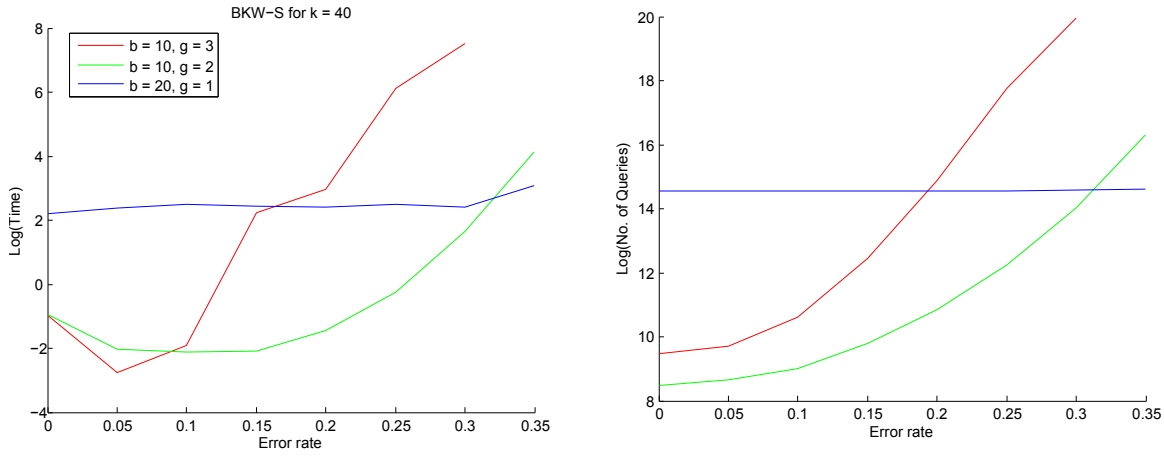

Figure 5.6: The experimental results of using the BKW-S algorithm for $k = 40$. The left figure shows $\log(T)$ vs. η and the right figure shows $\log(Q)$ vs. η for different combinations of b and g

parameter sets for high error rates, it is preferable to select $b = 20, g = 2$ since, as illustrated in the right of Figure 5.7, it utilizes less queries within the same amount of time.

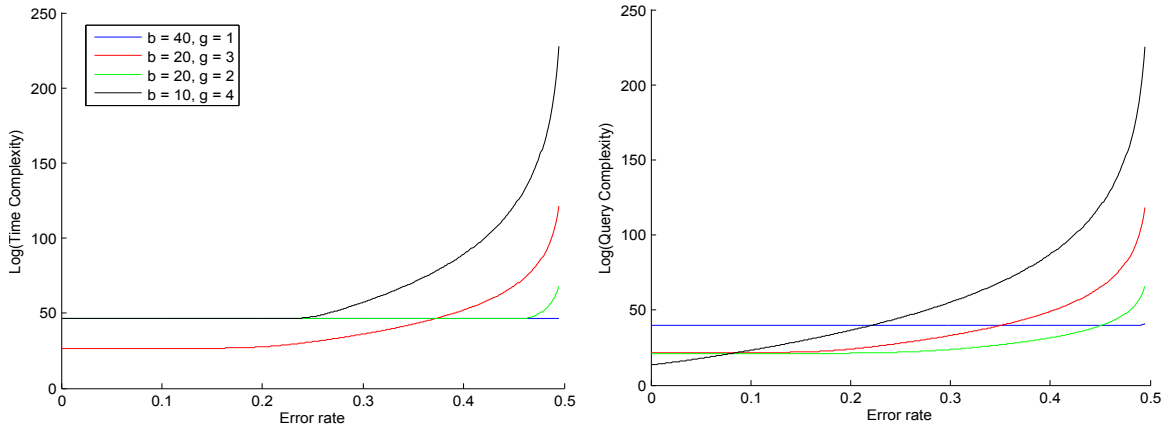

Figure 5.7: The theoretical expected performance of the BKW-S algorithm for $k = 80$. The left figure shows $\log(T)$ vs. η and the right figure shows $\log(Q)$ vs. η for different combinations of b and g

5.8.2 BKW-H Hybrid

Another hybrid procedure integrates the hashing collision algorithm described in Section 5.3 with the BKW algorithm. In particular the function $estimateHybrid()$ is materialized as Algorithm 5.3.1 but for smaller-sized inputs. The difference between Algorithm 5.3.1 and this BKW hybrid is that, in the hybrid, instead of hashing the whole k-bit examples, we only hash the $(a-g)$ bits of the example (those bits that were left after the other bits were reduced to zero during the normal BKW linear combination phase). Other than that, the same idea applies here whereupon examples that are on the verge of being inserted into the hash table are matched with examples already in the table with a Hamming distance of 1. When combined, the resultant examples are of weight 1 and can be used to directly infer an estimate of the corresponding bit of $\mathbf{s}$.

