Some Notes on Post-Quantum Cryptanalysis

Mårtensson, Erik

2020

Document Version:
Publisher's PDF, also known as Version of record

Some Notes on Post-Quantum Cryptanalysis

Doctoral Dissertation

Erik Mårtensson

LUND UNIVERSITY

Department of Electrical and Information Technology
Lund University, Lund, Sweden
December, 2020

Department of Electrical and Information Technology
Lund University
Box 118, SE-221 00 LUND
SWEDEN

This thesis is set in Computer Modern 10pt
with the LaTeX Documentation System

Series of licentiate and doctoral theses
ISSN 1654-790X; No. 135
ISBN: 978-91-7895-711-8 (print)
ISBN: 978-91-7895-712-5 (pdf)

Printed in Sweden by *Tryckeriet i E-huset*, Lund.
December 2020.

...To my father...

Abstract

Cryptography as it is used today relies on a foundational level on the assumption that either the Integer Factoring Problem (IFP) or the Discrete Logarithm Problem (DLP) is computationally intractable. In the 1990s Peter Shor developed a quantum algorithm that solves both problems in polynomial time. Since then alternative foundational mathematical problems to replace IFP and DLP have been suggested. This area of research is called post-quantum cryptology.

To remedy the threat of quantum computers the National Institute of Standards and Technology (NIST) has organized a competition to develop schemes for post-quantum encryption and digital signatures. For both categories lattice-based cryptography candidates dominate. The second most promising type of candidate for encryption is code-based cryptography.

The lattice-based candidates are based on the difficulty of either the Learning With Errors problem (LWE) or the N^{th} Degree Truncated Polynomial problem (NTRU), of which LWE is the focus of this thesis. The difficulty of both these problems in turn relies on the difficulty of variations of the Shortest Vector Problem (SVP). Code-based cryptography is based on the difficulty of decoding random linear codes.

The main focus of this thesis is on solving the LWE problem using the Blum-Kalai-Wasserman algorithm (BKW). We have the following improvements of the algorithm.

1. We combined BKW with state-of-the-art lattice sieving methods to improve the complexity of the algorithm. We also elaborate on the similarities and differences between BKW and lattice sieving, two approaches that on a shallow level look very different.

2. We developed a new binary approach for the distinguishing phase of the BKW algorithm and showed that it performs favorably compared to previous distinguishers.

3. We investigated the Fast Fourier Transform (FFT) approach for the distinguishing part of BKW showing that it performs better than theory predicts and identically with the optimal distinguisher. We showed that we could improve its performance by limiting the number of hypotheses being tested.

4. We introduced practical improvements of the algorithm such as non-integral step sizes, a file-based sample storage solution and an implementation of the algorithm.

We also improved the classical state-of-the-art approaches for k-sieving - lattice sieving where k vectors are combined at a time - by using quantum algorithms. At the cost of a small increase in time complexity we managed to drastically decrease the space requirement compared to the state-of-the-art quantum algorithm for solving the SVP.

Finally, we developed an algorithm for decoding linear codes where the noise is Gaussian instead of binary. We showed how code-based schemes with Gaussian noise are easily broken. We also found other applications for the algorithm in side-channel attacks and in coding theory.

Contribution Statement

This doctoral thesis concludes my work as a Ph.D. student, and is comprised of two main parts. The first part gives an overview of the research field in which I have been working during my Ph.D. studies and a brief summary of my work. The second part is composed of the six following papers:

1. Q. Guo, E. Mårtensson and P. Stankovski Wagner, On the Sample Complexity of solving LWE using BKW-Style Algorithms, under submission, 2020.

2. A. Budroni, Q. Guo, T. Johansson, E. Mårtensson and P. Stankovski Wagner, Making the BKW Algorithm Practical for LWE, in *21st International Conference on Cryptology in India (INDOCRYPT 2020)*, pp. 417-439, 2020, Bangalore, India.

3. E. Kirshanova, E. Mårtensson, E. Postlethwaite and S. Roy Moulik, Quantum Algorithms for the Approximate k-List Problem and their Application to Lattice Sieving, in *Advances in Cryptology–ASIACRYPT 2019, the 25th Annual International Conference on Theory and Application of Cryptology and Information Security*, pp. 521-551, 2019, Kobe, Japan.

4. Q. Guo, T. Johansson, E. Mårtensson and P. Stankovski, Some Cryptanalytic and Coding-Theoretic Applications of a Soft Stern Algorithm, in *Advances in Mathematics of Communications,* vol. 13, no. 4, pp. 559-578, 2019.

5. Q. Guo, T. Johansson, E. Mårtensson and P. Stankovski, On the Asymptotics of Solving the LWE Problem Using Coded-BKW with Sieving, in *IEEE Transactions on Information Theory*, vol. 65, no. 8, pp. 5243-5259, 2019.

6. E. Mårtensson, The Asymptotic Complexity of Coded-BKW with Sieving Using Increasing Reduction Factors, in *2019 IEEE International Symposium on Information Theory (ISIT),* pp. 2579-2583, 2019, Paris, France.

The papers are slightly reformatted to fit within the overall thesis structure. The thesis is concluded with a popular science summary in Swedish.

For all papers in this list and the list of other contributions, the authors are listed in alphabetical order, which is common in mathematically oriented research areas[1].

In Paper 1 I developed the pruned FFT approach together with the first author. I showed why the FFT distinguisher is optimal. I did the implementation work by modifying code developed for Paper 2. I developed the ideas for what implementations to run together with the first author and I ran the simulations. I did the main part of writing the paper.

In Paper 2 I did the implementation work together with the first and fifth author. I ran the simulations together with the first author. I developed the file-based solution algorithm together with the fifth author. I wrote the paper together with all the other authors.

In Paper 3 I developed the graph-based algorithms, derived their complexities and wrote the paper together with all the other authors.

In Paper 4 I developed the implementation, the numerical examples and parts of the applications. I did the complexity analysis together with the other authors. I ran the simulations together with the first author, which helped a lot in estimating the complexity of the algorithm. I wrote the paper together with all the other authors.

In Paper 5 I did the numerical optimizations to calculate the time complexities. I made the illustrations that increased the readability of the paper and greatly helped in generalizing the ideas for Paper 6. I wrote the paper together with all the other authors.

In Paper 6 I was the only author.

The papers' contributions to the research area are covered throughout Part I of the thesis. Paper 1 to 3, 5 and 6 are summarized in Chapter 9. Paper 4 is covered in Chapter 4.

[1] `http://www.ams.org/profession/leaders/CultureStatement04.pdf`

Other Contributions

The following peer-reviewed publications have also been published during my PhD studies, but are not included in this dissertation.

- S. Gunnarsson, P. Larsson, S. Månsson, E. Mårtensson and J. Sönnerup, Enhancing Student Engagement Using GitHub as an Educational Tool, In *Introduction to Teaching and Learning in Higher Education*, Centre for Engineering Education, Faculty of Engineering, Lund University, 2017.
- S. Gunnarsson, P. Larsson, S. Månsson, E. Mårtensson and J. Sönnerup, Engaging Students using GitHub as a Learning Management System, In *Lund University's Teaching and Learning Conference 2017*, 2017, Lund, Sweden.

The following publications were the conference versions of Paper 4 and Paper 5 respectively.

- Q. Guo, T. Johansson, E. Mårtensson and P. Stankovski, Information Set Decoding with Soft Information and some cryptographic applications, In *2017 IEEE International Symposium on Information Theory (ISIT),* pp. 1793-1797, Aachen, Germany, 2017.
- Q. Guo, T. Johansson, E. Mårtensson and P. Stankovski, Coded-BKW with Sieving, In *Advances in Cryptology–ASIACRYPT 2017, the 23rd Annual International Conference on Theory and Application of Cryptology and Information Security*, pp. 323-346, 2017, Hong Kong, China.

The work done during this PhD has been supported by the Swedish Research Counsel (grant 2015-04528) and the Swedish Foundation for Strategic Research (grant RIT17-0005).

Acknowledgments

Already in the fall of 2010 Fredrik Persson predicted that I would pursue a PhD after my undergraduate studies. While I wasn't as convinced as him at the time, he turned out to be correct. It wouldn't have been possible without the help of some people.

First of all my deepest gratitude goes to my main advisor Thomas Johansson. In a time of great uncertainty in my life he gave me the great opportunity of getting paid for learning how to do research. Under his patient guidance I've developed from a confused undergraduate student into a researcher with the confidence to publish research papers, travel the world and present them in front of cryptology experts.

I also thank my assistant advisors Paul Stankovski Wagner and Qian Guo. Paul for his patience during my many frustrating moments and for his ability to view things from a unique and clear perspective. Qian for being a role model research-wise and for always being helpful despite my never-ending series of questions.

Next I would like to thank my international collaborators. Elena, Eamonn and Subhayan for introducing me to the finer details of the fascinating areas of quantum computation and lattice sieving. Alessandro for his help in turning our BKW implementation from a mess into a polished product.

During my time as a PhD student my research group has increased tremendously in size, making it difficult to thank everyone. There are however some people that I would like to acknowledge a little bit extra.

Linus, you unofficially worked as my fourth advisor answering many stupid questions. Your low profile and general niceness make it hard to make any joke about you. Jonathan "den lille doktor" Sönnerup, even though we failed miserably at sharing an office we've had many great adventures together. You taught me the importance of sometimes going "full Jonathan". Without the two of us the research group might become a bit more productive, but it will also not have as much fun! Jing, in spite of all our pranks, jokes, cultural and linguistic misunderstandings you have always been kind and kept a positive spirit. Joakim, even though we oftentimes disagree we have had many interesting discussions and you have given me new perspectives on many topics.

The communication group deserves thanks for saving me from my own group's inability to realize the importance of having lunch in a different building than the one you work in. Special thanks go to Sara for initially inviting me to the lunches and teaching me to appreciate coffee.

I would like to thank the technical and administrative personnel for their helpfulness throughout my years at EIT. Special thanks go to Elisabeth Nordström for patiently helping me despite my ability to always find strange special

cases when reporting my many travel bills.

I would like to thank Bertil, Stefan, Mats, Martin, Morgan and the rest of the Friday pizza crew for the delicious food and the fascinating discussions, in spite of their inability to recognize Africana as the best pizza!

The number of people to thank outside the university is too many to mention them all. There are however a few that deserve special mentions. Måns Jarlskog, my brother in math, for being at my side all these years. Regardless of having worked in different areas for many years we have a spooky ability as former π students to view problems from a similar perspective. Sara, regardless of her mathematical shortcomings, for teaching me valuable things outside the academic area. Gustav, for many fun times together, regardless of his constant interruptions and spending more time in my office than most of my colleagues. I'd like to thank the TM crew for our many exciting, frustrating and fascinating board game sessions.

I would like to thank the Swedish taxpayers for funding my research and giving me the luxury of thinking for a living.

Finally I would like to thank my family who has supported and inspired me from the very beginning. My "little" brother Gustav, who is becoming a doctor in the more everyday meaning of the word. My mother Eva from whom I got my interest in mathematics. In another time era she would likely have studied more mathematics. My father Hans, who has both been a role-model academically and showed what truly is important in life.

Erik Mårtensson

December 2020

List of Acronyms

AES Advanced Encryption Standard 5

AWGN Additive White Gaussian Noise 33

BDD Bounded Distance Decoding 21

BKZ Block Korkine Zolotarev 20

BKW Blum-Kalai-Wasserman 17

CVP Closest Vector Problem 21

DES Data Encryption Standard 3

DFT Discrete Fourier Transform 15

DLP Discrete Logarithm Problem 7

FFT Fast Fourier Transform 16

FHE Fully Homomorphic Encryption 13

FWHT Fast Walsh-Hadamard Transform 16

IFP Integer Factoring Problem 7

IID Independent and Identically Distributed 33

ISD Information Set Decoding 33

Contents

Part I

Overview of Research Field

Chapter 1

Introduction to Post-Quantum Cryptology

> Factoring is hard. Let's go shopping!
>
> — *Nadia Heninger/Sharon Goldberg*

1.1 Brief Introduction

CRyptography (from the Greek kryptós "hidden/secret" and graphein "to write") is, slightly simplified, the study of methods for secure communication in the presence of adversaries. Cryptanalysis (from the Greek analýein "to loosen/untie") takes the perspective of the adversary and studies how to break cryptographic constructions. Cryptology (from the Greek logia "study") refers to the study of cryptography and cryptanalysis.

Cryptography should be distinguished from steganography, which is the art of concealing the existence of a message altogether.

Historically, before the computer area, cryptography mainly dealt with encryption of secret messages sent for state/military purposes. It was done by pen and paper or, at the first half of the 20th century, by mechanical devices. The methods were quite ad hoc and cryptanalysis was about solving puzzles rather than well-defined mathematical problems.

The fascinating history of cryptography is outside the scope of this thesis. For a brief, non-technical introduction to the history of cryptography, see [Sin99]. For a slightly more technical introduction, see [Chu01]. For a comprehensive introduction to the history of cryptography up until the early internet era, see [Kha96].

The modern history of cryptography arguably starts with Shannon's paper "Communication theory of secrecy systems" [Sha49], transforming cryptography from an artform into a science.

Cryptography as an academic discipline arguably starts in the 1970s with two major developments. Partly, the Data Encryption Standard (DES) was introduced as the first wide-spread, publicly available encryption method. Partly, the invention of public-key cryptography by Diffie and Hellman [DH76], and

Merkle [Mer78], made it possible for two people that never met before to securely communicate over an insecure channel.

Public-key cryptography is based on the inherent (assumed) difficulty of one among a small set of mathematical problems. The beautiful thing about this research area is that, unlike in many contexts where the mathematical model is an approximation of reality, in cryptography we get to decide the mathematical problem precisely.

In the 90s Peter Shor showed how the public-key cryptography used then, and still today, can be broken by a large-scale quantum computer [Sho94]. This sparked research into developing quantum-resistant methods for public-key cryptography. This area of research is called post-quantum cryptography[1].

Modern cryptography is in some sense quite a strange research area. Whereas in other areas researchers try to solve problems, in cryptography researchers come up with mathematical problems that no one should be able to solve.

Cryptanalysis, viewed as the art of breaking cryptographic constructions, is in a sense even stranger. Here researchers try their best to solve problems, that by their very construction, if correctly designed, are impossible to solve.

In other research areas quantum computers are seen as a fantastic future tool, making it possible to solve problems that by their very nature are hard or even impossible to solve with a classical computer. In the area of cryptography they are seen as a foundational threat[2] that needs to be addressed long before large-scale quantum computers even exist.

To further highlight the strangeness of cryptography, consider the famous P vs. NP problem [Coo71]. Imagine a future where someone would show the, improbable but technically not impossible, result that $P = NP$. For many research areas this would be a dream come true and it would make many currently intractable problems easy to solve[3]. However, in cryptography this would be a disaster since it would make public-key cryptography impossible to use[4].

The topic of this thesis is the even stranger area of post-quantum cryptanalysis. In this area researchers try their hardest to break systems that are designed to resist both classical and quantum computers.

To the uninitiated, it might seem strange to focus on breaking systems. However, the only way to make sure that a cryptographic construction is secure is to try as hard as possible to break it[5]. If you discourage people to study your system because you think academic researchers can break it, then chances are that malicious players can break it too!

[1] which is very different from quantum cryptography, the science of performing cryptographic tasks using quantum mechanical phenomenon.

[2] But also a job opportunity for researchers.

[3] Unless the solution is a polynomial but practically inefficient algorithm or the solution is non-constructive in nature.

[4] See Footnote 3.

[5] In some cases, such as the Vernam cipher, it is possible to prove that a construction is secure. However, at least in the area of public-key cryptography, constructions depend crucially on the inherent difficulty of mathematical problems. Since computer scientists famously have not even been able to prove that NP-complete problems are hard, we need to heuristically assume that problems are hard if no one has been able to solve them efficiently after large amounts of effort to do so.

1.1.1 General Thesis Focus

When writing an introduction to a collection thesis we try to

1. give a comprehensive introduction to the general research area,
2. give the reader an easier time to read the papers the thesis is based on,
3. write as strictly as possible,

under the constraint that the introduction should not be unreasonably long. Fully achieving all the goals under this constraint is not possible. I have decided to focus on making the papers the thesis is based on more readable. One of the main goals when writing a paper is to make the ideas easier to understand than it was for the author to come up with them in the first place. Analogously, my main goal with this thesis introduction is to make the papers it is based on easier to read, than they would be without having read this introduction, both by giving the reader extra intuitive explanations and by putting the papers in a shared larger context.

For more mathematical strictness the reader is referred to the papers the thesis is based on.

For many different areas, all the way from cryptography in general to specific topics like quantum computation and lattice sieving, well-written general introductions are referenced throughout the thesis to guide the interested reader.

1.2 Classical Cryptography

Let us now introduce classical cryptography in some more detail, covering the basic ideas of symmetric and asymmetric cryptography.

1.2.1 Symmetric Cryptography

Alice and Bob[6] are communicating over an insecure channel where Eve is eavesdropping. Secret values are marked in red. Encryption is used to hide the secret message from Eve. Alice maps her message m to an encrypted message c using an invertible function E_k, where the key k is only known to Alice and Bob. Typically, k is chosen uniformly random from a very large set K of possible keys. Next, Bob retrieves the secret message m by applying a decryption function D_k to the encrypted message c. Figure 1.1 illustrates this process.

A fundamental concept here is Kerckhoffs's principle, which says that a cryptosystem should remain secure even if the attacker knows everything about how the system works, except the secret key [Ker83a, Ker83b]. This principle is universally accepted in cryptographic research. The reasoning behind it is essentially that it is much easier to hide a short string like a key, than the whole inner workings of the system.

The current standard for general-purpose symmetric cryptography is the Advanced Encryption Standard (AES). Given access to only a message encrypted

[6] In cryptography instead of writing something like "the sender" and "the receiver", we usually use a set of human names, making the reading simpler. In general and practice Alice and Bob can refer to computers, servers, mobile phones, tablets or any type of electronical devices.

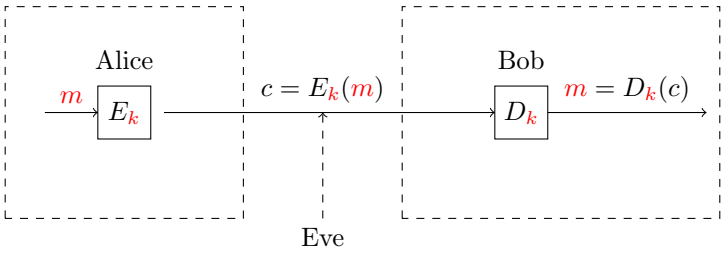


Figure 1.1: A simple illustration of symmetric encryption.

using AES, the best known way of decrypting the message without access to the secret key is just marginally faster than exhaustively trying to decrypt the message using every possible key [BKR11][7].

Hash Function

A (cryptographic) hash function is a function H that maps data of arbitrary size to a bit array of a fixed size n and has the following properties.

1. Pre-image resistance - given a hashed value h, it should be hard to find a message m such that $H(m) = h$.
2. Second pre-image resistance - given a message m_1, it should be hard to find another message m_2 such that $H(m_1) = H(m_2)$.
3. Collision-resistance - It should be hard to find two messages m_1 and m_2 such that $H(m_1) = H(m_2)$.

Notice here that unlike the case of encryption, a hash function is unkeyed. We can trivially find a pre-image by brute-force in time $\mathcal{O}(2^n)$. By the birthday paradox we will find a collision after having hashed $\mathcal{O}(2^{n/2})$ messages. To achieve m-bit collision-resistance we thus require $n \geq 2 \cdot m$.

1.2.2 Asymmetric Cryptography

Asymmetric cryptography is also more commonly known as public-key cryptography, but for reasons of symmetry we will refer to it as asymmetric throughout this thesis.

Asymmetric Encryption

A major problem for symmetric cryptography is how Alice and Bob can decide which shared key to use, if they only have access to an insecure channel and have not communicated beforehand. This seemingly impossible problem was solved in 1976 by the Diffie-Hellman key exchange protocol [DH76][8]. A year later, a similar system called Rivest-Shamir-Adleman (RSA) was invented [RSA78]. In

[7] In practice there are of course a lot of other attacks to consider.

[8] In 1997 it was revealed that the British signal intelligence agency secretly knew about asymmetric cryptography already back in 1969.

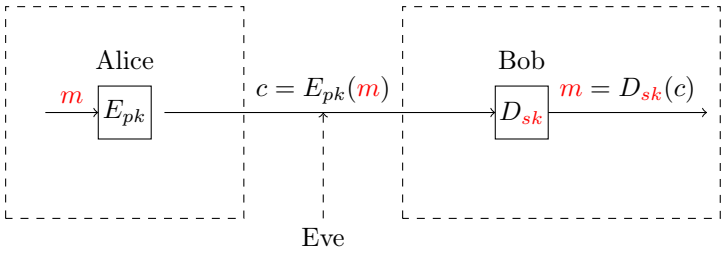


Figure 1.2: A simple illustration of asymmetric encryption.

RSA when Alice wants to send an encrypted message to Bob, Bob uses two keys, a private (secret) key sk and a public key pk. Using Bob's public key pk, Alice can encrypt a message and send it to Bob. Having access to the private key sk, Bob can easily decrypt the message. Eve, who does not have access to the private key, must solve a difficult mathematical problem to decrypt the message.

The security of RSA or the Diffie-Hellman key exchange rely on the assumption that the Integer Factoring Problem (IFP) or the Discrete Logarithm Problem (DLP) respectively, are computationally intractable[9]. Let us first quickly define the problems.

Definition 1.2.1 (The Integer Factoring Problem (IFP))**.** *Given the product $n = pq$ of two large primes p and q, find the secret p and q.*

Definition 1.2.2 (The Discrete Logarithm Problem (DLP))**.** *Given a cyclic group G, of large order q and with a generator g. Given $h = g^x$, find the secret x.*

Here we measure the problem size as the number of bits needed to represent the secret numbers. By computationally intractable we mean that there is no algorithm that has a runtime that grows polynomially with the problem size.

The best algorithms for the IFP is the general number field sieve, which solves it in subexponential time, both provably [LP92] and heuristically [BLP93].

The difficulty of the discrete logarithm problem depends on the group used. If we use the multiplicative group $\mathbb{F}_p^*$ for a prime p, then the best algorithm is also the general number field sieve with subexponential time. If we instead use an elliptic-curve group, then the best algorithm is a general-purpose discrete logarithm solving algorithm like Pollard's ρ algorithm [Pol78], taking exponential time.

Digital Signatures

Digital signatures make it possible for Alice to sign a message she sends to Bob. A simple illustration of how it works is found in Figure 1.3. Alice first

[9]Technically the the security of RSA relies on the difficulty of solving the RSA problem, which is a slight modification of the integer factoring problem. If you can solve integer factoring, then you can break RSA. There might be an efficient way of solving RSA even if integer factoring is hard, but no such algorithm is known. The situation is similar for Diffie-Hellman, where the actual problem is a slight modification of the discrete logarithm problem.

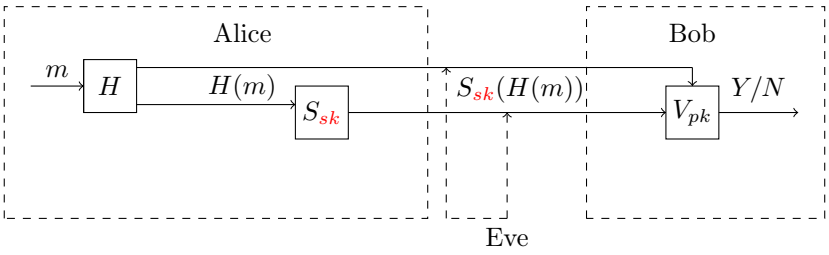


Figure 1.3: A simple illustration of a digital signature.

hashes her message using som well-chosen hash function H[10]. She then signs her message using her signing function S_{sk}, which depends on her secret key sk. Bob recieves her signed value $S_{sk}(H(m))$ and the hash of the original message $h = H(m)$. To check the validity of the message he uses a verification function V_{pk}, which depends on Alice's public key pk. He checks if $h' = V_{pk}(S_{sk}(H(m)))$ is equal to $h = H(m)$ to test the validity of the message. In order to create a valid signature of her own, Eve has to solve a difficult mathematical problem, in practice today either the DLP or the IFP[11].

The picture in this chapter is quite simplified. Specifically we do not explain in detail how the encryption/decryption functions or the signing/verification functions work. A lot more needs to be taken into consideration before implementing cryptography in practice. Asymmetric cryptography depends on, but not only on, DLP or IFP. The key take-home message is that the asymmetric primitives are important and if an attacker can break both the DLP and the IFP then asymmetric cryptography as used today, and hence cryptography as a whole, is broken.

For a more complete mathematical introduction to the cryptology area, see for example [Sma16, MvOV01, SP18, FS03].

1.3 Post-Quantum Cryptography

In the 1990s, quantum algorithms that can potentially be used to break cryptographic constructions in the future were developed.

1.3.1 Shor's Algorithm

In [Sho94] Peter Shor showed how a large-scale quantum computer can solve both the discrete logarithm problem, based on the multiplicative group or elliptic curves, and the integer factoring problem, in polynomial time. This constitutes a foundational threat towards asymmetric cryptography and makes it necessary to use other underlying mathematical problems for asymmetric cryptography.

The research area studying such problems is called post-quantum cryptography. There are currently 5 main categories of post-quantum cryptography primitives.

[10] The hashing is done to transform all message into the same length and to make it impossible to forge signatures.

[11] Also here the actual problem to solve is a slight modification of these problems.

- Lattice-based cryptography
- Code-based cryptography
- Multi-variate cryptography
- Symmetric/Hash-based cryptography
- Super-singular isogeny-based cryptography

Among these, lattice-based cryptography is the focus of this thesis and also arguably the most promising category. Paper 4 is about code-based cryptography [GJMS19a]. The other areas of post-quantum cryptography are outside of the scope of this thesis. For a general introduction to the area, after having read the Wikipedia article [Wik20c], see [BL17, BBD08].

1.3.2 Grover's Algorithm

In [Gro96] Lov Grover developed a quantum algorithm that makes it possible to find an element in an unstructered list of size n in time $\mathcal{O}(\sqrt{n})$, beating the classical time of $\mathcal{O}(n)$. For symmetric systems like AES this allows the attacker to search for the secret key much faster. It also allows for finding pre-images to hash functions. These attacks can easily be neutralized by doubling the key size.

A slight modification of Grover's algorithm is the BHT algorithm for finding collisions in $\mathcal{O}(2^{n/3})$ [BHT97]. To remedy this, the size of the hashed values need to increase by a factor $3/2$ to remain collision-resistant.

While not as powerful as Shor's algorithm, Grover's algorithm is much more general. The algorithm is frequently used in post-quantum cryptanalysis and will be studied in more detail in Chapter 3.

1.3.3 NIST Competition

The threat from quantum computers is not just considered an academic curiosity. The National Security Agency (NSA) has stated the following [Age].

> IAD will initiate a transition to quantum resistant algorithms in the not too distant future. Based on experience in deploying Suite B, we have determined to start planning and communicating early about the upcoming transition to quantum resistant algorithms.

The National Institute of Standards and Technology (NIST) is a United States national agency known for, among many other things, holding international competitions to develop new cryptographic standards, leading to globally recognized standards such as the AES for symmetric cryptography and Secure Hash Algorithm 3 (SHA-3) for hash functions. Currently they have an ongoing competition to develop a new standard for lightweight cryptography [NISa].

To handle the potential future threat of quantum computers NIST issued a competition to develop new standards for asymmetric encryption and digital signatures [NISb].

Category	Signatures	KEM/Encryption	Total
Lattice-based	5	21	26
Code-based	2	17	19
Multi-variate	7	2	9
Symmetric/Hash-based	3	0	3
Other	2	5	7
Total	19	45	64

Table 1.1: Summary of the first round NIST PQC competition candidates.

Category	Signatures	KEM/Encryption	Total
Lattice-based	3	9	12
Code-based	0	7	7
Multi-variate	4	0	4
Symmetric/Hash-based	2	0	2
Other	0	1	1
Total	9	17	26

Table 1.2: Summary of the second round NIST PQC competition candidates.

Throughout the competition[12] lattice-based schemes have dominated. Let us quickly cover the development of the competition.

1.3.4 First Round Candidates

The competition initially got 82 submissions in [NISc]. Out of these 69 were considered "acceptable and complete". 5 of these were withdrawn. The remaining 64 candidates can be divided into categories according to Table 1.1.

1.3.5 Second Round Candidates

Out of the 64 candidates, 26 made it to the second round [NISd]. These can be divided into categories according to Table 1.2.

1.3.6 Third Round Candidates

Out of the remaining 26 candidates, 15 made it to the third round [NISe]. The third round candidates are divided into finalists and alternates, where the finalists are considered the most promising. The categories for the finalists and alternates are summarized in Tables 1.3 and 1.4.

In Tables 1.1 to 1.4 super-singular isogeny-based candidates are considered as "Other". Key Encapsulation Mechanism (KEM) refers to an asymmetric method for establishing a shared key, which in turn is used for symmetric encryption.

[12]NIST technically does not call the process a competition and uses various creative ways of almost, but not quite, calling it a competition.

Category	Signatures	KEM/Encryption	Total
Lattice-based	2	3	5
Code-based	0	1	1
Multi-variate	1	0	1
Total	3	4	7

Table 1.3: Finalists of the third round of the NIST PQC competition.

Category	Signatures	KEM/Encryption	Total
Lattice-based	0	2	2
Code-based	0	2	2
Multi-variate	1	0	1
Symmetric/Hash-based	2	0	2
Other	0	1	1
Total	3	5	8

Table 1.4: Alternates of the third round of the NIST PQC competition.

1.3.7 Some Voices on Quantum Computing

This section will briefly discuss the progress of quantum computing and the seriousness of its threat to cryptography.

One of the first milestones for quantum computing is achieving quantum supremacy, referring to a demonstration of a quantum computer being able to solve a problem that no classical computer can solve in any feasible amount of time[13]. A Google team recently claimed to achieve quantum supremacy [AAB+19]. They sampled a million values from a randomized quantum circuit in about 200 seconds, a feat they claimed that the state-of-the-art classical supercomputers need 10000 years to do. IBM claimed in response that the calculation can be done in at most 2.5 days [PGN+19]. A very recent quantum supremacy claim in the area of boson sampling is from [ZWD+20]. Whether either of these results should be considered as achieving quantum supremacy or not, the time at which quantum supremacy can no longer be denied is near.

A next milestone is showing an example of where a quantum computer can solve an interesting problem significantly faster than any classical computer. There will likely be a couple of steps between this milestone and large integer factoring using quantum computers. The largest number factored by a quantum computer using Shor's algorithm is currently only 35 [ASK19]. While factorizations of larger numbers, such as 1,099,551,473,989 has been performed using a quantum computer [AOAGC18], the methods used are not the same as Shor's algorithm and do not scale as well when increasing the problem size and/or do only apply to numbers on special forms.

There are still reasons to be worried. Moore's law famously states that the number of transistors that fits on an integrated circuit doubles every two years, meaning that the available computational power grows exponentially over time. There is a corresponding quantum version called Neven's law[14], saying

[13] The usefulness of the problem is not relevant at this stage.

[14] after quantum computing researcher Hartmut Neven.

that quantum computers ability to solve certain problems, like integer factoring, grows doubly exponential, since they have an inherent exponential advantage to begin with. This doubly exponential growth assumes that the number of qubits a quantum computer can handle will grow exponentially over time, similar to the number of transistors in classical computers.

Another perspective on post-quantum computing is Mosca's theorem, named after quantum computing researcher Michele Mosca. It essentially says that we need to worry if

$$X + Y > Z,$$

where X is the time a message needs to remain secret, Y is the time it takes to implement post-quantum solutions and Z is the time until a quantum computer can break our current asymmetric cryptography. In general both X and Y are oftentimes many years, meaning that the threat from quantum computers needs to be dealt with many years before large-scale quantum computers exist.

Yet another perspective on the threat from quantum computers is that of a simple risk-benefit analysis. Due to the disastrous consequences a large-scale quantum computer would be for all today's secure communication, as long as we think that the probability of a quantum computer being built is more than non-negligible, we must take precautions against it in advance.

In the world of post-quantum cryptography you sometimes get the impression that governments and corporations spend enormous resources on building quantum computers for the sole purpose of destroying cryptography as being used today. Actually, quantum algorithms have applications in many other areas. A couple of examples with their corresponding Wikipedia articles are

- Simulating quantum systems that are too difficult to study empirically or simulate with classical supercomputers [Wik20f].
- Optimization using quantum annealing, essentially a quantum version of the optimization method simulated annealing [Wik20e].
- Quantum algorithms for solving linear systems of equations [Wik20d], which are prevalent in essentially every area of science and engineering.

For an attempt at listing all quantum algorithms see [Qua][15]. Due to the many different applications of quantum computers, both academia and industry spend lots of resources building quantum computers. If humanity fails to build a large-scale quantum computer, it will likely be because of some inherent physical difficulty, not for the lack of trying!

1.4 LWE and LPN

Most of the lattice-based candidates in the NIST post-quantum competition are based on the Learning With Errors (LWE) problem, introduced by Regev in [Reg05]. Let us define its search version.

[15] Compared to algorithms on a classical computer this list is still comically short. The idea of attempting to make a complete list of classical algorithms is of course absurd.

Definition 1.4.1 ((Search) Learning With Errors (LWE))**.** *Let n be a positive integer, q be an odd prime and χ be a distribution on $\mathbb{Z}_q$. Let $\mathbf{s}$ be a random secret vector in $\mathbb{Z}_q^n$, chosen from some distribution. Given access to m noisy products between $\mathbf{s}$ and known, uniformly random vectors $\mathbf{a}_i \in \mathbb{Z}_q^n$;*

$$(\mathbf{a}_i, b_i) = (\mathbf{a}_i, \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i),$$

where e_i is sampled from χ. The (search) LWE problem is to find the secret vector $\mathbf{s}$.

There is also a corresponding decision version of the LWE problem where, given access to m pairs $(\mathbf{a}_i, b_i)$, the task is to decide whether the pairs are LWE samples or uniformly random samples from $\mathbb{Z}_q^{n+1}$.

Other than being among the main problems (if not *the* main problem) for post-quantum cryptography, LWE has a couple of more fascinating properties. The first Fully Homomorphic Encryption (FHE) system is based on the LWE problem [Gen09]. For a long time the possibility of FHE was an open research question. There are reductions from worst-case lattice problems to average-case LWE, implying that even on average LWE is a hard problem to solve [Reg05, Pei09].

Let us also define the Learning Parity with Noise (LPN) problem, which is the corresponding binary problem.

Definition 1.4.2 ((Search) Learning Parity with Noise (LPN))**.** *Let k be a positive integer, let $\mathbf{x}$ be a secret binary vector of length k and let $X \sim Ber_\eta$ be Bernoulli distributed with the parameter $\eta > 0$. Let $L_{\mathbf{x},X}$ denote the probability distribution on $\mathbb{F}_2^k \times \mathbb{F}_2$ obtained by choosing $\mathbf{g}$ uniformly at random, choosing $e \in \mathbf{F}_2$ from X and returning*

$$(\mathbf{g}, z) = (\mathbf{g}, \langle \mathbf{g}, \mathbf{x} \rangle + e).$$

The (search) LPN *problem is to find the secret vector $\mathbf{x}$ given a fixed number of samples from $L_{\mathbf{x},X}$.*

The LPN problem also has a a decision version, analogously with the LWE problem.