Following the same analysis performed in Section 5.3 and using the hybrid complexity structure (see Equation 5.17), we will set $T_c = O\left(g2^b + (a-g)b2^{(a-g)b/2}\right)$ because we require about $g2^b$ for the linear combination phase of BKW and an expected $(a-g)b2^{(a-g)b/2}$ number of examples to ensure that we get a collision (due to the birthday paradox). Furthermore, since we will be hashing all the available examples, we get $T_e = (a-g)b2^{(a-g)b/2}$. Very much like the original BKW algorithm, we will set the number of estimates $q = poly(b, (1-2\eta)^{-2^{g+2}})$ and take the majority vote for estimating each bit of $\mathbf{s}$.

5.9 Experimental Tests

We present here the experimental results that compare the time and query complexity of some of the discussed algorithms. We decided to implement and test those algorithms that best represented the class of attacks to which they belong.

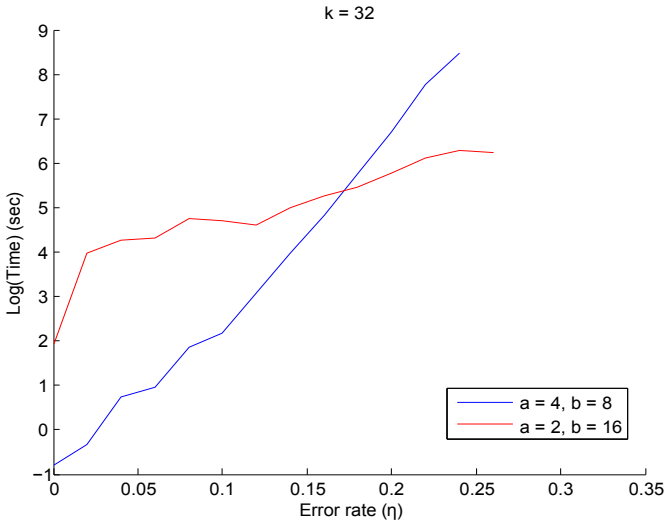

Figure 5.8: Shown for $k = 32$, the parameter a has a noticeable effect on the performance of the BKW algorithm. The higher the error rate, the more preferable it is to decrease a

5.9.1 Setup

The experiments were performed using a 2.4 GHz processor with a 12 MB L3 Cache and 32 GB of physical memory. All tests were conducted using Java on a 1.6 64-bit runtime environment.

5.9.2 BKW Parameters

The graph in Figure 5.8 shows that the optimal values for the internal variables, a and b, of the BKW algorithm are dependant on the error rate for a specific value of k. Therefore, in order to give an efficient concrete instantiation of the BKW algorithm (or its derivatives) one must find the suitable values for these parameters, given the error rate used by the scheme and the size of the security parameter k. The asymptotic values for a and b are given as $(\log k)/2$ and $2k/\log k$, respectively [19].

5.9.3 Results

Figure 5.9 and Figure 5.10 show how the time and query complexity changes as the error rates rises for $k = 32$ and $k = 60$, respectively. The solid lines are the deterministic algorithms, whilst the dashed lines are the probabilistic (Random Selection and Information Set Decoding) algorithms. We use the LF1 [51] variant of BKW-S here.

We notice from the left figures that the RS algorithm performs comparatively well in the low error range ($\eta < 0.2$) but deteriorates quickly to be superseded by the hybrid BKW algorithms. Furthermore, we note that BKW-H seems to be strictly slower than BKW-S, but is still preferable over the normal BKW procedure. While the ISD algorithm is generally slower when compared to the other algorithms (even against the original BKW for high error rates), it appears that its strengths lie in its low query complexity.

The main observation we can deduce from the right figures is that the probabilistic algorithms use much less queries than the deterministic ones. Furthermore, the ISD is able to adapt to use less queries than RS, which uses its upper bound for the number of queries to ensure that the minimum expected time complexity is attained. Additionally, it is interesting to note the BKW-S algorithm becomes saturated with queries at medium-to-high error rates such that the query complexity rate slows down rapidly.

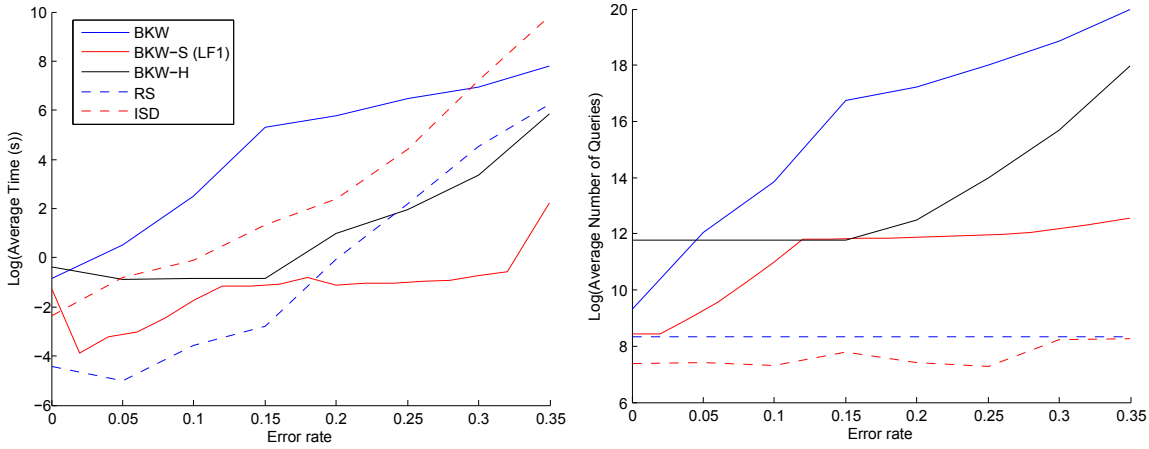

Figure 5.9: For $k = 32$ and using different algorithms, the left figure shows how $\log(T)$ changes with error rate, and the right figure shows how $\log(Q)$ changes with error rate