1.4.1 Matrix Formulation of the LWE Problem

We can reformulate the LWE problem of Definition 1.4.1 on matrix form as

$$\mathbf{b} = \mathbf{sA} + \mathbf{e}, \tag{1.1}$$

where $\mathbf{A} = [\mathbf{a}_1^T \cdots \mathbf{a}_m^T]$, $b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i$ and $e_i \xleftarrow{\$} \chi$.

The corresponding reformulation can of course also be done for the LPN problem.

1.4.2 Parameter Choices

Asymptotically we typically use parameters $q = \mathcal{O}(n^{c_q})$ and $\sigma = \mathcal{O}(n^{c_s})$, for small positive constants c_q and c_s. In Regev's original paper he used $c_q = 2$ and $c_s = 1.5$. His reduction proof requires that $c_s \geq 0.5$. As we will briefly discuss

in Chapter 2, there are efficient algorithms for solving LWE when $c_s < 0.5$. Therefore either $c_s \geq 0.5$ is used or the scheme needs to make sure that the number of available samples is small, preventing the efficient attacks explained in Chapter 2.

1.5 Preliminaries

Let us cover some notation, concepts and methods that will be useful to know when reading the rest of the thesis.

1.5.1 Notation

Let $S^{d-1} \subset \mathbb{R}^d$ denote the unit sphere in $\mathbb{R}^d$. Unless otherwise specified in this introduction we use the Euclidean norm, denoted by $\|\cdot\|$. We let $\langle \mathbf{x}, \mathbf{y} \rangle$ denote the scalar product between vectors $\mathbf{x}, \mathbf{y}$.

Given a set $E \subset \mathbb{R}^d$, by its orthogonal complement we mean

$$E^{\perp} = \{\mathbf{x} \in \mathbb{R}^d : \langle \mathbf{x}, \mathbf{y} \rangle = 0, \forall \mathbf{y} \in E\}. \tag{1.2}$$

We define the orthogonal projection of $\mathbb{R}^d$ on $E^{\perp}$ as the (unique) linear map $\pi_E : \mathbb{R}^d \to E^{\perp}$ such that

$$\pi_E(\mathbf{x}) = \begin{cases} \mathbf{x}, \forall \mathbf{x} \in E^{\perp}, \\ \mathbf{0}, \forall \mathbf{x} \in E. \end{cases} \tag{1.3}$$

1.5.2 Complexity Theory

There are essentially three different ways of comparing the computational effort of running an algorithm; asymptotic complexity, concrete complexity and runtime. All are relevant in cryptography and the papers this thesis is based on use all these notations of complexity.

Asymptotic Complexity

Let $f(n)$ denote the number of operations needed to run an algorithm with a problem size n. For asymptotics we use the following standard notation.

- $f(n) = \mathcal{O}(g(n))$ if there is a constant C such that $f(n) \leq C \cdot g(n)$ for sufficiently large values of n.
- $f(n) = \Omega(g(n))$ if there is a constant C such that $f(n) \geq C \cdot g(n)$ for sufficiently large values of n.
- $f(n) = \Theta(g(n))$ if $f(n) = \mathcal{O}(g(n))$ and $f(n) = \Omega(g(n))$.
- $f(n) = \tilde{\mathcal{O}}(g(n))$, if there is a constant k such that $f(n) = \mathcal{O}(g(n) \log^k g(n))$.
- $f(n) = o(g(n))$, if $\lim_{n \to \infty} f(n)/g(n) = 0$.

Asymptotic analysis makes it simple to compare algorithms behavior as the problem sizes grow towards infinity. An asymptotically faster algorithm will eventually beat a slower one. However, asymptotic complexity in and of itself is not enough. An algorithm can be asymptotically faster than another algorithm, while being slower for all practically solvable instances even if we would use all the world's computing power until the heat death of the universe!

In the cryptographic context we prefer problems where the best algorithm solves the problem in exponential time. We must avoid problems where the best algorithms solve the problem in polynomial time.

Concrete Complexity

We can calculate how many operations an algorithm takes to solve a problem instance of a given problem size n. Here we can talk about arithmetic complexity, referring to the number of arithmetic operations in a specific field (like $\mathbb{F}_q$), or even the number of bit operations needed to run the algorithm.

For a specific value of n this method gives us a more precise idea of the complexity of an algorithm. However, calculating the concrete complexity of an algorithm is oftentimes more complicated than calculating the asymptotic complexity.

In the cryptographic context we talk about λ-bit complexity/security when we refer to a problem instance where the best algorithm needs at least 2^λ operations to solve the problem instance. In cryptography common security levels are 80, 128 or 256 bits.

Runtime Complexity

Finally we can run an implementation on actual hardware and measure the runtime, in for example seconds or even clock cycles. In a sense this is the practically most relevant way of comparing algorithms. However, in order to make an apples to apples comparison between two algorithms we need to take into consideration efficiency of the implemented algorithm, available hardware and so on. Comparing only runtime also makes it hard to tell how the performance of the algorithms change as n increases.

1.5.3 Transforms

Here we will introduce some important transforms which will be used later.

The Discrete Fourier Transform

An important cryptanalytic tool is the Discrete Fourier Transform (DFT).

Definition 1.5.1 (Discrete Fourier Transform (DFT))**.** *Let $\theta_q = \exp(2\pi i/q)$ denote the q-th root of unity and let $f : \mathbb{F}_q^n \to \mathbb{R}$. The DFT of f is the mapping*

$$\hat{f}(\mathbf{y}) = \sum_{\mathbf{x} \in \mathbb{F}_q^n} f(\mathbf{x}) \theta_q^{-<\mathbf{x},\mathbf{y}>}, \tag{1.4}$$

for all $\mathbf{y} \in \mathbb{F}_q^n$.

Calculating the DFT naively takes time $\mathcal{O}(q^{2n})$. However, by applying a divide-and-conquer technique this can be sped-up to $\mathcal{O}(n \cdot q^n \cdot \log(q))$ using for example the Cooley-Tukey algorithm [CT65]. This foundational method in computer science is called the Fast Fourier Transform (FFT).

The Walsh-Hadamard Transform

An important, binary special case of the DFT is the Walsh-Hadamard Transform (WHT).

Definition 1.5.2 (Walsh-Hadamard Transform (WHT))**.** *Given the Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}$. The WHT of f is the mapping*

$$\hat{f}(\mathbf{y}) = \sum_{\mathbf{x} \in \mathbb{F}_2^n} f(\mathbf{x})(-1)^{<\mathbf{x},\mathbf{y}>}, \tag{1.5}$$

for all $\mathbf{y} \in \mathbb{F}_2^n$.

Just like for the DFT there is a divide-and-conquer algorithm for calculating the WHT faster than naively. This algorithm is called the Fast Walsh-Hadamard Transform (FWHT) and takes $\mathcal{O}(n \cdot 2^n)$ time.

1.5.4 Probability Distributions

Let us define a couple of important probability distributions.

Definition 1.5.3 (Bernoulli Distribution)**.** *Given $\rho \in [0, 1]$. A variable X is called Bernoulli distributed Ber_ρ if it has the following Probability Mass Function (PMF)*

$$\mathbf{Pr}[X = x] = \begin{cases} 1 - \rho & \text{if } x = 0, \\ \rho & \text{if } x = 1. \end{cases}$$

Definition 1.5.4 (Uniform Distribution)**.** *A variable X is called uniformly distributed $\mathcal{U}(a, b)$ if it has the following PMF*

$$\mathbf{Pr}[X = x] = \frac{1}{b - a + 1}, \quad x = a, a + 1, \ldots, b.$$

Definition 1.5.5 (Gaussian Distribution)**.** *Given $\mu, \sigma \in \mathbb{R}$, where $\sigma > 0$. A variable X is called Gaussian distributed $\mathcal{N}(\mu, \sigma)$ if it has the following Probability Density Function (PDF)*

$$f(x|\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

Definition 1.5.6 (Discrete Gaussian Distribution)**.** *Given $q \in \mathbb{N}, \sigma \in \mathbb{R}$, where $q > 2$ and q odd, $\sigma > 0$. Let $f(x|0, \sigma)$ denote the PDF of the Gaussian distribution. Let us first consider the discrete Gaussian distribution over the integers as the distribution with PMF proportional to $\exp(-x^2/(2\sigma^2))$ to each $x \in \mathbb{Z}$. The discrete Gaussian function over $\mathbb{Z}_q$ ($\chi_{\sigma,q}$) is the distribution we get by sampling from the integer version and then folding the values to the closest value in $\mathbb{Z}_q$.*

Definition 1.5.7 (Rounded Gaussian Distribution)**.** *Given $q \in \mathbb{N}, \sigma \in \mathbb{R}$, where $q > 2$ and q odd, $\sigma > 0$. Let $f(x|0,\sigma)$ denote the PDF of the Gaussian distribution. A variable X is called rounded Gaussian distributed $\bar{\Psi}_{\sigma,q}$ if it has the following PMF.*

$$\Pr(X = e) = \sum_{k=-\infty}^{\infty} \int_{e-1/2+k\cdot q}^{e+1/2+k\cdot q} f(x|0,\sigma)dx, \quad e = -(q-1)/2, \ldots, (q-1)/2.$$

Another way of viewing the rounded Gaussian distribution is as the distribution that samples from the Gaussian distribution $\mathcal{N}(0,\sigma)$, rounds to the nearest integer and then wraps the value to the interval $[-(q-1)/2, (q-1)/2]$. The discrete and rounded Gaussian distributions are quite similar in nature and both are commonly used for the noise in the LWE problem.

1.6 Thesis Outline

The structure of the rest of the introductory part of the thesis is as follows. Chapter 2 briefly covers the main types of algorithms for solving LWE. Next, Chapter 3 introduces quantum algorithms, with a focus on Grover's algorithm and its generalizations. Thereafter, Chapter 4 introduces coding theory, code-based cryptography and cryptanalysis. Chapter 5 discusses lattice sieving with a focus on asymptotics. Next, Chapters 6 to 8 constitute the main body of the thesis, covering the Blum-Kalai-Wasserman (BKW) algorithm, including the basic algorithm, improved reduction steps and improved hypothesis testing. Finally, Chapter 9 concludes the thesis.

Chapter 2

Algorithms for Solving LWE

> LWE is hard. Let's go factoring!
>
> — *Post-quantum version of the factoring quote*

The LWE problem is believed to be a hard problem. For typical parameter settings used for cryptographic purposes the time complexity for the fastest known algorithms, even when using a quantum computer, are exponential in the problem size. Algorithms for solving the LWE problem is the main topic of this thesis. This chapter will give an overview of the algorithms used to solve LWE. These algorithms can be divided into three main categories.

2.1 Algebraic Approaches

Algebraic methods for solving LWE were introduced by Arora and Ge in [AG11]. Consider an LWE sample on the form

$$b = \mathbf{s} \cdot \mathbf{a} + e. \tag{2.1}$$

Assume that $e \in S \subset \mathbb{Z}_q$. As an example, if the noise is Discrete Gaussian, then we can assume that the error term has a magnitude of at most $t \cdot \sigma$, where t is a small integer. Now consider the function

$$f_{\mathbf{a},b}(\mathbf{x}) = \prod_{e \in S} (b - \mathbf{x} \cdot \mathbf{a} - e). \tag{2.2}$$

If $\mathbf{s}$ is the secret vector, then $f_{\mathbf{a},b}(\mathbf{s}) = 0$. For $\mathbf{x} \neq \mathbf{s}$ we, most likely, get $f_{\mathbf{a},b}(\mathbf{x}) = 0$. Having access to m samples we consider the system of polynomial equations

$$\{f_{\mathbf{a}_i,b_i}(\mathbf{x}) = 0\}_{i=1}^{m}. \tag{2.3}$$

These polynomials have a degree of $|S|$. The number of monomials is $\binom{n+|S|}{|S|}$[1]. Next we solve (2.3) by linearization. Replace each monomial by a

[1] Each monomial is formed by picking one term out of $n+1$ from each of the $|S|$ parantheses, allowing repeating a term but ignoring the order in which the terms are picked. Thus the number is $\binom{(n+1)+|S|-1}{|S|} = \binom{n+|S|}{|S|}$ [Wik20a]. (To be precis this number should be subtracted by 1 since we do not need to introduce a variable for the constant term.)

new variable. Then solve the linear system of equations. If the number of available samples is large enough, then this system of equations has a unique solution $\mathbf{x} = \mathbf{s}$.

The original Arora-Ge algorithm was improved by Albrecht et al. in [ACFP14], using Gröbner bases instead of linearization. If $c_s < 0.5$, then Arora-Ge is a subexponential algorithm. However, for $c_s > 0.5$ it is inefficient compared to the other algorithms of this chapter. The algorithm also does not work if the number of provided samples is too small.

2.2 Lattice-based Approaches

To explain lattice-based approaches for solving LWE we need to introduce some lattice concepts. These concepts will also be useful later in Chapter 5 where we discuss lattice sieving. For a more general introduction to lattice-based cryptography, see [MR09].

2.2.1 Introduction to Lattices

Let us start by defining a lattice.

Definition 2.2.1 (Lattice)**.** *A lattice $\mathcal{L} \subset \mathbb{R}^d$ is a discrete, additive subgroup of $(\mathbb{R}^d, +)$.*

A lattice being discrete means that every lattice point in $\mathcal{L}$ has a neighborhood around it in which it is the only element. Next we define the rank of and a basis of a lattice.

Definition 2.2.2 (Rank)**.** *The rank of a lattice $\mathcal{L}$, rank($\mathcal{L}$), is the maximum number of linearly independent vectors in $\mathcal{L}$.*

Definition 2.2.3 (Basis)**.** *Let $\mathcal{L}$ be a lattice and let $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ be linearly independent lattice vectors in $\mathcal{L}$. If for every $\mathbf{v} \in \mathcal{L}$ there exists a set of integers $\{x_1, \ldots, x_n\}$ such that $\mathbf{v} = \sum_{i=1}^{n} x_i \mathbf{b}_i$, then $\mathbf{B}$ is a basis of $\mathcal{L}$. Then we also say that $\mathbf{B}$ spans $\mathcal{L}$.*

To indicate that $\mathbf{B}$ is a basis of a lattice $\mathcal{L}$ we write $\mathcal{L}(\mathbf{B})$.

In this thesis we are interested in maximum-rank lattices, in other words, from now on we assume that $\text{rank}(\mathcal{L}) = d$. To explain the Block Korkine Zolotarev (BKZ) algorithm we need the concept of projective lattices. To do so, let us first introduce sublattices and primitive sublattices.

Definition 2.2.4 (Sublattice)**.** *Given a lattice $\mathcal{L}$ in $\mathbb{R}^d$. A set $\mathcal{L}' \subset \mathcal{L}$, which in turn is a lattice is called a sublattice of $\mathcal{L}$.*

Definition 2.2.5 (Primitive sublattice)**.** *Given a lattice $\mathcal{L}$ in $\mathbb{R}^d$. A sublattice $\mathcal{L}'$ of $\mathcal{L}$ is called primitive if for any basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_r\}$ of $\mathcal{L}'$ there exists a set of vectors $\{\mathbf{b}_{r+1}, \ldots, \mathbf{b}_d\}$ in $\mathcal{L}$, such that $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ is a basis of $\mathcal{L}$.*

Now, let us now introduce projected lattices.

Lemma 2.2.1 (Projected lattice)**.** *Let $\mathcal{L}$ be a full-rank lattice in $\mathbb{R}^d$ and let $\mathcal{L}'$ be primitive sublattice of $\mathcal{L}$ with a basis $\mathbf{B}_r = \{\mathbf{b}_1, \ldots, \mathbf{b}_r\}$. Let $\pi_{\mathcal{L}'}$ denote the orthogonal projection on $(\mathcal{L}')^{\perp}$. Then $\pi_{\mathcal{L}'}(\mathcal{L})$ is a $(d-r)$-rank lattice.*

Proof 2.2.1. *The basis of $\mathcal{L}'$ can be completed into a basis for $\mathcal{L}$ as $\mathbf{B} = \mathbf{B}_r \cup \{\mathbf{b}_{r+1}, \ldots, \mathbf{b}_d\} = \{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$. Then $\pi_{\mathcal{L}'}(\mathcal{L})$ is the lattice generated by the basis $\{\pi_{\mathcal{L}'}(\mathbf{b}_{r+1}), \ldots, \pi_{\mathcal{L}'}(\mathbf{b}_d)\}$.*

Given a basis $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$, the projection $\pi_{\mathcal{L}(\mathbf{B}_{i-1})}$ is denoted π_i. Let also π_1 denote the identity map. Let $\mathbf{B}_{[i,j]}$ denote the basis $\{\pi_i(\mathbf{b}_i), \ldots, \pi_i(\mathbf{b}_j)\}$ and let $\mathcal{L}_{[i,j]}$ denote the lattice spanned by $\mathbf{B}_{[i,j]}$.

2.2.2 Computational Problems

We will now cover some computational lattice problems that are useful in cryptography. Let us start off with the concept of a shortest lattice vector[2].

Definition 2.2.6 (Shortest vector)**.** *A non-zero lattice vector $\mathbf{v}$ whose norm is shorter than or equal to the norm of all non-zero lattice vectors in a lattice $\mathcal{L}$ is called a shortest vector. We denote its norm by $\lambda_1(\mathcal{L})$.*

Finding a shortest vector in a lattice, solving the Shortest Vector Problem (SVP), is a fundamental problem in lattice-based cryptography. The concept of a shortest vector is also generalized to successive minima.

Definition 2.2.7 (Successive minima)**.** *We call a shortest vector in a lattice the first successive minima and denote its length by $\lambda_1(\mathcal{L})$. A shortest possible vector which is linearly independent of the first successive minima is the second successive minima, with its length denoted by $\lambda_2(\mathcal{L})$. In general, given the first $k-1$ successive minima, a shortest possible lattice vector which is linearly independent of the first $k-1$ first successive minima is called the k-th successive minima and its length is denoted $\lambda_k(\mathcal{L})$; where $1 < k \leq d$.*

Let us define a couple of related problems. Let us start with the Closest Vector Problem (CVP).

Definition 2.2.8 (Closest Vector Problem (CVP))**.** *Given a lattice $\mathcal{L}$ and a target vector $\mathbf{t} \in \mathbb{R}^d$. The CVP is the problem of finding a lattice vector that is at least as close to $\mathbf{t}$ as all the other lattice vectors.*

Typically $\mathbf{t}$ is not a lattice vector. Unlike the SVP, the CVP mostly has a unique solution. A problem related to the CVP is the Bounded Distance Decoding (BDD) problem.

Definition 2.2.9 (α-Bounded Distance Decoding (BDD_α))**.** *Given a lattice $\mathcal{L}$, a small positive number α and a target vector $\mathbf{t} \in \mathbb{R}^d$. Given a guarantee that a closest vector $\mathbf{v}$ to $\mathbf{t}$ fulfills $\|\mathbf{v} - \mathbf{t}\| \leq \alpha\lambda_1(\mathcal{L})$, find a vector closest to $\mathbf{t}$.*

The BDD problem is thus essentially CVP with a guarantee that closest vector is unusually close. The smaller α is, the easier the BDD_α problem is. The BDD problem can be transformed to another version of SVP called unique-SVP.

Definition 2.2.10 (γ-unique Shortest Vector Problem (uSVP_γ))**.** *Given a lattice $\mathcal{L}$ and a guarantee that $\lambda_2(\mathcal{L}) > \gamma \cdot \lambda_1(\mathcal{L})$, for a positive constant $\gamma > 1$, find a shortest vector in $\mathcal{L}$.*

[2]Notice the phrasing here. There is never a unique shortest vector. If $\mathbf{v}$ is a shortest vector, then so is $-\mathbf{v}$, for example. For some lattices even more shortest vectors exist.

Unique-SVP is essentially SVP with the guarantee that a shortest vector is unusually short. Except that; given that $\mathbf{v}$ is a solution to unique-SVP, so is $-\mathbf{v}$; the solution is typically unique. The larger γ is, the easier uSVP_γ is.

There are approximate versions of many of the lattice problems. Let us define the approximate SVP.

Definition 2.2.11 (Approximate SVP)**.** *Given a lattice $\mathcal{L}$ and a factor γ. The γ-approximate SVP is to find a lattice vector $\mathbf{v} \in \mathcal{L}$ such that $\|\mathbf{v}\| \leq \gamma\lambda_1(\mathcal{L})$. We write SVP_γ to denote this problem.*

There are of course obvious corresponding approximate versions of some of the other lattice problems defined in this section, for example approximate CVP denoted CVP_γ.

2.2.3 Transforming LWE Into a Lattice Problem

Given an LWE instance on the matrix form of (1.1), there is an implied reduction modulo q, which means that we can find a vector $\boldsymbol{\lambda}$ such that

$$\mathbf{b} = \mathbf{sA} + \mathbf{e} + q \cdot \boldsymbol{\lambda}. \tag{2.4}$$

We can rewrite this as

$$\begin{pmatrix} \mathbf{s} & \boldsymbol{\lambda} \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{A} \\ \mathbf{0} & q\mathbf{I} \end{pmatrix} = \begin{pmatrix} \mathbf{0} & \mathbf{b} \end{pmatrix} + \begin{pmatrix} \mathbf{s} & -\mathbf{e} \end{pmatrix}. \tag{2.5}$$

We can thus solve LWE by solving the BDD problem looking for vectors close to the target vector $\mathbf{t} = \begin{pmatrix} \mathbf{0} & \mathbf{b} \end{pmatrix}$ in the lattice spanned by the basis

$$\mathbf{B} = \begin{pmatrix} \mathbf{I} & \mathbf{A} \\ \mathbf{0} & q\mathbf{I} \end{pmatrix}. \tag{2.6}$$

This approach was studied in [LP11, LN13]. Solving LWE by transforming the problem to unique-SVP was studied in [AFG14]. There is also another approach of solving the distinguishing problem in the dual lattice [MR09,Alb17], which is outside the scope of this thesis.

2.2.4 Lattice Reduction

This section discusses methods for finding short/close vectors in a lattice using lattice reduction. First, let us define a unimodular matrix.

Definition 2.2.12 (Unimodular matrix)**.** *A square matrix U is unimodular if it has integer coefficients and* $\det(U) = \pm 1$.

The basis for a lattice is not unique. Given a basis $\mathbf{B}$ and a unitary matrix $\mathbf{U}$, we can form a new basis as $\mathbf{B}' = \mathbf{UB}$. The general method for finding short vectors in a lattice of large dimension is to change the basis to a "better" one and then, if needed, search for an even shorter vector using this basis. A better basis roughly means a basis where the basis vectors are closer to orthogonal and are as short as possible. The process of finding a better basis is called lattice reduction.

Size Reduction

For a vector space an orthogonal basis can always be achieved through Gram-Schmidt orthogonalization, which we will briefly repeat. Given any basis $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ of a vector space, we can calculate an orthogonal basis $\mathbf{B}^\star = \{\mathbf{b}_1^\star, \ldots, \mathbf{b}_d^\star\}$ as

$$\mathbf{b}_i^\star = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{ij} \mathbf{b}_j^\star, \tag{2.7}$$

where

$$\mu_{ij} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^\star \rangle}{\langle \mathbf{b}_j^\star, \mathbf{b}_j^\star \rangle}. \tag{2.8}$$

Each new basis vector $\mathbf{b}_i^\star$ is created as the difference between $\mathbf{b}_i$ and the orthogonal projection of $\mathbf{b}_i$ on the plane spanned by the new basis vectors $\{\mathbf{b}_1^\star, \ldots, \mathbf{b}_{i-1}^\star\}$, making sure that $\mathbf{b}_i^\star$ is orthogonal against all the previously created new basis vectors.

The only problem with the Gram-Schmidt orthogonalization process is that the coefficients μ_{ij} are non-integers and thus the resulting basis vectors $\mathbf{b}_i^\star$ are not lattice vectors. The most obvious way of doing lattice reduction is using the Gram-Schmidt orthogonalization procedure, but rounding the coefficients to the closest integer. This process is called size-reduction.

The LLL Algorithm

Next, let us define the Lovász condition.

Definition 2.2.13 (Lovász condition)**.** *Given the basis* $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ *of a lattice, the corresponding Gram-Schmidt orthogonalization basis* $\mathbf{B}^\star = \{\mathbf{b}_1^\star, \ldots, \mathbf{b}_d^\star\}$ *and* $0 < \epsilon < 3/4$*. The Lovász condition is that*

$$\mu_{i,i-1}^2 \left\|\mathbf{b}_{i-1}^\star\right\|^2 + \left\|\mathbf{b}_i^\star\right\|^2 \geq (1-\epsilon) \left\|\mathbf{b}_{i-1}^\star\right\|^2, \tag{2.9}$$

for $1 < i \leq d$*.*

A basis is called Lenstra–Lenstra–Lovász (LLL)-ϵ reduced if it is size reduced and fulfills the Lovász condition for a certain value ϵ. If a certain value ϵ is implied we simply call the basis LLL-reduced.

The LLL algorithm achieves a basis which is size-reduced and fulfills the Lovász condition [LLL82]. The algorithm starts by size-reducing the basis. Then it checks for adjacent pairs of vectors where the Lovász condition is not fulfilled. Then it swaps the pair of basis vectors and size-reduces the basis again. This process is repeated until there are no pairs where the Lovász condition is not fulfilled and the basis is LLL-reduced.

The BKZ Algorithm

The LLL algorithm was improved and generalized in [Sch87], introducing the concept of a block size β, where $\beta = 2$ corresponds to the LLL algorithm. A basis is called BKZ-β reduced if it is LLL-reduced and for each $1 \leq j < d$, $\mathbf{b}_j^\star = \lambda_1(\mathcal{L}_{[j,k]})$, where $k = \min(j + \beta - 1, d)$.

There are many methods for finding short vectors in the projected lattices, but the two main practical ones are enumeration and sieving. Enumeration uses only a polynomial amount of space, but is super-exponential in time. It essentially consists of trying linear combinations of the basis vectors for a combination that is as short as possible[3]. The idea of pruning this process and skipping unlikely candidates was introduced in [SE94]. The idea of using extreme pruning, where many enumerations are done with each having a very low success probability, to drastically improve this process, was introduced in [GNR10]. Even though enumeration is asymptotically slow, it was fastest for practical instances until recently.

Sieving is exponential in time and space, and recently became faster than enumeration in practice. For the current state-of-the-art in practice, see [ADH+19]. Sieving will be discussed in more detail in Chapter 5.

The SVP Challenge is a way of comparing the performance of SVP solving algorithms [SVP]. A similar challenge specifically for LWE is the Darmstadt LWE Challenge [Dar]. The largest LWE instances are solved using the implementation from [ADH+19]. Using a GPU-implementation of [ADH+19], Ducas, Stevens and van Woerden have currently solved the largest SVP challenge instance of dimension 176 [DSvW20].

For an easy introduction to lattice-reduction, see [Må16]. For a more complete introduction to BKZ, see [Che14]. Notice that in both of these introductions enumeration was used inside the BKZ algorithm, since enumeration was the fastest algorithm in practice during the writing of both theses. See [ADH+19] for how to use sieving inside BKZ.

It was shown in [HPS11] that the number of calls to the SVP subroutine of BKZ is polynomial in the dimension d. The total (asymptotic) complexity of BKZ hence depends directly on the complexity of the SVP subroutine.

2.3 Combinatorial Approaches

The combinatorial approaches include the basic BKW algorithm, as introduced in [BKW00] for LPN and first analyzed for LWE in [ACF+15], and all later improvements of it. This approach is the main topic of this thesis and will be discussed in great detail in Chapters 6 to 9.

2.4 Surveys

There are a couple of good surveys on LWE solving algorithms that are much more thorough than this short introduction. For a survey on the concrete complexity of solving LWE using different approaches, see [APS15]. For a survey on the asymptotic complexity, see [HKM18]. For an asymptotic algorithm comparison covering the most recent improvements of the BKW algorithm, see Paper 5 [GJMS19b].

[3] Since we can estimate the size of a shortest vector we only need to try a limited number of linear combinations.

Chapter 3

Quantum Computation

> The book is too elementary, it starts off with the assumption that the reader does not even know quantum mechanics.
>
> — *Anonymous Post-doc at Bell Labs*

This chapter introduces quantum computation with a focus on Grover's algorithm and its generalizations, including amplitude amplification. The reader is assumed to know basic linear algebra. However, let us repeat the definition of a unitary matrix.

Definition 3.0.1 (Unitary Matrix)**.** *A complex, square matrix U is called unitary if its inverse U^{-1} is equal to its complex conjugate U^*.*

Equivalently a matrix U is called unitary if it maps a vector of norm 1 to a vector of norm 1.

3.1 Quantum Computation Basics

The presentation here is based on the lecture notes of de Wolf [dW19], which can be used for a much broader introduction to the area. Another great lecture series is Aaronson's [Aar18]. For an introductory book see [KLM17]. For the most complete book introduction covering both the necessary computer science and quantum mechanics, see [NC10]. A great non-technical introduction to quantum computation is the following podcast episode [Fri20].

A classic computer works with bits. A bit can take the value 0 or 1. A qubit in turn can be in a superposition between the two states $|0\rangle$ and $|1\rangle$,

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle , \tag{3.1}$$

where $\alpha_0, \alpha_1 \in \mathbb{C}$ and $|\alpha_0|^2 + |\alpha_1|^2 = 1$.

A register with n bits can be in any of 2^n distinct states. Denote these states by 0, 1, ..., $2^n - 1$. A register with qubits in turn can be in a superposition between the basis states $|0\rangle , |1\rangle , \ldots , |2^n - 1\rangle$;

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle , \tag{3.2}$$

where $\alpha_i \in \mathbb{C}$ and $\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1$. We can write the state as the column vector $|\psi\rangle = (\alpha_0 \cdots \alpha_{2^n-1})^T$.

3.1.1 Measurement

We can measure the state of a system of qubits (3.2). The system then collapses to one of the states. The probability of the system collapsing to the state $|i\rangle$ is

$$|\alpha_i|^2. \tag{3.3}$$

3.1.2 Unitary Transformation

We can apply also apply transformations to our qubits. The laws of quantum mechanics only allows us linear transformations that also keep the norm of the transformed state equal to 1. This means that every transformation of the state $|\psi\rangle = (\alpha_0 \cdots \alpha_{2^n-1})^T$ corresponds to multiplying a unitary matrix U by $|\psi\rangle$.

3.1.3 Entanglement

Consider a 2-qubit system in the state

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle). \tag{3.4}$$

Both the qubits can take the values $|0\rangle$ or $|1\rangle$ when measured. However, measuring one of them means knowing the value of the other one. When measuring, the outcome of the two qubits are not independent of each other, the qubits are entangled.

Quantum computation is essentially the art of, given input data, applying unitary transformations to the qubits in the correct way and then (with high probability) retrieving the correct answer when measuring the state.

3.2 Grover's Algorithm

We will cover Grover's algorithm in some detail. To shorten the exposition a bit we will cover what the different unitary transformations do, but not specify the different unitary matrices.

While Grover's algorithm does not lead to an exponential speed-up like Shor's algorithm, it can be used as a building block to speed-up many algorithms.

Consider an unsorted database with $N = 2^n$ elements on the form $x \in \{0,1\}^N$. Let x_i denote element number i. The problem is to find an i such that $x_i = 1$. Let us initially assume that there are precisely t such values i. Classically, on average, we need to look through the whole database in time $\Theta(N/t)$ to find a desirable element. We will now show how this can be drastically improved quantumly. Let us define the good state

$$|G\rangle = \frac{1}{\sqrt{t}} \sum_{i:x_i=1} |i\rangle ,$$

and the bad state

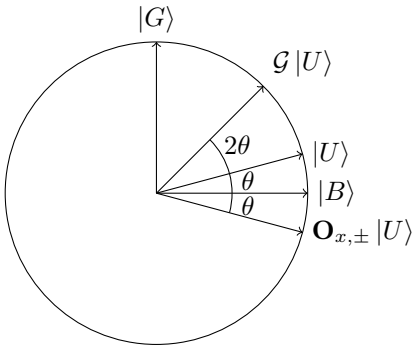


Figure 3.1: Illustration of the first Grover iteration.

$$|B\rangle = \frac{1}{\sqrt{N-t}} \sum_{i:x_i=0} |i\rangle\,.$$

Let us begin in the zero-state $|0\rangle$. First we apply a unitary transformation that turns the state into the uniform superposition state and write it as linear combination of the good and the bad state

$$|U\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle = \sqrt{\frac{t}{N}}\,|G\rangle + \sqrt{\frac{N-t}{N}}\,|B\rangle = \sin(\theta)\,|G\rangle + \cos(\theta)\,|B\rangle\,,$$

where $\theta = \arcsin(\sqrt{t/N})$. Typically $t \ll N$, which means that $|U\rangle$ is almost parallel with $|B\rangle$. Thus, if we measure immediately, the probability, $1 - t/N$, is overwhelming that the state collapses to a non-solution $x_i = 0$. Grover's algorithm works by repeatadly doing a reflection of the state in $|B\rangle$, followed by a reflection of the state in $|U\rangle$, until the state is close to parallel with $|G\rangle$ instead, at which time measurement is likely to lead to a solution $x_i = 1$.

Figure 3.1 illustrates the first Grover iteration. Here $\mathbf{O}_{x,\pm}$ is a unitary that reflects the state in $|B\rangle$ and $\mathcal{G}$ is the unitary for the whole Grover iteration.

The first iteration increases the angle to 3θ. In general, iteration k increases the angle from $(2k-1)\theta$ to $(2k+1)\theta$. To get a success probability of 1 we would like to make $\tilde{k} = \pi/(4\theta) - 1/2$ iterations. Since we cannot make a non-integer number of iterations, we set the number of iterations to $k = \lfloor \pi/(4\theta) - 1/2 \rceil$ and thus achieve an angle as close as possible to $\pi/2$. If we denote the success probability as P_k, then we can upper limit the risk of failure as

$$\begin{aligned} 1 - P_k &= 1 - \sin^2((2k+1)\theta) = \cos^2((2k+1)\theta) = \cos^2((2\tilde{k}+1)\theta + 2(k-\tilde{k})\theta) \\ &= \cos^2(\pi/2 + 2(k-\tilde{k})\theta) = \sin^2(2(k-\tilde{k})\theta) \leq \sin^2(\theta) = \frac{t}{N}. \end{aligned}$$

If we are unlucky and do not find a solution, we can just re-run Grover's algorithm until we find a solution.

For small angles α we have $\arcsin\alpha > \alpha$. Thus, $k < \pi/(4\theta) \leq \pi/4\sqrt{N/t} = \mathcal{O}(\sqrt{N/t})$.

Grover's algorithm is asymptotically optimal [BBBV97]. There are many possible generalizations of the algorithm as described above. If we know t, then it is possible to tweak the algorithm such that the success probability when measuring is 1 [dW19, Exercise 7.7]. While the description above requires that N is a power of 2, the algorithm can be generalized to handle any size N [BBHT98]. If t is not known in advance, it is still possible to achieve $\mathcal{O}(\sqrt{N/t})$ time complexity by trying different values of k in a smart way [BBHT98].

One important generalization of Grover's algorithm which we will cover next is amplitude amplification.

3.3 Amplitude Amplification

Let us generalize Grover's algorithm a bit to what is called amplitude amplification [BHMT02]. We have a function $\chi : \mathbb{Z} \to \{0, 1\}$ and search for values $z \in \mathbb{Z}$ such that $\chi(z) = 1$.

Let $\mathcal{A}$ be an algorithm that turns the starting state $|0\rangle$ into a state $|U\rangle$ where measuring results in a probability p of success. Let us define good and bad states, $|G\rangle$ and $|B\rangle$, like for the Grover setting. Now let us apply the following algorithm.