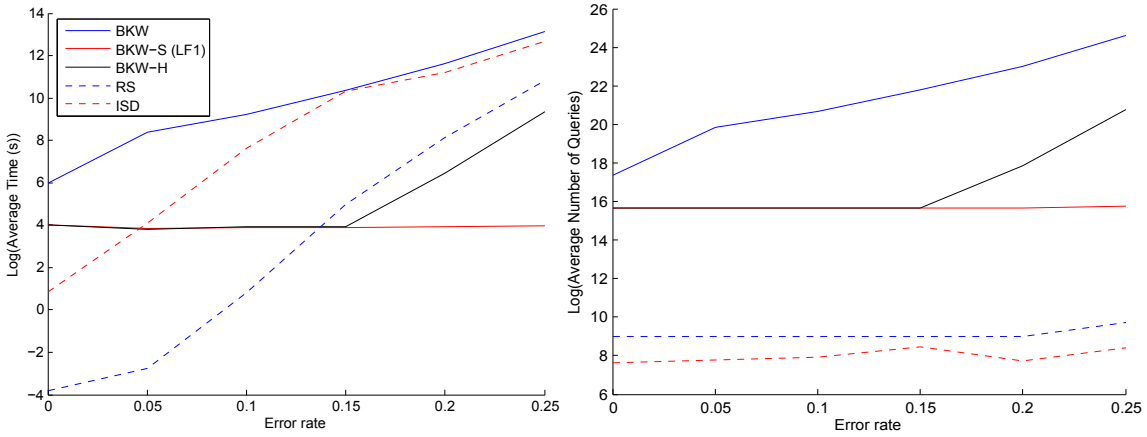

Figure 5.10: For $k = 60$ and using different algorithms, the left figure shows how $\log(T)$ changes with error rate, and the right figure shows how $\log(Q)$ changes with error rate

6

Conclusion

In this thesis, we have briefly discussed the various applications and uses of lattice problems in modern cryptography and presented some schemes that assume the hardness of certain worst-case lattice problems to ensure security and confidentiality of the encrypted data. In particular, we focused on the Learning with Errors (LWE) and Learning Parity with Noise (LPN) problems, which were shown to be flexible enough to be used as underlying problems for cryptographic schemes and primitives. It is, therefore, important to determine how resilient these problems are so that suitable security parameters are selected to counter the efforts of determined adversaries.

Several LPN adversaries were described in detail and analyzed in terms of performance, query complexity, and memory utilization. Furthermore, suggestions for improvements and adaptations were proposed for some of the algorithms, yielding enhancements in one or more of the aforementioned criteria. Each of the different algorithms demonstrated distinctive strengths and drawbacks, which allowed them to be effective against some LPN applications but incompetent against others. For example, a relatively fast algorithm that utilizes large amounts of memory to solve

the LPN problem may not be feasible if this memory is not available. Thus, an adversary with well-defined capabilities and limitations should identify the weaknesses of the targeted LPN protocol and select the best algorithm capable of breaking the scheme as efficiently as possible given the available resources.

We conclude by raising a series of open questions for possible future work. It would be interesting to investigate other variations of adversaries with stronger capabilities. In particular, given more than one oracle (with possibly non-uniform key distributions) to supply the examples, can an adversary's strategies be modified somehow to solve the LPN problem in a more efficient way?

Another venue of exploration that is related to this subject involves studying the possibility of extending LPN algorithms to work on solving the LWE problem, which is used more frequently in cryptographic applications than the LPN problem. While not all extensions appear to be trivial, or even immediately possible, it is nevertheless a natural place to start discovering more effective solutions for the LWE problem.

Bibliography

[1] S. Agrawal, D. Boneh, and X. Boyen, "Efficient lattice (h)ibe in the standard model," in *Proceedings of the 29th Annual international conference on Theory and Applications of Cryptographic Techniques*, ser. EUROCRYPT'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 553–572. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-13190-5_28. [Accessed April 2013].

[2] M. Ajtai, "Generating hard instances of lattice problems (extended abstract)," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, ser. STOC '96. New York, NY, USA: ACM, 1996, pp. 99–108. [Online]. Available: http://doi.acm.org/10.1145/237814.237838. [Accessed August 2012].

[3] ——, "The shortest vector problem in l_2 is np-hard for randomized reductions (extended abstract)," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, ser. STOC '98. New York, NY, USA: ACM, 1998, pp. 10–19. [Online]. Available: http://doi.acm.org/10.1145/276698.276705. [Accessed April 2013].

[4] M. Ajtai and C. Dwork, "A public-key cryptosystem with worst-case/average-case equivalence," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, ser. STOC '97. New York, NY, USA: ACM, 1997, pp. 284–293. [Online]. Available: http://doi.acm.org/10.1145/258533.258604. [Accessed August 2012].

[5] M. Ajtai, R. Kumar, and D. Sivakumar, "A sieve algorithm for the shortest lattice vector problem," in *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, ser. STOC '01. New York, NY, USA: ACM, 2001, pp. 601–610. [Online]. Available: http://doi.acm.org/10.1145/380752.380857. [Accessed November 2012].

[6] A. Akavia, S. Goldwasser, and V. Vaikuntanathan, "Simultaneous hardcore bits and cryptography against memory attacks," in *Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography*, ser. TCC '09.

Berlin, Heidelberg: Springer-Verlag, 2009, pp. 474–495. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-00457-5_28. [Accessed April 2013].

[7] B. Applebaum, D. Cash, C. Peikert, and A. Sahai, "Fast cryptographic primitives and circular-secure encryption based on hard learning problems," in *Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 595–618. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-03356-8_35. [Accessed March 2013].

[8] L. Babai, "On lovasz' lattice reduction and the nearest lattice point problem (shortened version)," in *Proceedings of the 2nd Symposium of Theoretical Aspects of Computer Science*, ser. STACS '85. London, UK, UK: Springer-Verlag, 1985, pp. 13–20. [Online]. Available: http://dl.acm.org/citation.cfm?id=646502.696106. [Accessed March 2013].

[9] A. Banerjee, C. Peikert, and A. Rosen, "Pseudorandom functions and lattices," in *Proceedings of the 31st Annual international conference on Theory and Applications of Cryptographic Techniques*, ser. EUROCRYPT'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 719–737. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-29011-4_42. [Accessed April 2013].

[10] E. Barker, A. Roginsky, N. I. of Standards, and T. (U.S.), *Recommendation for Cryptographic Key Generation*, ser. NIST special publication. U.S. Department of Commerce, National Institute of Standards and Technology, 2012.

[11] 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," in *Proceedings of the 31st Annual international conference on Theory and Applications of Cryptographic Techniques*, ser. EUROCRYPT'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 520–536. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-29011-4_31. [Accessed January 2013].

[12] M. Bellare and P. Rogaway, "Random oracles are practical: a paradigm for designing efficient protocols," in *Proceedings of the 1st ACM conference on Computer and communications security*, ser. CCS '93. New York, NY, USA: ACM, 1993, pp. 62–73. [Online]. Available: http://doi.acm.org/10.1145/168588.168596. [Accessed March 2013].

[13] ——, "Introduction to modern cryptography," in *UCSD CSE 207 Course Notes*, 2005. [Online]. Available: http://cseweb.ucsd.edu/~mihir/cse207/classnotes.html. [Accessed April 2013].

[14] E. Berlekamp, R. McEliece, and H. van Tilborg, "On the inherent intractability of certain coding problems (corresp.)," *IEEE Trans. Inf. Theor.*, vol. 24, no. 3, pp. 384–386, Sep. 2006. [Online]. Available: http://dx.doi.org/10.1109/TIT.1978.1055873. [Accessed February 2013].

[15] D. J. Bernstein, J. Buchmann, and E. Dahmen, *Post Quantum Cryptography*, 1st ed. Springer Publishing Company, Incorporated, 2008.

[16] D. J. Bernstein, T. Lange, and C. Peters, "Smaller decoding exponents: ball-collision decoding," in *Proceedings of the 31st annual conference on Advances in cryptology*, ser. CRYPTO'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 743–760. [Online]. Available: http://dl.acm.org/citation.cfm?id=2033036.2033092. [Accessed January 2013].

[17] J. Blömer and J.-P. Seifert, "On the complexity of computing short linearly independent vectors and short bases in a lattice," in *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, ser. STOC '99. New York, NY, USA: ACM, 1999, pp. 711–720. [Online]. Available: http://doi.acm.org/10.1145/301250.301441. [Accessed April 2013].

[18] A. Blum, M. L. Furst, M. J. Kearns, and R. J. Lipton, "Cryptographic primitives based on hard learning problems," in *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '93. London, UK: Springer-Verlag, 1994, pp. 278–291. [Online]. Available: http://dl.acm.org/citation.cfm?id=646758.759585. [Accessed January 2012].

[19] A. Blum, A. Kalai, and H. Wasserman, "Noise-tolerant learning, the parity problem, and the statistical query model," *J. ACM*, vol. 50, no. 4, pp. 506–519, Jul. 2003. [Online]. Available: http://doi.acm.org/10.1145/792538.792543. [Accessed January 2012].

[20] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical gapsvp," in *Advances in Cryptology - Crypto 2012*, ser. Lecture Notes in Computer Science, vol. 7417. Springer, 2012, pp. 868–886. [Online]. Available: http://eprint.iacr.org/2012/078.pdf. [Accessed April 2013].

[21] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," pp. 309–325, 2012. [Online]. Available: http://doi.acm.org/10.1145/2090236.2090262. [Accessed April 2013].