1. Set-up using $\mathcal{A}$ to form the state $|U\rangle = \mathcal{A}\,|0\rangle$.
2. Repeat k times:
   (a) Reflect the current state in $|B\rangle$.
   (b) Reflect the current state in $|U\rangle$.
3. Measure the state and check classically if the found element z is a solution.

The number k is optimized the same way as for Grover's algorithm and is $\mathcal{O}(1/\sqrt{p})$. Amplitude amplification makes it possible to speed-up any classical algorithm that has a success probability of p by a factor of $\sqrt{p}$. Grover's algorithm covers the special case where the classical algorithm corresponds to picking an element from the database randomly.

In Paper 3 [KMPM19a], we use both Grover's algorithm and amplitude amplification to speed-up classical algorithms for lattice sieving, see Chapter 5. We also use Grover's algorithm to speed-up BKW-type algorithms in Papers 5 and 6 [GJMS19b, Må19], see Chapter 7 for some details.

3.4 Shor's Algorithm

Let us cover how to solve the IFP of Definition 1.2.1 using Shor's algorithm. Given a large positive integer n that we want to factor. We can handle an arbitrary number n and do not need to limit the discussion to products of two primes p and q like in Definition 1.2.1. We can assume without loss of generality that n is odd and not a prime power[1]. In other words, let us assume that n is the product of two, odd co-prime factors.

[1] Testing both these properties is trivial.

The goal of Shor's algorithm is to find a non-trivial root b of 1 modulo n. If we find such a (non-trivial) root we get

$$b^2 = 1 \mod n \Leftrightarrow (b-1)(b+1) = 0 \mod n \Leftrightarrow (b-1)(b+1) = mn, \quad (3.5)$$

where m is a positive integer. Now n has $\gcd(b-1, n)$ and $\gcd(b+1, n)$ as non-trivial factors[2]. Let us briefly cover the steps of how Shor's algorithm achieves this.

1. Pick a random number $1 < a < n$.
2. Calculate $\gcd(a, n)$. If this number is different from 1, then we have found a factor and are done.
3. Given the function $f(x) = a^x \mod n$. Find the smallest positive integer r, such that $f(r) = 1$. In other words calculate the period of f. (Since $\gcd(a, n) = 1$ we know that the period is well-defined and equal to the order of a in $\mathbb{Z}_n^\star$.)
4. If r is odd, go back to step 1 and start over.
5. If $a^{r/2} = \pm 1 \mod n$, go back to step 1 and start over.
6. Now $\gcd(a^{r/2} + 1, n)$ and $\gcd(a^{r/2} - 1, n)$ are non-trivial factors of n and we are done.

The only computationally heavy part is step 3. For classical algorithms this step is not easier than the original problem. However, this step can be performed in polynomial time on a quantum computer using the quantum Fourier transform. The details of how this is done is outside the scope of this thesis.

3.4.1 The DLP

Now consider the DLP of Definition 1.2.2. Consider the function $f(a) = g^a h^{-1}$. Using the quantum period finding of step 3 above we find an r, such that

$$f(r) = g^r h^{-1} = 1 \Leftrightarrow g^r = h, \quad (3.6)$$

which is a solution to the DLP. It is no coincidence that Shor's algorithm can be used to solve both the IFP and DLP. Both the problems can be stated as hidden subgroup problems for finite abelian groups, which (a slight generalization of) Shor's algorithm can solve [Kit96, ME99].

[2]The idea is similar to how the classical sieving algorithms work.

Chapter 4

Code-based Cryptanalysis

It's Not a Bug, It's a Feature.

— *Unclear origin*

4.1 Coding Theory Basics

Coding theory is the study of, among other things, how to reliably communicate over a channel whose noise creates errors in the transmitted messages. It turns out that the techniques from coding theory can be used for cryptographic purposes. Some basic coding theory is needed to understand code-based cryptography and cryptanalysis. Let us first define a (binary) linear code.

Definition 4.1.1 (Linear code)**.** *An [n, k] (binary) linear code $\mathcal{C}$ is a k-dimensional subspace of $\mathbb{F}_2^n$.*

Analogously, we define a q-ary linear code as a k-dimensional subspace of $\mathbf{F}_q^n$. Let us also define the generator matrix and parity-check matrix of a linear code. For this chapter we only work with binary codes.

Definition 4.1.2 (Generator matrix)**.** *A generator matrix $\mathbf{G}$ of a linear code $\mathcal{C}$ is a $k \times n$ matrix in $\mathbb{F}_2^{k \times n}$ whose rows form a basis of $\mathcal{C}$.*

In other words, $\mathbf{G}$ is a matrix such that each codeword $\mathbf{c} \in \mathcal{C}$ can be written as $\mathbf{c} = \mathbf{mG}$, for some vector $\mathbf{m} \in \mathbb{F}_2^k$. If the first k columns of G form an identity matrix, then we say that $\mathbf{G}$ is in systematic form. We can always turn $\mathbf{G}$ systematic through Gaussian elemination.

Definition 4.1.3 (Parity-check matrix)**.** *A parity-check matrix of a linear code $\mathcal{C}$ is a matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$, whose kernel is the linear code $\mathcal{C}$.*

In other words, $\mathbf{H}$ is a matrix such that $\mathbf{cH}^T = \mathbf{0}$, for all $\mathbf{c} \in \mathcal{C}$. Let us also define the support of, the Hamming weight of and the Hamming distance between binary vectors.

Definition 4.1.4 (Support)**.** *The support of an n-bit binary vector $\mathbf{v}$ is supp($\mathbf{v}$) $= \{i : v_i = 1, 1 \leq i \leq n\}$.*

Definition 4.1.5 (Hamming weight)**.** *The Hamming weight of an n-bit binary vector* $\mathbf{v}$ *is* $w_H(\mathbf{v}) = |supp(\mathbf{v})|$.

Definition 4.1.6 (Hamming distance)**.** *The Hamming distance between two n-bit binary vectors* $\mathbf{v}$ *and* $\mathbf{w}$ *is* $d_H(\mathbf{v}, \mathbf{w}) = w_H(\mathbf{v} + \mathbf{w})$.

Coding theory is about reliable communication over a noisy channel that adds errors to sent messages. This is done using linear codes. By turning an original k-bit message $\mathbf{m}$ into an n-bit codeword $\mathbf{c} = \mathbf{mG}$, for a generator matrix $\mathbf{G}$, redundancy gets added to the message. The noisy channel adds a binary error vector $\mathbf{e}$ to the codeword turning the recieved vector into $\mathbf{r} = \mathbf{c} + \mathbf{e}$. If the Hamming weight of $\mathbf{e}$ does not exceed a threshold t[1], the errors can be corrected and the codeword $\mathbf{c}$, and hence $\mathbf{m}$, can be retrieved. If we pick the linear code and parity-check matrix $\mathbf{H}$ in a smart way the decoding can be done efficiently.

This introduction to coding theory is very brief in nature. For a much more thorough introduction to the area, see for example [LC04, Bos99].

4.2 Code-based Cryptography

We can use linear codes for cryptographic purposes too. Unlike in coding theory, in cryptography we intentionally add errors to the codewords. To make a software analogy, while errors are viewed as a bug that needs to be fixed in coding theory, in cryptography they are considered a feature that makes encryption possible.

Asymmetric encryption using coding theoretic methods was originally suggested in 1978 by McEliece, making it almost as old as RSA [McE78]. Figure 4.1 shows a simplified illustration of encryption using the McEliece cryptosystem. To encrypt a secret message $\mathbf{m}$, Alice first multiplies it by the generator matrix $\mathbf{G}$ of a linear code to turn it into a codeword in the linear code $\mathbf{c} = \mathbf{mG}$. Then she intentionally adds an error vector $\mathbf{e}$ of a suitable Hamming weight t to form $\mathbf{r} = \mathbf{mG} + \mathbf{e}$. Observing $\mathbf{r}$, it is computationally infeasible for Eve to correct the t errors and decrypt the message. This is related to the fact that decoding of general linear codes is NP-complete [BMv78]. By having access to a parity-check matrix of a certain form, Bob can easily correct the errors to retrieve $\mathbf{c}$ and thus the original message $\mathbf{m}$.

In the original McEliece scheme Goppa codes were used [Gop70], allowing efficient decoding [Pat75]. In principle any code that allows efficient decoding can be considered[2]. One large disadvantage of code-based encryption schemes is the large public key $\mathbf{G}$.

Two more recent and popular versions of the McEliece cryptosystem are based on Quasi-Cyclic (QC) Moderate Density Parity-Check (MDPC) codes [MTSB13] and QC Low Density Parity-Check (LDPC) codes respectively [BC07, BBC08]. These very sparse codes reduce the size of the public key. As originally introduced they are vulnerable towards reaction attacks that take advantage of dependencies between the secret key and the risk of decryption failures [GJS16, FHS+17]. Counter-measures against these types of attacks need

[1] whose value depends on n, k and the linear being code used.

[2] Notice however that most codes with structure that allows efficient decoding suffer from efficient attacks taking advantage of that structure.

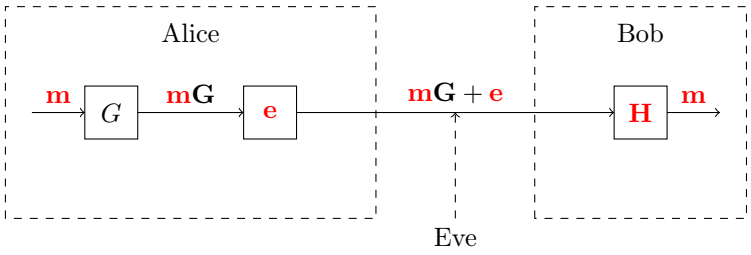


Figure 4.1: A simplified illustration of encryption using the McEliece cryptosystem.

to be implemented for all code-based crypto systems that contain decryption failures.

For a much more thorough introduction to code-based cryptography, see for example [Lö14, OS09].

4.2.1 Soft McEliece

In [BSC16] a version of the McEliece with real-valued noise called soft McEliece was suggested. Here encryption is done by first transforming a codeword $\mathbf{c}$ to $\hat{\mathbf{c}}$, where $\hat{c}_i = (-1)^{c_i}$, for $1 \leq i \leq n$. Next we add Additive White Gaussian Noise (AWGN) to form the encrypted message

$$\mathbf{r} = \hat{\mathbf{c}} + \mathbf{w}. \tag{4.1}$$

Here AWGN means that the noise terms w_i are Independent and Identically Distributed (IID) and $w_i \sim \mathcal{N}(0, \sigma)$, for a suitable noise level σ.

4.3 Stern's Algorithm

The most promising algorithms for decoding of general linear codes are based on Information Set Decoding (ISD). Given the task of decoding the recieved vector

$$\mathbf{r} = \mathbf{uG} + \mathbf{e}, \tag{4.2}$$

to retrieve the message $\mathbf{u}$, where $w_H(\mathbf{e}) \leq w$. Let us first define an information set. Write the generator matrix of a linear code as

$$\mathbf{G} = \left[\mathbf{g}_1^T \cdots \mathbf{g}_n^T\right]. \tag{4.3}$$

Definition 4.3.1 (Information set)**.** *A subset $I \subset \{1, \ldots, n\}$ of size k is an information set to an $[n, k]$ linear code if the corresponding vectors $(\mathbf{g}_i)_{i \in I}$ are linearly independent.*

The first ISD algorithm is Prange's algorithm [Pra62]. In Prange's algorithm we perform a random permutation π on the system (4.2) forming

$$\mathbf{r}' = \mathbf{uG}' + \mathbf{e}'. \tag{4.4}$$

Assume that the first k indices of the permuted system constitute an information set I; if not, then we can perform new permutations until they do. If we also assume that the error terms in (4.4) are all 0, then we can solve for $\mathbf{u}$ as

$$\hat{\mathbf{u}} = \mathbf{r}'(\mathbf{G}')^{-1}. \tag{4.5}$$

Whether the solution is correct can be checked by testing if $d_H(\hat{\mathbf{u}}\mathbf{G}, \mathbf{r}) \leq w$. If the solution is incorrect we can make a new permutation and repeat the process until we find a solution. An improvement of Prange's algorithm was made by Lee and Brickell [LB88]. The idea here is to assume that the error vector in (4.4) has a Hamming weight less than or equal to p, where $0 \leq p \leq w$. Then we can calculate

$$\hat{\mathbf{u}} = (\mathbf{r}' + \mathbf{e}')\mathbf{G}'^{-1}, \tag{4.6}$$

for all such error patterns $\mathbf{e}'$ and check if any of them is correct. A big improvement was made by Stern in [Ste89]. Similar ideas were also introduced in [Dum91]. Here a speed-up is achieved using the birthday paradox. We use the same type of parameter p as in Lee-Brickell, but also another parameter $l \in \{0, \ldots, n-k\}$. Also here we start by performing a permutation π to the positions (4.2) forming a system of the form (4.4), such that the first k positions constitute an information set. Next perform Gaussian elemination such that the generator matrix is on the systematic form

$$\mathbf{G}_{\text{sys}} = \begin{pmatrix} \mathbf{I} & \mathbf{Q} & \mathbf{J} \end{pmatrix}, \tag{4.7}$$

where I is a $k \times k$ identity matrix, Q is a $k \times l$ matrix and J is a $k \times (n-k-l)$ matrix. Next we enumerate all vectors $\mathbf{u}$ of size $k/2$ and Hamming weight less than or equal to $p/2$ and form the list $\mathcal{L}_1$ of vectors on the form $\mathbf{c}_1 = (\mathbf{u}||\mathbf{0})\mathbf{G}_{\text{sys}}$. We also form the list $\mathcal{L}_2$ of vectors on the form $\mathbf{c}_2 = (\mathbf{0}||\mathbf{u})\mathbf{G}_{\text{sys}}$. Collide the two lists and create a new list $\mathcal{L}_1 \diamond \mathcal{L}_2$ of all vectors $\mathbf{c} = \mathbf{c}_1 + \mathbf{c}_2$, that are equal to $\mathbf{0}$ on positions $k+1$ to $k+l$. For each codeword in this list we test if its Hamming distance to $\mathbf{r}'$ is less than or equal to w.

Typically, a good parameter choice here is $|\mathcal{L}_1| = |\mathcal{L}_2| = 2^l$. Then we get the expected list size

$$\mathbb{E}(|\mathcal{L}_1 \diamond \mathcal{L}_2|) \approx |\mathcal{L}_1| = |\mathcal{L}_2| = 2^l. \tag{4.8}$$

The cost of building the two lists and colliding them are all equal to $\mathcal{O}(2^l)$. We manage to test the viability of $(2^l)^2 = 2^{2l}$ codewords in the process, leading to improved performance over Lee-Brickell[3]. ISD algorithms have since the seminal work of Stern been improved in a number of follow-up papers classically [CC98, FS09, BLP11, MMT11, BJMM12, MO15, BM17], and also quantumly [Ber10, KT17, Kir18].

4.4 Soft Stern

In Paper 4 we developed an algorithm for solving the problem of decoding linear codes with soft noise on AWGN form (4.1) [GJMS17b, GJMS19a]. When using soft noise an important concept is the Log-Likelihood Ratio (LLR) defined as

[3] We do however only find codewords where positions $k+1$ to $k+l$ are equal to 0, lowering the magnitude of the improvement.

$$L_i = \ln\left[\frac{p(r_i|c_i=0)}{p(r_i|c_i=1)}\right], \tag{4.9}$$

where $p(r_i|c_i)$ is the PDF of r_i conditioned on c_i. In our setting the LLR can be rewritten as[4]

$$L_i = \frac{2r_i}{\sigma^2}. \tag{4.10}$$

For each position we can translate the soft information to the most likely hard choice as

$$\operatorname{sgn}(r_i) = \begin{cases} 1, & \text{if } r_i \leq 0, \\ 0, & \text{otherwise.} \end{cases} \tag{4.11}$$

We can also calculate the probability of this guess being correct as

$$p_i = \Pr[c_i = \operatorname{sgn}(r_i)|r_i] = \frac{1}{1+e^{-|L_i|}}. \tag{4.12}$$

The larger $|L_i|$, the higher the probability is of correctly guessing the position by making a hard decision. We say that a position is more reliable the larger $|L_i|$ is.

We can now solve the decoding problem (4.1) using a Stern-type approach but taking advantage of the soft information. Instead of picking a random permutation, we pick a permutation π such that

1. The first $k+l$ positions are the most reliable.
2. The first k columns of $\pi(\mathbf{G})$ are linearly independent.
3. $\prod_{i=1}^{(k+l)/2} p_i \approx \prod_{i=1}^{(k+l)/2} p_i$.

Here we can apply a transformation such that the first $k+l$ positions all have $r_i \geq 0$ meaning that 0 is the most probable value for each position c_i. The probability of a bit pattern with 1s in the index set J for the first $(k+l)/2$ positions, being all correct, can now be written as

$$\exp\left(-\sum_{j\in J} L_j\right) \prod_{i=1}^{(k+l)/2} p_i. \tag{4.13}$$

We get an analogous expression for the next $(k+l)/2$ positions.

Instead of enumerating all bit patterns up to a certain Hamming weight when forming the lists $\mathcal{L}_1$ and $\mathcal{L}_2$, we enumerate the bit patterns with the highest probabilities according to (4.13).

The soft version has two advantages over the standard Stern algorithm. First of all, the first $(k+l)/2$ positions individually have a very large probability of containing a 0 (after transformation). Secondly, we can cover much more probable error patterns by enumerating in order of probability instead of purely in order of Hamming weight.

[4]Notice that our algorithm does not require that the noise is AWGN, we just need to have it on LLR form.

If more iterations of the soft Stern algorithm are needed, we do not just pick a uniformly random new permutation, we add/remove positions depending on their reliability, see Paper 4 for more details [GJMS19a].

The idea of decoding random linear codes using ordered statistics has been considered before, in for example [FS95] and how to efficiently enumerate the error patterns was considered in [VF01]. A Stern-type approach to soft information decoding was considered in [VF04]. The way we combined these approaches in Paper 4 is novel [GJMS19a].