[22] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) lwe," in *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, ser. FOCS '11. Washington,

DC, USA: IEEE Computer Society, 2011, pp. 97–106. [Online]. Available: http://dx.doi.org/10.1109/FOCS.2011.12. [Accessed October 2012].

[23] J. Bringer and H. Chabanne, "Trusted-hb: A low-cost version of hb+ secure against man-in-the-middle attacks," *IEEE Trans. Inf. Theor.*, vol. 54, no. 9, pp. 4339–4342, Sep. 2008. [Online]. Available: http://dx.doi.org/10.1109/TIT.2008.928290. [Accessed December 2012].

[24] J. Carrijo, R. Tonicelli, H. Imai, and A. C. A. Nascimento, "A novel probabilistic passive attack on the protocols hb and hb+," *IACR Cryptology ePrint Archive*, vol. 2008, p. 231, 2008. [Online]. Available: http://eprint.iacr.org/2008/231.pdf. [Accessed January 2013].

[25] P. Chose, A. Joux, and M. Mitton, "Fast correlation attacks: An algorithmic point of view," in *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology*, ser. EUROCRYPT '02. London, UK: Springer-Verlag, 2002, pp. 209–221. [Online]. Available: http://dl.acm.org/citation.cfm?id=647087.715845. [Accessed December 2012].

[26] D. Coppersmith, "Small solutions to polynomial equations, and low exponent rsa vulnerabilities," *Journal of Cryptology*, vol. 10, no. 4, pp. 233–260, 1997. [Online]. Available: http://dx.doi.org/10.1007/s001459900030. [Accessed March 2013].

[27] J. Ding, "New cryptographic constructions using generalized learning with errors problem," Cryptology ePrint Archive, Report 2012/387, 2012, http://eprint.iacr.org/, [Accessed April 2013].

[28] I. Dinur, G. Kindler, and S. Safra, "Approximating-cvp to within almost-polynomial factors is np-hard," in *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, ser. FOCS '98. Washington, DC, USA: IEEE Computer Society, 1998, p. 99. [Online]. Available: http://dl.acm.org/citation.cfm?id=795664.796466. [Accessed April 2013].

[29] J.-B. Fischer and J. Stern, "An efficient pseudo-random generator provably as secure as syndrome decoding," in *Proceedings of the 15th annual international conference on Theory and application of cryptographic techniques*, ser. EUROCRYPT'96. Berlin, Heidelberg: Springer-Verlag, 1996, pp. 245–255. [Online]. Available: http://dl.acm.org/citation.cfm?id=1754495.1754526. [Accessed December 2012].

[30] M. P. Fossorier, M. J. Mihaljevic, H. Imai, Y. Cui, and K. Matsuura, "A novel algorithm for solving the lpn problem and its application to security evaluation of the hb protocol for rfid authentication," 2006. [Online]. Available: http://eprint.iacr.org/2006/197. [Accessed January 2013].

[31] N. Gama and P. Q. Nguyen, "Predicting lattice reduction," in *Proceedings of the theory and applications of cryptographic techniques 27th annual international conference on Advances in cryptology*, ser. EUROCRYPT'08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 31–51. [Online]. Available: http://dl.acm.org/citation.cfm?id=1788414.1788417. [Accessed March 2013].

[32] C. F. Gauss, *Disquisitiones Arithmeticae*. New York: Springer-Verlag, 1986.

[33] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the 41st annual ACM symposium on Theory of computing*, ser. STOC '09. New York, NY, USA: ACM, 2009, pp. 169–178. [Online]. Available: http://doi.acm.org/10.1145/1536414.1536440. [Accessed April 2012].

[34] C. Gentry, C. Peikert, and V. Vaikuntanathan, "Trapdoors for hard lattices and new cryptographic constructions," in *Proceedings of the 40th annual ACM symposium on Theory of computing*, ser. STOC '08. New York, NY, USA: ACM, 2008, pp. 197–206. [Online]. Available: http://doi.acm.org/10.1145/1374376.1374407. [Accessed January 2013].

[35] H. Gilbert, M. J. Robshaw, and Y. Seurin, "How to encrypt with the lpn problem," in *Proceedings of the 35th international colloquium on Automata, Languages and Programming, Part II*, ser. ICALP '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 679–690. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-70583-3_55. [Accessed December 2012].

[36] H. Gilbert, M. J. B. Robshaw, and Y. Seurin, "Hb#: increasing the security and efficiency of hb+," in *Proceedings of the theory and applications of cryptographic techniques 27th annual international conference on Advances in cryptology*, ser. EUROCRYPT'08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 361–378. [Online]. Available: http://dl.acm.org/citation.cfm?id=1788414.1788435. [Accessed December 2012].

[37] H. Gilbert, M. J. B. Robshaw, and H. Sibert, "An active attack against hb+ - a provably secure lightweight authentication protocol," *IACR Cryptology ePrint Archive*, vol. 2005, p. 237, 2005. [Online]. Available: http://eprint.iacr.org/2005/237.pdf. [Accessed December 2012].

[38] O. Goldreich, D. Micciancio, S. Safra, and J. P. Seifert, "Approximating shortest lattice vectors is not harder than approximating closet lattice vectors," *Inf. Process. Lett.*, vol. 71, no. 2, pp. 55–61, Jul. 1999. [Online]. Available: http://dx.doi.org/10.1016/S0020-0190(99)00083-6. [Accessed April 2013].

[39] O. Goldreich, S. Goldwasser, and S. Halevi, "Eliminating decryption errors in the ajtai-dwork cryptosystem," in *Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '97. London, UK: Springer-Verlag, 1997, pp. 105–111. [Online]. Available: http://dl.acm.org/citation.cfm?id=646762.706188. [Accessed December 2012].

[40] ——, "Public-key cryptosystems from lattice reduction problems," in *Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '97. London, UK: Springer-Verlag, 1997, pp. 112–131. [Online]. Available: http://dl.acm.org/citation.cfm?id=646762.706185. [Accessed April 2013].

[41] S. Goldwasser, Y. T. Kalai, C. Peikert, and V. Vaikuntanathan, "Robustness of the learning with errors assumption," in *ICS'10*, 2010, pp. 230–240. [Online]. Available: http://www.cs.toronto.edu/~vinodv/robustlwe.pdf. [Accessed March 2013].

[42] M. Henk, "Note on shortest and nearest lattice vectors," *Inf. Process. Lett.*, vol. 61, no. 4, pp. 183–188, Feb. 1997. [Online]. Available: http://dx.doi.org/10.1016/S0020-0190(97)00019-7. [Accessed April 2013].

[43] N. J. Hopper and M. Blum, "Secure human identification protocols," in *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ser. ASIACRYPT '01. London, UK: Springer-Verlag, 2001, pp. 52–66. [Online]. Available: http://dl.acm.org/citation.cfm?id=647097.717000. [Accessed September 2012].

[44] N. Howgrave-Graham and A. Joux, "New generic algorithms for hard knapsacks," in *Proceedings of the 29th Annual international conference on Theory and Applications of Cryptographic Techniques*, ser. EUROCRYPT'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 235–256. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-13190-5_12. [Accessed January 2013].

[45] A. Joux, *Algorithmic Cryptanalysis*, 1st ed. Chapman & Hall/CRC, 2009.

[46] A. Juels and S. A. Weis, "Authenticating pervasive devices with human protocols," in *Proceedings of the 25th annual international conference on Advances in Cryptology*, ser. CRYPTO'05. Berlin, Heidelberg: Springer-Verlag,

2005, pp. 293–308. [Online]. Available: http://dx.doi.org/10.1007/11535218_18. [Accessed September 2012].

[47] J. Katz and J. S. Shin, "Parallel and concurrent security of the hb and hb+ protocols," in *Proceedings of the 24th annual international conference on The Theory and Applications of Cryptographic Techniques*, ser. EUROCRYPT'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 73–87. [Online]. Available: http://dx.doi.org/10.1007/11761679_6. [Accessed December 2012].

[48] S. Khot, "Hardness of approximating the shortest vector problem in lattices," *J. ACM*, vol. 52, no. 5, pp. 789–808, Sep. 2005. [Online]. Available: http://doi.acm.org/10.1145/1089023.1089027. [Accessed April 2013].

[49] A. Lenstra, J. Lenstra, H.W., and L. Lovasz, "Factoring polynomials with rational coefficients," *Mathematische Annalen*, vol. 261, no. 4, pp. 515–534, 1982. [Online]. Available: http://dx.doi.org/10.1007/BF01457454. [Accessed March 2013].

[50] J. S. Leon, "A probabilistic algorithm for computing minimum weights of large error-correcting codes," *IEEE Trans. Inf. Theor.*, vol. 34, no. 5, pp. 1354–1359, Sep. 2006. [Online]. Available: http://dx.doi.org/10.1109/18.21270. [Accessed January 2013].

[51] E. Levieil and P.-A. Fouque, "An improved lpn algorithm," in *Proceedings of the 5th international conference on Security and Cryptography for Networks*, ser. SCN'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 348–359. [Online]. Available: http://dx.doi.org/10.1007/11832072_24. [Accessed January 2013].

[52] Y.-K. Liu, V. Lyubashevsky, and D. Micciancio, "On bounded distance decoding for general lattices," in *Proceedings of the 9th international conference on Approximation Algorithms for Combinatorial Optimization Problems, and 10th international conference on Randomization and Computation*, ser. APPROX'06/RANDOM'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 450–461. [Online]. Available: http://dx.doi.org/10.1007/11830924_41. [Accessed April 2013].

[53] V. Lyubashevsky and D. Micciancio, "On bounded distance decoding, unique shortest vectors, and the minimum distance problem," in *Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 577–594. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-03356-8_34. [Accessed April 2013].

[54] M. J. M. Marc P. C. Fossorier and H. Imai, "A unified analysis on block decoding approaches for the fast correlation attack," in *IEEE Int. Symp. Inform. Theory - ISIT*, 2005, pp. 2012 – 2015.

[55] A. May, A. Meurer, and E. Thomae, "Decoding random linear codes in $O(2^{0.054n})$," in *Proceedings of the 17th international conference on The Theory and Application of Cryptology and Information Security*, ser. ASIACRYPT'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 107–124. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-25385-0_6. [Accessed January 2013].