We found a couple of applications of the soft Stern algorithm, including the following.

1. We showed that the soft McEliece system of [BSC16] is severely broken. For the suggested parameters for both 80 and 128 bits of security with high probability the k most reliable bits have 0 errors, turning decoding using the soft Stern algorithm into simple linear algebra. Increasing the noise level σ to achieve a better security level would increase the risk of decoding failure to an unacceptably high level.

2. The algorithm can be used in side-channel attacks where the general decoding of unstructured codes with soft noise appears [PBY17, PM16] and would lead to an improvement in that setting.

3. In [BMPC16] a hybrid decoder was suggested for decoding linear codes with soft information. The idea here is to use an efficient decoder first and in the case of decoding failure fall back on using a general decoder for linear codes with soft information. Also here our algorithm would lead to improved performance.

Chapter 5

Lattice Sieving

This chapter discusses lattice sieving, the asymptotically and practically fastest type of algorithm for solving the SVP. It is the topic of Paper 3 [KMPM19a] and used as a tool to improve BKW in Papers 5 and 6 [GJMS19b, Må19].

5.1 Lattice Sieving

Lattice sieving algorithms were introduced in the seminal paper [AKS01], as the first exponential algorithms for solving the SVP.

Given a basis $\mathbf{B}$ of a lattice $\mathcal{L}$. Lattice sieving approaches start off by producing an exponential amount of lattice vectors by forming linear combinations of the basis vectors [Kle00]. Next given a large list L of vectors we want to find pairs of vectors in L that produce slightly shorter vectors when combined. We want to find at least $|L|$ such pairs to keep the list size.

According to the so called sieving heuristic, which is widely adopted in the literature, we assume that our vectors are approximately of equal length. We also assume that the vectors, when scaled to length 1, are uniformly distributed on the unit sphere. The analysis of heuristic algorithms for lattice sieving is thus done on uniformly distributed vectors on the unit sphere.

If for two unit vectors $\mathbf{x}, \mathbf{y}$ on the unit sphere $\langle \mathbf{x}, \mathbf{y} \rangle \geq \alpha$, for $0 \leq \alpha \leq 1$, we say that the vectors are α-close. Let us now define a lattice sieving iteration.

Definition 5.1.1 (Lattice sieving iteration)**.** *Given a list L of vectors on the unit sphere and a constant $0 \leq c \leq 1$. One iteration of lattice sieving refers to finding $|L|$ pairs $\mathbf{x}_1, \mathbf{x}_2 \in L$, such that $\mathbf{x}_1, \mathbf{x}_2$ are c-close.*

Normally we are interested in the $c = 1/2$ case, which means that for each sieving iteration we just slightly decrease the length of the vectors. Let us focus on it for this subsection.

You can show that the probability of two uniformly random vectors being α-close is $\tilde{\mathcal{O}}((1-\alpha^2)^{d/2})$. For the difference between two vectors $\mathbf{x}$ and $\mathbf{y}$ to be less than or equal to 1, we need that $\langle \mathbf{x}, \mathbf{y} \rangle \geq 1/2$, which happens with probability $\tilde{\mathcal{O}}((1-(1/2)^2)^{d/2}) = \tilde{\mathcal{O}}((3/4)^{d/2})$. Since a list of size $|L|$ can produce $\Theta(|L|^2)$ pairs of vectors, we (heuristically) need a list size of

$$|L| = \tilde{\mathcal{O}}\left(\left(\frac{4}{3}\right)^{d/2}\right) = 2^{0.2075d+o(d)}, \tag{5.1}$$

to be able to produce a new list of slightly shorter vectors. In [NV08] the new list was simply created by combining every possible pair and keeping the shortest vectors, leading to a time complexity of $2^{0.4150d+o(d)}$. It is also shown in [NV08] that the we only need to do a polynomial number of iterations to find the shortest lattice vector, leading to a total time of $2^{0.4150d+o(d)}$. This approach is called the Nguyen-Vidick (NV) sieve.

A different type of sieving algorithm was introduced in [MV10]. Instead of starting with a large list and comparing all pairs, they start with an empty list L. Then they sample vectors $\mathbf{v}$ and before $\mathbf{v}$ gets added to the list, they try to shorten $\mathbf{v}$ and the vectors in L by adding/subtracting list vectors to/from $\mathbf{v}$ and vice versa. This sieving algorithm is called the Gauss sieve[1]. Asymptotically this sieve performs identically with the NV-sieve, but it is faster in practice.

The asymptotic improvements of [NV08, MV10] are due to more efficient ways of doing the sieving iterations of Definition 5.1.1, using Nearest Neighbor Search (NNS) techniques. NNS techniques refers to methods of, given a lattice vector $\mathbf{x}_1 \in \mathcal{L}$, find a c-close vector $\mathbf{x}_2 \in \mathcal{L}$ faster than by checking all $|\mathcal{L}|$ lattice elements. In this work we will only discuss the currently asymptotically fastest sieving algorithms.

The current state-of-the-art sieving algorithms with provable complexities take $2^{d+o(d)}$ time and space complexity classically [ADRS15] and $2^{0.5d+o(d)}$ space and $2^{0.9532d+o(d)}$ time quantumly [ACKS20]. We will focus on the heuristically fastest algorithms. Partly because the current state-of-the-art implementations of SVP solving algorithms use heuristic sieving algorithms [ADH+19]. Partly because the derived heuristic complexities of these algorithms agree quite well with performance in practice.

For a much more thorough introduction to the topic of lattice sieving, see [Laa15].

5.2 Locality-Sensitive Filtering

The asymptotically fastest sieving algorithm uses so called Locality-Sensitive Filtering (LSF) [BDGL16]. Let us use the presentation of the algorithm from Appendix B of Paper 3 [KMPM19a].

Assume that we have a large list L of at least size $|L| = \tilde{\mathcal{O}}((1/(1-c^2))^{d/2})$ vectors uniformly distributed on the unit sphere and want to find $|L|$ c-close pairs of vectors from L.

The idea of LSF is to create a large amount of filter vectors uniformly spread out over the unit sphere. For each filter vector we map all vectors $\mathbf{x} \in L$ that are α-close to that filter vector. When we query a vector $\mathbf{q} \in L$ to find a close vector $\mathbf{x} \in L$, we first find all filter vectors $\mathbf{v}$ that are β-close to $\mathbf{q}$ and then we only iterate over vectors $\mathbf{x}$ that are α-close to one of these filter vectors $\mathbf{v}$. We use $\mathcal{V}$ to denote the set of filter vectors.

[1] Since for all pairs $(\mathbf{v}, \mathbf{w})$ in L we have $\|\mathbf{v} \pm \mathbf{w}\| \geq \max(\|\mathbf{v}\|, \|\mathbf{w}\|)$. This is what a pair of vectors being Gauss reduced means.

One can show that the following conditional probability applies for the uniformly random triplet of vectors $(\mathbf{x}, \mathbf{v}, \mathbf{q})$ on the unit sphere:

$$\Pr(\langle \mathbf{q}, \mathbf{x} \rangle \geq c, \langle \mathbf{q}, \mathbf{v} \rangle \geq \beta, \langle \mathbf{x}, \mathbf{v} \rangle \geq \alpha | \langle \mathbf{q}, \mathbf{x} \rangle \geq c) \tag{5.2}$$

$$= \frac{\left(\det \begin{pmatrix} 1 & \alpha & \beta \\ \alpha & 1 & c \\ \beta & c & 1 \end{pmatrix}\right)^{d/2}}{\left(\det \begin{pmatrix} 1 & c \\ c & 1 \end{pmatrix}\right)^{d/2}}. \tag{5.3}$$

Thus the number of filters we need is

$$|\mathcal{V}| = \frac{1}{\Pr(\langle \mathbf{q}, \mathbf{x} \rangle \geq c, \langle \mathbf{q}, \mathbf{v} \rangle \geq \beta, \langle \mathbf{x}, \mathbf{v} \rangle \geq \alpha | \langle \mathbf{q}, \mathbf{x} \rangle \geq c)} \tag{5.4}$$

$$= \frac{\left(\det \begin{pmatrix} 1 & c \\ c & 1 \end{pmatrix}\right)^{d/2}}{\left(\det \begin{pmatrix} 1 & \alpha & \beta \\ \alpha & 1 & c \\ \beta & c & 1 \end{pmatrix}\right)^{d/2}}. \tag{5.5}$$

For each $\mathbf{x} \in L$ we want to find all α-close filters. Here we assume that the time needed to find the filters is equal to the number of filters, in other words $|\mathcal{V}| \cdot (1 - \alpha^2)^{d/2}$. In [BDGL16] they discuss how to realistically achieve this. Thus, the time needed to map all vectors in L to all relevant filters is

$$T_{\text{prep}} = |L| \cdot |\mathcal{V}| \cdot (1 - \alpha^2)^{d/2}. \tag{5.6}$$

The memory needed to store the data structure is also equal to T_{prep}. When we query a vector $\mathbf{q} \in L$ to find a c-close vector $\mathbf{x}$ we first find all β-close filters $\mathbf{v}$ and then we look through all α-close list vectors $\mathbf{x}$ connected to any such filter. For each such $\mathbf{x}$ we check if it is c-close to $\mathbf{q}$. Thus, the time to query a vectors $\mathbf{q}$ is $|\mathcal{V}| \cdot (1 - \beta^2)^{d/2} \cdot |L| \cdot (1 - \alpha^2)^{d/2}$. The time to query all vectors in L is therefore

$$T_{\text{query}} = |L| \cdot |\mathcal{V}| \cdot (1 - \beta^2)^{d/2} \cdot |L| \cdot (1 - \alpha^2)^{d/2}. \tag{5.7}$$

The sieving iterations problem described above corresponds to setting $c = 0.5$. Letting $\alpha = \beta = 0.5$ minimizes the time complexity leading to a time and space complexity of $2^{0.2925d+o(d)}$. By letting $\alpha = 0.25$ and $\beta = 0.5$ we get the best possible time complexity of $2^{0.3685d+o(d)}$ while not increasing the space complexity above $2^{0.2075d+o(d)}$.

As is shown [BGJ15,LdW15,Laa15], it is possible to achieve the optimal time complexity of $2^{0.2925d+o(d)}$ while only using a space complexity of $2^{0.2075d+o(d)}$. This requires using the NV sieve, which is slower than the Gauss sieve in practice. This is done by generating the relevant filters on the fly every time we query a vector $\mathbf{q}$, instead of keeping them all at once in memory.

5.2.1 Quantum Speed-up

It is possible to speed-up the sieving iterations using LSF if we have a quantum computer. Building of the filter structure is not faster than in (5.6) quantumly, but the query cost (5.7) can be improved. For each query we first need to find all $|L| \cdot (1 - \beta^2)^{d/2}$ adjacent close filters. Then we can speed-up the search for

a c-close vector connected to these filters using Grover's algorithm. Thus, the time it takes to query a vector $\mathbf{q}$ is

$$|\mathcal{V}| \cdot (1-\beta^2)^{d/2} + \sqrt{|\mathcal{V}| \cdot (1-\beta^2)^{d/2} \cdot |L|(1-\alpha^2)^{d/2}}, \tag{5.8}$$

leading to a total query time of

$$T_{\text{query}} = |L| \left(|\mathcal{V}| \cdot (1-\beta^2)^{d/2} + \sqrt{|\mathcal{V}| \cdot (1-\beta^2)^{d/2} \cdot |L|(1-\alpha^2)^{d/2}} \right). \tag{5.9}$$

Optimizing for time we get a time and space complexity of $2^{0.2653d+o(d)}$. Unlike the classial case, the optimal quantum time complexity is not possible to achieve without using an equal amount of memory. As is shown in Paper 3, it is possible to achieve a time-memory tradeoff curve between the classical memory and time exponents of $(0.2075, 0.2925)$ and the quantum exponents of $(0.2653, 0.2653)$, see Figure 1 of the full version of the paper [KMPM19b].

5.3 k-sieving

The reader is assumed to know the basic concepts in graph theory. We will repeat the definitions of an induced subgraph and a clique though.

Definition 5.3.1 (Induced subgraph)**.** *Given an undirected graph $G = (V, E)$ with vertices V and edges E. A subset of the vertices together with all edges between these vertices is an induced subgraph.*

Definition 5.3.2 (Clique)**.** *An induced subgraph which is complete is a clique. If the clique has k elements it is called a k-clique.*

The problem of finding a single k-clique in a graph and the problem of listing all k-cliques in a graph are considered important problems in graph theory

When doing lattice sieving we do not have to limit ourselves to looking for pairs of vectors resulting in new short vectors. More generally we want to find k-tuples of vectors $(x_1, x_2, \ldots, x_k)$, such that $|x_1 + \cdots + x_k| \leq 1$[2]. This was originally studied in [BLS16]. It is shown in [HK17, Theorem 3] that we need a list size of

$$|L| = \tilde{\mathcal{O}} \left(\left(\frac{k^{\frac{k}{k-1}}}{k+1} \right)^{\frac{d}{2}} \right), \tag{5.10}$$

or more, to find $|L|$ such k-tuples. The required list size from (5.10) decreases with k. This comes at a cost of increased time complexity, leading to a tradeoff between time and space.

Given a small constant $\epsilon > 0$. It is shown in [HK17] that almost all good k-tuples can be found by finding the k-tuples that satisfy

[2] In practical implementation we of course look at all 2^k combinations $x_1 \pm \cdots \pm x_k$. Asymptotically speaking we get the same results by just considering additions of all the vectors. The techniques explained in this subsection can trivially be translated to any of the 2^k different combinations.

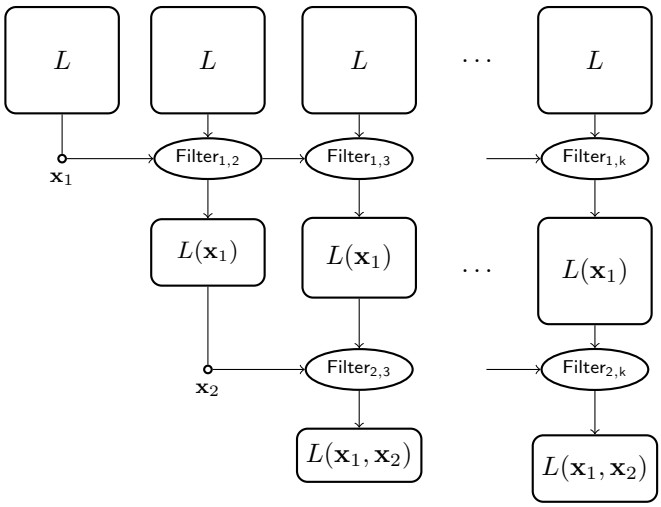


Figure 5.1: The algorithm of Herold et al. [HKL18] for finding good k-tuples.

$$|\langle \mathbf{x}_i, \mathbf{x}_j \rangle + 1/k| < \epsilon, \tag{5.11}$$

for all $i \neq j$. These k-tuples have a geometrical interpretation. Let the vectors in L be nodes in a graph. Let there be an edge between two nodes $\mathbf{x}, \mathbf{y}$ if and only if $|\langle \mathbf{x}, \mathbf{y} \rangle + 1/k| < \epsilon$. Listing all good k-tuples corresponds to listing all k-cliques in this graph. In the special case $k = 3$ this corresponds to listing all triangles.

Figure 5.1 illustrates how theses k-tuples were found in [HKL18]. We start off with the initial list L. For each element $\mathbf{x}_1 \in L$ we iterate through the list and filter out all vectors $\mathbf{x}_2 \in L$ such that $|\langle \mathbf{x}_1, \mathbf{x}_2 \rangle + 1/k| < \epsilon$ to form $L(\mathbf{x}_1)$. Next we iterate over all $\mathbf{x}_2 \in L(\mathbf{x}_1)$ and filter out all $\mathbf{x}_3 \in L(\mathbf{x}_1)$ such that $|\langle \mathbf{x}_2, \mathbf{x}_3 \rangle + 1/k| < \epsilon$ to form $L(\mathbf{x}_1, \mathbf{x}_2)$. This then continues until we find good k-tuples at the lowest layer. Let us denote $L(\mathbf{x}_1, \ldots, \mathbf{x}_j) = L^{(j)}$. It is shown in [HK17, Equation 16] that the intermediate list sizes are

$$|L^{(j)}| = \tilde{\mathcal{O}}\left(\left(k^{\frac{1}{k-1}} \cdot \frac{k-j}{k-j+1}\right)^{d/2}\right). \tag{5.12}$$

The cost of creating layer i here is $|L| \cdot |L^{(i-1)}| \cdot \prod_{j=1}^{i-1} |L^{(j)}|$. Thus the time complexity for all steps is

$$|L| \cdot \max_{1 \leq i \leq k-1} |L^{(i-1)}| \cdot \prod_{j=1}^{i-1} |L^{(j)}|. \tag{5.13}$$

In [HKL18] the idea of using slightly different $\text{Filter}_{i,j}$ for different pairs (i, j) in Figure 5.1 was introduced. Using the same number $1/k$ for all pairs is called balanced configuration, while deviations from it are called unbalanced configurations. Unbalanced configurations allow us to slightly improve the time complexity at the cost of a slightly increased space complexity.

Just like for the normal sieving setting, for k-sieving it is also possible to improve the time complexity by using NNS techniques like LSF. Unlike the basic $k = 2$ setting we cannot achieve optimal time without increasing the memory.

5.3.1 Quantum Improvements

Also k-sieving can be improved using a quantum computer. This was studied in detail in Paper 3 [KMPM19a]. The obvious speed-up is to use Grover's algorithm to speed-up the LSF part of k-sieving, as described in Section 5.2.1. More interestingly, it is possible to speed-up other aspects of k-sieving in a less obvious way.