[56] R. J. Mceliece, "A public-key cryptosystem based on algebraic coding theory," Jet Propulsion Lab Deep Space Network Progress report, Tech. Rep., 1978.

[57] D. Micciancio, "The shortest vector in a lattice is hard to approximate to within some constant," in *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, ser. FOCS '98. Washington, DC, USA: IEEE Computer Society, 1998, p. 92. [Online]. Available: http://dl.acm.org/citation.cfm?id=795664.796467. [Accessed April 2013].

[58] D. Micciancio and S. Goldwasser, *Complexity of Lattice Problems: a cryptographic perspective*, ser. The Kluwer International Series in Engineering and Computer Science. Boston, Massachusetts: Kluwer Academic Publishers, Mar. 2002, vol. 671.

[59] D. Micciancio and O. Regev, "Worst-case to average-case reductions based on gaussian measures," *SIAM J. Comput.*, vol. 37, no. 1, pp. 267–302, Apr. 2007. [Online]. Available: http://dx.doi.org/10.1137/S0097539705447360. [Accessed April 2013].

[60] M. J. Mihaljevic, M. P. C. Fossorier, and H. Imai, "Fast correlation attack algorithm with list decoding and an application," in *Revised Papers from the 8th International Workshop on Fast Software Encryption*, ser. FSE '01. London, UK: Springer-Verlag, 2002, pp. 196–210. [Online]. Available: http://dl.acm.org/citation.cfm?id=647936.741079. [Accessed January 2013].

[61] M. J. Mihaljević, M. P. C. Fossorier, and H. Imai, "Cryptanalysis of keystream generator by decimated sample based algebraic and fast correlation attacks," in *Proceedings of the 6th international conference on Cryptology in India*, ser. INDOCRYPT'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 155–168. [Online]. Available: http://dx.doi.org/10.1007/11596219_13. [Accessed January 2013].

[62] ——, "A general formulation of algebraic and fast correlation attacks based on dedicated sample decimation," in *Proceedings of the 16th international conference on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, ser. AAECC'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 203–214. [Online]. Available: http://dx.doi.org/10.1007/11617983_20. [Accessed January 2013].

[63] P. Q. Nguyen, "Cryptanalysis of the goldreich-goldwasser-halevi cryptosystem from crypto '97," London, UK, pp. 288–304, 1999. [Online]. Available: http://dl.acm.org/citation.cfm?id=646764.703978. [Accessed December 2012].

[64] P. Q. Nguyen and J. Stern, "Cryptanalysis of the ajtai-dwork cryptosystem," in *Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '98. London, UK: Springer-Verlag, 1998, pp. 223–242. [Online]. Available: http://dl.acm.org/citation.cfm?id=646763.706326. [Accessed December 2012].

[65] C. Peikert, "Public-key cryptosystems from the worst-case shortest vector problem: extended abstract," in *Proceedings of the 41st annual ACM symposium on Theory of computing*, ser. STOC '09. New York, NY, USA: ACM, 2009, pp. 333–342. [Online]. Available: http://doi.acm.org/10.1145/1536414.1536461. [Accessed June 2012].

[66] C. Peikert, V. Vaikuntanathan, and B. Waters, "A framework for efficient and composable oblivious transfer," in *Proceedings of the 28th Annual conference on Cryptology: Advances in Cryptology*, ser. CRYPTO 2008. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 554–571. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-85174-5_31. [Accessed April 2012].

[67] C. Peikert and B. Waters, "Lossy trapdoor functions and their applications," in *Proceedings of the 40th annual ACM symposium on Theory of computing*, ser. STOC '08. New York, NY, USA: ACM, 2008, pp. 187–196. [Online]. Available: http://doi.acm.org/10.1145/1374376.1374406. [Accessed April 2012].

[68] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," in *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, ser. STOC '05. New York, NY, USA: ACM, 2005, pp. 84–93. [Online]. Available: http://doi.acm.org/10.1145/1060590.1060603. [Accessed June 2012].

[69] A. Shamir, "A polynomial time algorithm for breaking the basic merkle-hellman cryptosystem," in *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, ser. SFCS '82. Washington, DC,

USA: IEEE Computer Society, 1982, pp. 145–152. [Online]. Available: http://dx.doi.org/10.1109/SFCS.1982.55. [Accessed March 2013].

[70] D. Simon, "Selected applications of lll in number theory," in *The LLL Algorithm: Survey and Applications*, ser. LLL+25, 2007, pp. 264–281. [Online]. Available: http://www.math.unicaen.fr/~simon/maths/lll25.html. [Accessed March 2013].

[71] J. Stern, "A method for finding codewords of small weight," in *Proceedings of the 3rd International Colloquium on Coding Theory and Applications*. London, UK: Springer-Verlag, 1989, pp. 106–113. [Online]. Available: http://dl.acm.org/citation.cfm?id=646721.702702. [Accessed January 2013].

[72] J. van de Pol, *Lattice-based Cryptography*, The Netherlands, 2011.

[73] P. van Emde Boas, "Another np-complete partition problem and the complexity of computing short vectors in a lattice," in *Technical Report 81-04*, Mathematische Instituut, University of Amsterdam, 1981, p. 10. [Online]. Available: http://staff.science.uva.nl/~peter/vectors/mi8104c.html. [Accessed April 2013].