5.3.2 The $k = 3$ Setting

Let us first describe the approach for the $k = 3$ case. Here we use the geometric view of the problem and view all vectors in the list as nodes and say that there is an edge between nodes $\mathbf{x}$ and $\mathbf{y}$ if and only if $|\langle \mathbf{x}, \mathbf{y} \rangle + 1/k| < \epsilon$. The description comes from Section 5 and Appendix C of [KMPM19a]. Then the following algorithm can be used to find one triangle within this graph.

1. Use Grover's algorithm to find any edge $(\mathbf{x}_1, \mathbf{x}_2) \in E$ among all potential $\mathcal{O}(n^2)$ edges.
2. Given an edge $(\mathbf{x}_1, \mathbf{x}_2)$ from Step 1, use Grover's algorithm to find a vertex $\mathbf{x}_3 \in V$, such that $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ is a triangle.
3. Apply amplitude amplification on Steps 1–2.

The graph has $n = |L|$ nodes and $m = |L||L(x_1)|$ edges, where the list sizes are from (5.10) and (5.12) respectively. Step 1 takes $\sqrt{n^2/m} = \sqrt{|L|/|L(\mathbf{x}_1)|}$ and step 2 takes $\sqrt{n} = \sqrt{|L|}$, of which step 2 is dominant. The probability that the random edge we pick belongs to a triangle is $|L|/(|L||L(\mathbf{x}_1)|) = 1/|L(\mathbf{x}_1)|$. By applying amplitude amplification on steps 1-2 the total cost of finding a triangle is $\sqrt{|L(\mathbf{x}_1)|} \cdot \sqrt{|L|}$.

By the coupon collectors problem, we need to repeat the algorithm $|L| \cdot \log(|L|)$ times to find all $|L|$ triangles [Wik20b]. Thus the total cost of this approach to 3-sieving is $|L| \log(|L|) \sqrt{|L|} \sqrt{|L(\mathbf{x}_1)|} = 2^{0.3349d+o(d)}$ using $|L| = 2^{0.1887d+o(d)}$ memory.

5.3.3 General k Setting

The approach can be generalized to listing all k-cliques in the following way.

1. Use Grover's algorithm to find an edge $(\mathbf{x}_1, \mathbf{x}_2) \in E$ among all $\mathcal{O}(|L|^2)$ pairs of nodes.

$\vdots$

i. Given an i-clique $(\mathbf{x}_1, \ldots, \mathbf{x}_i)$ from step $i - 1$, use Grover's algorithm to find a vertex $\mathbf{x}_{i+1} \in V$, such that $(\mathbf{x}_1, \ldots, \mathbf{x}_{i+1})$ is an $(i + 1)$-clique.

$\vdots$

k. Apply amplitude amplification on Steps 1–$(k-1)$.

By combining the approach above with unbalanced configurations we were able to show that quantum k-sieving for large k leads to an algorithm using $2^{0.2989d+o(d)}$ time and $2^{0.1395d+o(d)}$ space. Compared to the quantum version of sieving with LSF using space and time $2^{0.2653+o(d)}$ this corresponds to almost halving the exponent for space at a relatively small increase in time.

Chapter 6

The BKW Algorithm

The topic of this chapter is the BKW algorithm, one of the main algorithms for solving the LWE problem. This algorithm is the main topic of the whole thesis.

Many of the methods used for solving LWE using BKW were originally developed for solving LPN. We will focus on the papers applying the techniques to LWE, but also mention the original methods applied to LPN. The BKW algorithm was the developed as the first subexponential algorithm for solving LPN [BKW00]. The first paper describing in detail how to tweak the original BKW algorithm for solving LWE is [ACF+15].

The BKW algorithm consists of two parts, reduction and hypothesis testing. In this chapter we will introduce the basic versions of these two parts. Improvements will be discussed in Chapters 7 to 9.

6.1 Reduction

Consider two samples $([\pm\mathbf{a}_0, \mathbf{a}_1], b_1)$ and $([\pm\mathbf{a}_0, \mathbf{a}_2], b_2)$, where $\pm\mathbf{a}_0$ are the first b positions in the $\mathbf{a}$ vectors of the two samples. By adding/subtracting the $\mathbf{a}$ vectors and the corresponding b_i values we get a new sample with the $\mathbf{a}$ vector

$$\mathbf{a_{1,2}} = [\underbrace{0 \quad 0 \quad \cdots \quad 0}_{b \text{ symbols}} \quad * \quad * \quad \cdots \quad *],$$

and the corresponding b_i value $b_{1,2} = b_1 \pm b_2$. The corresponding error term is $e_1 \pm e_2$. This new sample has dimensionality reduced by b at the cost of increasing the standard deviation of the noise by a factor $\sqrt{2}$. The first step of the reduction part of the BKW algorithm consists of first mapping all the samples into buckets[1] based on the b first positions of the $\mathbf{a}$ vectors. Since $\pm\mathbf{a}$ are in the same bucket, while $\mathbf{0}$ is its own bucket, the total number of buckets is $(q^b + 1)/2$. Next go through each bucket and form new samples by adding/subtracting suitable pairs of samples. This reduction process is iterated a total of t times, reducing the first $t \cdot b$ positions in the $\mathbf{a}$ vectors to 0.

[1] We sometimes also use the word category which in the context of BKW reduction steps refers to the same thing as bucket.

6.1.1 LF1 vs. LF2

Which pairs to pick within a bucket was studied in detail for LPN in [LF06]. The methods transfer to LWE directly. In Levieil-Fouque 1 (LF1), we pick a representative sample from each bucket. Then we form new samples by adding/subtracting each of the other samples to/from the representative. This guarantees that all the samples are independent after the reduction phase. However, this also means that the number of samples decrease by $(q^b + 1)/2$ each step, resulting in having only $m - t(q^b + 1)/2$ samples left for the hypothesis testing.

In Levieil-Fouque 2 (LF2), we allow forming new samples by combining any pair of samples in a bucket. In the extreme case we form every single possible pair within each bucket. This means that we only need $3 \cdot (q^b + 1)/2$ samples to keep the sample size constant between the steps[2].

The disadvantage of LF2 is that it the reduced samples are no longer independent, leading to an increased noise level. For LPN experiments show that this effect is small in practice [BTV16]. In Paper 1 we studied the sample complexity of LF2 when applying the BKW algorithm to LWE and also in this setting the sample dependency problem is small [GMS20].

In Chapter 7 and 9 we will discuss improvements of these basic reduction steps.

6.2 Hypothesis Testing

After t steps of reduction all positions of the $\mathbf{a}$ vectors are canceled except the last $k = n - bt$. The samples are on the form

$$b = \sum_{i=1}^{k} a_i \cdot s_i + e \Leftrightarrow b - \sum_{i=1}^{k} a_i \cdot s_i = e. \tag{6.1}$$

Here, the problem is to decide the k different s_i values of the secret. By guessing these values the corresponding error terms in (6.1) can be calculated. If we make the wrong guess the corresponding error terms should look uniformly random, but if we make the correct guess, the values should follow another distribution.

As is shown in [BJV04] an optimal distinguisher picks the hypothesis that results in an error distribution that has a minimal relative entropy to the Discrete/Rounded Gaussian distribution with mean 0 and standard deviation $\sigma \cdot 2^{t/2}$. In other wordes, we calculate the guess $\hat{\mathbf{s}}$ that maximizes

$$\sum_{e=0}^{q-1} N(e) \log \frac{\Pr_{\bar{\Psi}_{\sigma_f,q}}(e)}{\Pr_{\mathcal{U}(0,q-1)}(e)}, \tag{6.2}$$

where $N(e)$ denotes the number of times the error term e occurs for the guess $\hat{\mathbf{s}}$ and $\Pr_D(e)$ denotes the probability of drawing the value e from the distribution D. Here we assume that the original noise was rounded Gaussian with a noise level of σ and therefore the final noise level was $\sigma_f = \sigma \cdot 2^{t/2}$. The time complexity of this approach is

[2] We can get away with slightly fewer than $3 \cdot (q^b+1)/2$ samples due to the spread of created samples not being perfectly even.

$$\mathcal{O}(mq^k). \tag{6.3}$$

After performing the secret-noise transformation of Section 6.4 we can make sure that the magnitude of the values in the secret $\mathbf{s}$ is small. Limiting ourselves to a magnitude of d we can decrease the complexity to

$$\mathcal{O}(m(2d+1)^k). \tag{6.4}$$

In Chapter 8 we will discuss how to improve this approach.

6.3 Sample Amplification

In some versions of LWE, the number of samples m we have access to is limited. Consider k different samples $(\mathbf{a}_i, b_i)$. We can now form a new sample in 2^k different ways by forming

$$\left(\sum_{i=1}^{k} \pm\mathbf{a}_i, \sum_{i=1}^{k} \pm b_i\right). \tag{6.5}$$

Doing this for all k-tuples allows us to form up to $2^k \cdot \binom{m}{k}$ samples. This comes at a cost of increasing the noise level by a factor $\sqrt{k}$ and increasing the sample dependency.

Counterintuitively, the increased noise from sample amplification does not punish BKW that much asymptotically compared to lattice-based approaches, a fact which was shown in [HKM18] and further analyzed in Paper 5 and 6 [GJMS19b, Må19].

The effect of sample dependency was studied in Paper 1 [GMS20]. In summary, the sample dependency problem from sample amplification seems to be negligible or at least not a major problem.

6.4 Secret-Noise Transformation

There is a transformation that makes the distribution of the secret follow the secret of the noise [ACPS09, Kir11]. If for example the secret follows a uniform distribution this transform could be applied to achieve multiple advantages. Given an LWE instance on the form (1.1). Arrange the columns such that the first n are linearly independent and form the matrix $\mathbf{A}_0$. Denote $\mathbf{D} = \mathbf{A}_0^{-1}$. Now write $\hat{\mathbf{s}} = \mathbf{s}\mathbf{A_0} - [b_1 \cdots b_n] = -[e_1 \cdots e_n]$. Also introduce $\hat{\mathbf{A}} = \mathbf{DA} = \begin{bmatrix}\mathbf{I}\, \hat{\mathbf{A}}_1\end{bmatrix}$ and $\hat{\mathbf{b}} = \mathbf{b} - [b_1 \cdots b_n]\hat{\mathbf{A}} = \begin{bmatrix}\mathbf{0}\, \hat{b}_{n+1} \cdots \hat{b}_m\end{bmatrix}$. Now we get the modified LWE problem

$$\hat{\mathbf{b}} = \hat{\mathbf{s}}\hat{\mathbf{A}} + \mathbf{e}, \tag{6.6}$$

with a secret $\hat{\mathbf{s}}$ with small entries. After solving (6.6) for $\hat{\mathbf{s}}$, we find $\mathbf{s}$ via trivial linear algebra.

The transformation comes at a cost of losing the first n samples. However, there are at least two advantages to the secret-noise transformation, in case the original secret has larger variance than the noise terms.

One advantage of performing this transformation is that the smaller noise in the secret vector means that we can reduce the magnitude of the $\mathbf{a}$ vectors less

and still be able to distinguish the correct secret. We discuss these improved reduction methods in Chapter 7.

Another advantage is that it lowers the amount of probable hypotheses for the hypothesis testing stage. We explain in Section 8.1 how to take advantage of this fact by using a pruned FFT approach.

Chapter 7

Improvements of the BKW Reduction Steps

This chapter introduces improvements over the basic reduction steps of Section 6.1. The chapter covers asymptotic and concrete complexity improvements, and covers some implementation aspects too.

In [AFFP14] the first improvement of the reduction part of the BKW algorithm on LWE was introduced. Instead of mapping samples with the exact same values on the $\mathbf{a}$ vectors to the same buckets, they allowed for samples that were almost, but not quite, equal on these positions to be mapped to the same buckets.

More precisely, pick a positive integer parameter p, $1 \leq p < q$. Transform each relevant position value a_i to $a'_i = a_i \backslash p$, where $\backslash$ denotes integer division. Next we map into buckets based on the transformed values instead of the original ones. The plain BKW algorithm from Chapter 6 corresponds to letting $p = 1$. This technique is called Lazy Modulus Switching (LMS).

To make a programming analogy, LMS is like mapping numbers viewed as floats, while the calculations on the numbers are done using double precision, leading to small errors, but allowing for longer steps.

One problem with the reduction steps of [AFFP14] is that the final distribution of the $\mathbf{a}$ vectors is uneven. Every new reduction step increases the magnitude of the previously reduced positions by a factor $\sqrt{2}$. See column 2 in Figure 7.1 for an illustration of the behavior.

The other idea introduced in the paper was unnatural selection. When choosing what pairs to combine within a bucket, they picked the pairs resulting in the smallest magnitudes on the previously reduced positions. In Section 7.2 we will discuss how this basic idea can be improved using techniques from lattice sieving.

7.1 Coded-BKW and LMS

The problem of the uneven distribution of the values in the $\mathbf{a}$ vectors was solved in [KF15, GJS15]. Here both the degree of reduction and the step length vary over the steps. The first step is only slightly longer than a plain BKW step and reduces the positions almost to 0. Then gradually the steps become longer

and longer and the reduction becomes less and less strict. The result is an evenly distributed $\mathbf{a}$ vector. See column 3 of Figure 7.1 for an illustration of the behavior.

A new type of reduction step was also introduced in [GJS15]. Let us explain the first reduction step, the next steps follow in an obvious way. Given an $[n_1, k]$ q-ary linear code $\mathcal{C}$. In the first step of coded-BKW samples $([\mathbf{a}_0, \mathbf{a}_1], b)$ are mapped to buckets based on the closest codeword $\mathbf{c} \in \mathcal{C}$ to $\mathbf{a}_0$. Thus, LMS corresponds to a concatenation of arbitrarily many trivial codes for each position and is thus a special case of coded-BKW.

7.2 Coded-BKW with Sieving

A new way of solving the problem of the increasing magnitudes of the previously reduced positions was introduced in [GJMS17a]. Let B denote the final magnitude needed to correctly distinguish the secret after the reduction. Map samples to buckets the same way as in [GJS15, KF15] to reduce the magnitude of the current n_i positions from q to B. Within each bucket, we pick the pairs that make sure to keep the magnitude of the previously reduced N_{i-1} positions reduced to B. For each bucket, this corresponds to doing a sieving iteration of Definition 5.1.1 from Section 5.1[1]. We can therefore apply LSF here to speed-up this process.

7.2.1 Quantum Improvement

Just like the LSF algorithm of Section 5.2 can be improved quantumly by applying Grover's algorithm, the same trick can of course be used for coded-BKW with sieving within each bucket to improve the time complexity.

7.3 Pre-processing

In most cases it is beneficial to start the reduction process with plain BKW steps. The positions in the $\mathbf{a}$ vectors that are reduced to 0 during the pre-processing phase do not grow in size in the following non-plain reduction steps. While the pre-processing does increase the noise level, the reduced dimensionality generally makes the problem easier to solve.

7.4 An Illustration of the Different BKW Steps

Figure 7.1 shows a comparison of the behavior of the different BKW versions. For each column the width corresponds to the number of positions and the height corresponds to the magnitude of the position of the $\mathbf{a}$ vector. For plain BKW we reduce a constant number of positions to 0 in each step. Thus the previously reduced positions do not increase in magnitude, but we are also limited in step size.

For a naive LMS implementation we reduce a fixed number of positions in each step, but not completely to 0. This allows us to take longer steps.

[1] You could argue that the algorithm should be called coded-BKW with NNS search. Seeing all reduced steps as a single unit you could argue that the word sieving still makes sense.

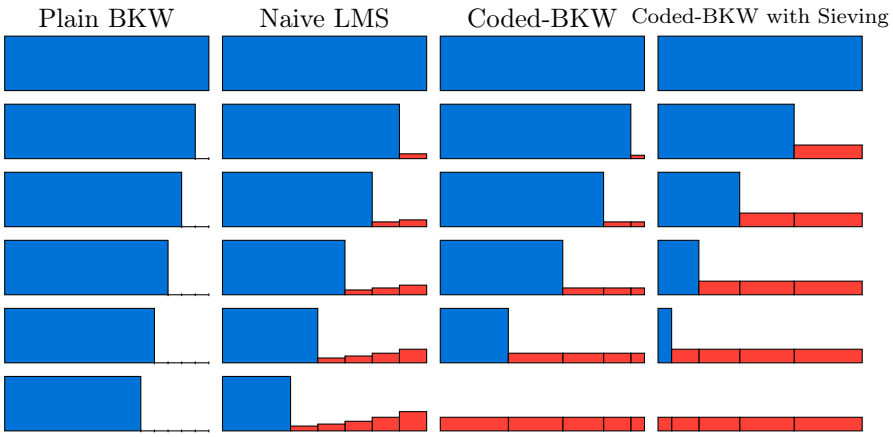


Figure 7.1: An illustration of how the magnitudes of the **a** vectors change over the reduction steps for different versions of the BKW algorithm. Slightly modified version of Figure 2 of Paper 5 [GJMS19b].

However, the magnitude of the previously reduced positions grow by a factor $\sqrt{2}$ in each step, resulting in an uneven distribution of the **a** vectors, where the first positions dominate.

The reduction strategy of [KF15, GJS15] corresponds to column 3. Here the step sizes increase and the degree of reduction decreases gradually. The final resulting distribution of the **a** vectors is even. Compared to a naive LMS approach this allows us to reduce more positions for the same amount of effort.

Column 4 corresponds to the coded-BKW with sieving strategy from [GJMS17a]. Here we begin by taking long steps since we do not have to reduce the positions that strictly. However, gradually we need to decrease the step size since the cost of keeping the magnitude of the previously reduced positions down gradually increases.

7.4.1 Optimizing the Reduction Factor

In Paper 5 [GJMS19b] the idea of [GJMS17a] was generalized by looking at different reduction factors γ for the sieving iteration part of the reduction. Instead of keeping the magnitude of the reduced positions constant we let it change from B_{i-1} to $B_i = \gamma B_{i-1}$ in step i, such that the final magnitude $B_t = B$. A value γ corresponds to setting $c = 1 - \gamma^2/2$ in Definition 5.1.1. From this perspective coded-BKW corresponds to the special case of letting $\gamma = \sqrt{2}$ and the basic coded-BKW with sieving algorithm corresponds to letting $\gamma = 1$. The optimal value γ depends on q and σ. For almost all settings this approach resulted in a strict improvement over both coded-BKW and coded-BKW with sieving.