[74] D. Wagner, "A generalized birthday problem," in *Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '02. London, UK: Springer-Verlag, 2002, pp. 288–303. [Online]. Available: http://dl.acm.org/citation.cfm?id=646767.704294. [Accessed February 2013].

[75] U. Wagner, "Solving the LPN problem in cube-root time," *CoRR*, vol. abs/1201.4725, 2012. [Online]. Available: http://arxiv.org/pdf/1201.4725.pdf. [Accessed January 2013].

Appendix A

Theorems

Minkowski's Theorem

This version of the theorem's definition is adopted from [58].

Theorem A.0.1 (Minkowski's Theorem)**.** *Given any n-dimensional full-rank lattice* Λ *and a convex set* $S \subset span(\Lambda)$ *that is symmetric about the origin, if the volume of* S *is greater than* $2^n det(\Lambda)$*, then* S *contains a non-zero lattice point* $x \in \Lambda$*.*

Boole's Inequality

Theorem A.0.2 (Boole's Inequality)**.** *Let* $E_1, ..., E_n$ *be events over some sample space* S*. Then the following inequality holds:*

$$Pr[E_1 \vee ... \vee E_n] \leq \sum_{i=1}^{n} Pr[E_i] \tag{A.1}$$

Hoeffding's Inequality

We show here the special case of the theorem for Bernoulli random variables.

Theorem A.0.3 (Hoeffding's Inequality)**.** *Let* $X_1, ..., X_n$ *be n independent, identically distributed random variables where, for all i,* $X_i \sim Ber(p)$*. Let* $X = \sum_{i=1}^{n} X_i$

and $E[X] = \mu = np$. *Then, for any* $\epsilon > 0$, *the following inequalities hold:*

$$Pr[X - \mu > n\epsilon] \leq 2\exp(-2n\epsilon^2) \tag{A.2}$$

$$Pr[X - \mu < -n\epsilon] \leq 2\exp(-2n\epsilon^2) \tag{A.3}$$

Appendix B

Sub-Procedures

Categorizing Examples

Algorithm B.0.1 describes the procedure used to categorize the $(k+1)$-bit labeled examples (columns) in a given set $\mathbf{A} \in \mathbb{Z}_2^{(k+1)\times n}$ into separate partitions based on bits $(b, b+1, ..., c)$ of each example. The output, $\{\mathbf{Q}_j\}_{j=1}^{t}$, is a set of $t = 2^{(c-b+1)}$ disjoint partitions, where each partition has examples with identical segments of bits in positions $[b, c]$. The expected size of each partition is n/t. Assuming that $n >> k$, the time complexity is $O(n)$.

Algorithm B.0.1 Categorize Sub-Procedure

```
1: procedure CATEGORIZE(A, b, c)
2:     t = 2^(c−b+1)
3:     Q_j = {} ∀ j ∈ [1, t]
4:     for each a ∈ A do
5:         p ← (a_c||a_{c−1}||...||a_b)
6:         Q_p = Q_p ∪ a
7:     end for
8:     return {Q_j}_{j=1}^t
9: end procedure
```

Merging Examples

Algorithm B.0.2 describes the sub-procedure used by the BKW algorithm (see Algorithm 5.2.1) to add (mod 2) one randomly selected $(k+1)$-bit example from the set $\mathbf{A}$ of size n to all the other $(k+1)$-bit labeled examples in the set. The size of the $\mathbf{A}$ at the end is $n-1$ since we discard the victim example. Assuming that $n >> k$, the time needed to perform this procedure is $O(n)$.

Algorithm B.0.2 Merge Sub-Procedure

```
1: procedure MERGE(A)
2:     a_r ← randomSelect(A)
3:     for each a ∈ A\a_r do
4:         a = a ⊕ a_r
5:     end for
6:     A ← A − a_r
7:     return A
8: end procedure
```

Bit Estimation

Algorithm B.0.3 describes the sub-procedure used by the BKW algorithm (see Algorithm 5.2.1) to aggregate the estimates of each bit of the key by using the unit vectors present in the supplied set $\mathbf{A}$ of n examples. Assuming that $n >> k$, the time needed to perform this procedure is $O(n)$.

Algorithm B.0.3 Estimation Sub-Procedure

1: **procedure** ESTIMATE($\mathbf{A}, \alpha, b$)
2: $(k, n) = Dim(\mathbf{A})$
3: $\mathbf{v}_0 i \; \forall \; i \in [1, b]$
4: $\mathbf{v}_1 i \; \forall \; i \in [1, b]$
5: **for** each $(\mathbf{a}, z) \in \mathbf{A}$ **do**
6: **if** $\text{wt}(\mathbf{a}) = 1$ **then**
7: $j \leftarrow findIndexOfOne(\mathbf{a})$
8: $j = (\alpha - 1)b - j$
9: $\mathbf{v}_z(j) \leftarrow \mathbf{v}_z(j) + 1$
10: **end if**
11: **end for**
12: **return** $(\mathbf{v}_0, \mathbf{v}_1)$
13: **end procedure**