7.4.2 Using Varying Reduction Factors

In Paper 6 [Må19], the idea of [GJMS17a] was further generalized. Here, the idea was to use different γ_i factors in different steps of the reduction. Linearly

increasing γ_i values seem to be close to optimal for this type of approach, leading to a small but noticeable improvement over Paper 5 [GJMS19b].

7.5 k-BKW

In [EHK+18] k-BKW was introduced. The main focus was on the LPN problem, but they also showed how to apply their techniques to LWE. The basic idea of the paper is to combine k samples in the reduction part of BKW instead of the standard 2.

Naively this can be done by summing all possible k-tuples

$$(\sum_{i=1}^{k} \mathbf{a}_i, \sum_{i=1}^{k} b_i), \tag{7.1}$$

where the samples $(\mathbf{a}_i, b_i)$ are any of the m available samples[2] and picking the ones that reduce the b positions to 0. We can do slightly better by instead summing all possible $(k-1)$-tuples

$$(\sum_{i=1}^{k-1} \mathbf{a}_i, \sum_{i=1}^{k-1} b_i), \tag{7.2}$$

and colliding them against the list of samples. The larger value of k we use the less samples and memory we need, but the more time we need. An advantage of k-BKW, for $k > 2$, is that we need to introduce less extra noise due to sample amplification, compared to the other reduction methods. We will discuss briefly how to improve k-BKW in Section 9.4.

7.6 Implementation Aspects

In Paper 2 we implemented the BKW algorithm [BGJ+20]. The most important contributions in the paper, other than the implementation itself, are the smooth-LMS reduction steps, our file-based reduction method and our binary distinguishing approach. We will discuss the first two contributions in this section and the binary distinguishing approach in Section 8.2.

7.6.1 Smooth-LMS

When solving concrete instances of LWE we must reduce an integer number of positions. It is oftentimes the case that reducing b positions is easy while reducing $b+1$ positions is computationally infeasible. We solved this problem by introducing so called smooth-LMS steps, allowing us to, effectively, use non-integer step sizes. While this method does not perform better than LMS/coded-BKW asymptotically, it leads to a significant improvement in concrete complexity and implementation.

[2] We can of course form any of the 2^k different combinations of samples on the form $\left(\sum_{i=1}^{k} \pm\mathbf{a}_i, \sum_{i=1}^{k} \pm b_i\right)$. While it does not change the asymptotic analysis, forming every possible combination of samples is still better for concrete complexity and when implementing the algorithm in practice.

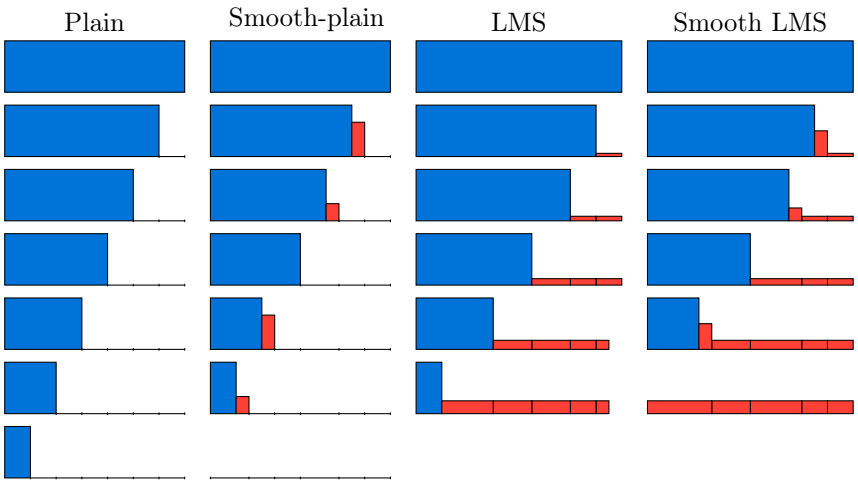


Figure 7.2: An illustration of how the magnitudes of the **a** vectors change over the reduction steps for different versions of the BKW algorithm. The heights correspond to magnitudes and the width corresponds to the number of positions. Slightly modified version of Figure 2 of Paper 5 [GJMS19b].

In the first step we reduce n_1 positions to the desired level and position n_1+1 partially. In the second step we reduce positions n_1+1 to n_1+n_2 to the desired level and position n_1+n_2+1 partially, and so on. In the final step number t we reduce positions $N_{t-1}+1$ to N_t to the desired level. The difference between LMS and smooth-LMS steps is illustrated on the right half of Figure 7.2. In this illustration we are able to reduce 2 more positions in total, compared to the standard LMS approach, by partially reducing an extra position in each step.

Smooth Plain BKW

A special case of the smooth-LMS steps is smooth plain BKW steps. Here we reduce b positions completely and an extra position partially in each step, allowing us to pre-process with an arbitrary step size.

On the left part of Figure 7.2 a comparison between smooth plain BKW and plain BKW is illustrated. Using plain steps we reduce 2 positions at a time, since reducing 3 positions is considered infeasible in this setting. However, by using smooth-LMS steps we manage to reduce 2 and 1/3 positions at a time, allowing us to reduce 2 more positions in total after 6 steps. Notice that, unlike for all the other columns, the scale for the smooth-plain reduction is logarithmic[3].

7.6.2 File-based Reduction Steps

When implementing BKW the amount of Random Access Memory (RAM) is oftentimes a limiting factor. We introduce a file-based way of doing BKW reduction steps. The problem with file-based solutions is that the file reading/writing

[3]Reducing half of a position in terms of work corresponds to reducing it from q to $q^{1/2}$.

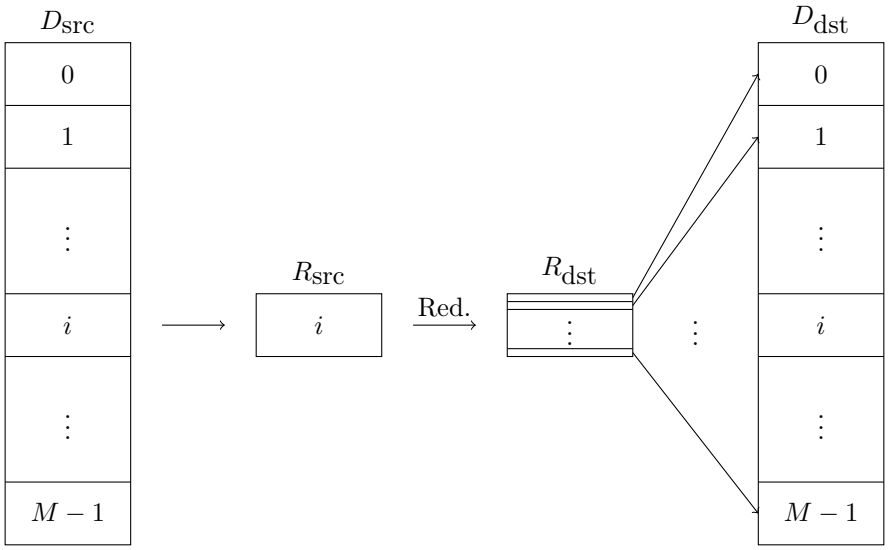


Figure 7.3: An illustration of the handling of file-based BKW reduction steps.

can lead to a large overhead cost. Here we describe how to minimize this cost when implementing file-based reduction steps[4].

Figure 7.3 illustrates our solution. In each reduction step the (source) samples are initially stored on disk in what we call D_{src}. In each reduction step the goal is to create a certain number of reduced samples and store these (destination) samples on disk in what we call D_{dst}. These destination samples are then the source samples of the next reduction step and so on. We can let the source samples and the destination samples take up roughly half of the available disk space respectively if we want to push the algorithm maximally.

The source samples are divided up into M different so called meta-categories, each containing the samples corresponding to a large amount of categories. The reduction step first reads the meta-categories one by one, each time stored in RAM in R_{src}. Then a precise mapping into categories is made, as explained in previous sections on reduction, using any type of reduction step. When new samples are formed by combining samples within each category we store the result according to the meta-category of the next reduction step in R_{dst}[5]. When R_{dst} is full we iterate through the meta-categories of R_{dst} and for each meta-category we append the meta-categories on disk D_{dst} with the samples stored on the meta-category in R_{dst}. To push the algorithm maximally we let R_{src} and R_{dst} take up roughly half of the available RAM each.

This approach means that we use M file reads and M^2 file writes for each reduction step. To push the available hardware we let $M \approx M_{\text{disk}}/M_{\text{RAM}}$, where M_{disk} is the size of the available disk space and M_{RAM} is the size of the available RAM.

[4]This description of the file-based approach is slightly different from and slightly better than the one used in Paper 2.

[5]In a purely RAM based solution we would store the samples according to their exact category.

Chapter 8

Improvements of the Guessing Procedure

In this chapter we will cover improvements over the basic guessing procedure as explained in Section 6.2. While not improving the asymptotic complexity of the algorithm, these improvements are still important for the concrete complexity of it and when implementing it in practice.

8.1 FFT and Pruned FFT

Let us revisit the situation from Section 6.2 and introduce the FFT distinguisher from [DTV15]. This approach in turn is a generalization of the corresponding distinguisher for LPN which uses the FWHT [LF06].

We have m equations of the form (6.1). Consider the function

$$f(\mathbf{x}) = \sum_{j=1}^{m} \mathbb{1}_{\mathbf{a}_j=\mathbf{x}} \theta_q^{b_j}, \tag{8.1}$$

where $\mathbf{x} \in \mathbb{Z}_q^k$, $\mathbb{1}_{\mathbf{a}_j=\mathbf{x}}$ is equal to 1 if and only if $\mathbf{x} = \mathbf{a}_j$ and 0 otherwise, and θ_q denotes the q-th root of unity. Calculating the FFT of f gives us

$$\hat{f}(\mathbf{y}) = \sum_{\mathbf{x}\in\mathbb{Z}_q^k} f(\mathbf{x}) \theta_q^{-\langle \mathbf{x},\mathbf{y}\rangle} = \sum_{j=1}^{m} \theta_q^{-(<\mathbf{a}_j,\mathbf{y}>-b_j)}, \tag{8.2}$$

for all $\mathbf{y} \in \mathbb{Z}_q^k$. For the secret $\mathbf{y} = \mathbf{s}$, the terms in (8.2) simplify to $\theta_q^{e_j}$. Since the error terms are slightly biased around 0, this turns (8.2) into a random walk with steps along the unit circle, biased towards the direction 1. For the wrong guess $\mathbf{y} \neq \mathbf{s}$ the exponents are uniformly random, turning (8.2) into a uniform random walk with directions along the unit circle. Thus, assuming that we have enough samples m for the noise level σ_f, the correct guess is the one that maximizes the real part of (8.2).

The time complexity of this approach is $\mathcal{O}(m+k\cdot q^k \cdot \log(q))$, which is a large improvement of the optimal distinguisher from Section 6.2, unless the original noise level σ is very small.

In [DTV15] the following upper limit formula for the number of samples needed to guess the secret is presented.

$$8 \cdot \ln\left(\frac{q^k}{\epsilon}\right)\left(\frac{q}{\pi}\sin\left(\frac{\pi}{q}\right)e^{-2\pi^2\sigma^2/q^2}\right)^{-2^{t+1}}. \tag{8.3}$$

Here ϵ refers to the probability of incorrectly guessing the secret. In Paper 1 we investigate the performance of this distinguisher and the optimal distinguisher [GMS20]. In summary we found the following.

1. FFT requires the same number of samples as the optimal distinguisher
2. FFT is much better than theory, the constant 8 in (8.3) is roughly an order of magnitude too large.
3. The sample dependency due to LF2 and sample amplification both seem to be relatively small.

We also suggested a slightly improved version of the FFT distinguisher using a pruned FFT. After having done the secret-noise transform introduced in Section 6.4, we know that the magnitude of the values in the secret vector is small. Assuming that the values of $\mathbf{s}$ are at most d in magnitude, we only need to calculate the FFT for the input values with magnitude less than or equal to d. Firstly, efficiently calculating the pruned FFT [SB93], this reduces the cost of calculation to

$$\mathcal{O}(m + k \cdot q^k \cdot \log(2d+1)). \tag{8.4}$$

Secondly, and more importantly, this reduces the number of samples needed to distinguish the secret. Redoing the proofs from [DTV15], but limiting the number of hypotheses to $(2d+1)^k$, we get the following upper limit formula for the number of samples needed for distinguishing.

$$8 \cdot \ln\left(\frac{(2d+1)^k}{\epsilon}\right)\left(\frac{q}{\pi}\sin\left(\frac{\pi}{q}\right)e^{-2\pi^2\sigma^2/q^2}\right)^{-2^{t+1}}. \tag{8.5}$$

Notice that also the expression (8.5) is roughly an order of magnitude larger than the simulated values in Paper 1 [GMS20]. A small bonus of the FFT approach is that it does require us to know the exact noise level of the reduced samples[1].

8.2 Binary Guessing

In Paper 2 we introduced a new approach to the guessing part of BKW [BGJ+20]. Instead of guessing the last couple of position values in $\mathbf{s}$, we translated the problem into a binary one and guessed the Least Significant Bit (LSB)s of a large number of the position values.

[1] Calculating the final noise level can be a bit complicated when taking non-plain reduction steps.

8.2.1 Retrieving the Least Significant Bits of s

Let us start with the LWE problem written on matrix form as[2]

$$\mathbf{b} = \mathbf{sA} + \mathbf{e}. \tag{8.6}$$

Next, multiply (8.6) by 2 to form

$$\mathbf{b}' = \mathbf{sA}' + 2\mathbf{e}. \tag{8.7}$$

Now perform t reduction steps of any kind that does not reduce the position values in the $\mathbf{a}$ vectors completely to 0, to form

$$\mathbf{b}'' = \mathbf{sA}'' + 2\mathbf{E}. \tag{8.8}$$

Here $\mathbf{A}'' = [\mathbf{a}_1^T \cdots \mathbf{a}_m^T]$ consists of vectors with small norm, $\mathbf{E} = [E_1 \cdots E_m]$ and $E_i = \sum_{j=1}^{2^t} e_{ij}$. Now, reduce (8.8) modulo 2 to form

$$\mathbf{b_0''} = \mathbf{s_0}\mathbf{A_0''} + \mathbf{e} \mod 2, \tag{8.9}$$

where $\mathbf{b_0''}$ is a vector with the LSBs of $\mathbf{b}''$, $\mathbf{s_0}$ is a vector with the LSBs of $\mathbf{s}$, $\mathbf{A_0''} = \mathbf{A}'' \mod 2$ and $\mathbf{e}$ denotes the binary error[3]. Now (8.8) consists of equations that can be written on integer form as

$$b_j = \sum_i s_i a_{ij} + 2E_j + k_j \cdot q. \tag{8.10}$$

When reducing modulo 2 the binary error of (8.9) is equal to $e_j = k_j \mod 2$. The higher value of k_j the lower the probability is of getting that value. Thus there is a bias towards even k_j values and thus a bias towards the binary error being 0.

Let us interpret each binary column of the matrix $\mathbf{A_0''}$ on the form $\mathbf{k} = (k_0, k_1, \ldots, k_{n-1})$ as its corresponding integer $k = \sum_{j=0}^{n-1} k_j 2^j$. Let $N = 2^n$. Let J_k denote the set of columns of the $\mathbf{A_0''}$ matrix that are equal to k. Now define the function

$$X_k = \sum_{j \in J_k} (-1)^{b_j''}. \tag{8.11}$$

Now let us calculate the WHT of X_k as

$$\hat{X}_{\boldsymbol{\omega}} = \sum_{k=0}^{N-1} X_k (-1)^{\boldsymbol{\omega}\cdot\mathbf{k}} = \sum_{k=0}^{N-1} \sum_{j \in J_k} (-1)^{b_j''} (-1)^{\boldsymbol{\omega}\cdot\mathbf{k}} = \sum_{j=0}^{m-1} (-1)^{b_j'' + \mathbf{a}_j \cdot \boldsymbol{\omega}}. \tag{8.12}$$

Since the values $\mathbf{e}$ are biased towards 0, the value 1 is more common than the value -1 in the sum (8.12), for the correct guess $\boldsymbol{\omega} = \mathbf{s_0}$. The magnitude of $\hat{X}_{\boldsymbol{\omega}}$ for this guess will be large, while for the wrong guesses the sum while be closer to 0. Thus, if the number of samples are enough compared to the noise level, then $\mathbf{s} = \bar{\boldsymbol{\omega}}$, where $|\hat{X}_{\bar{\boldsymbol{\omega}}}| = \max_{\boldsymbol{\omega}} |\hat{X}_{\boldsymbol{\omega}}|$. This random walk is precisely the binary equivalent of the corresponding random walk for the FFT distinguisher in (8.2).

[2]Notice that there is an implied reduction modulo q here, as usual for the LWE problem.
[3]Due to the reduction modulo q it is not the case that $2\mathbf{E} \mod 2 = \mathbf{0}$.

8.2.2 Retrieving the Whole s Vector

Once we have found the least significant bits of $\mathbf{s}$ we can write $\mathbf{s} = 2\mathbf{s}' + \mathbf{s_0}$, $\hat{\mathbf{A}} = 2\mathbf{A}$ and $\hat{\mathbf{b}} = \mathbf{b} - \mathbf{s_0}\mathbf{A}$. Now we get

$$\hat{\mathbf{b}} = \mathbf{s}'\hat{\mathbf{A}} + \mathbf{e}. \tag{8.13}$$

We can in turn solve (8.13) for the LSBs of $\mathbf{s}'$ using the same procedure as in Section 8.2.1. Repeating this process we can solve for the whole secret $\mathbf{s}$. Every time we have solved for new bits of $\mathbf{s}$, the remaining secret halves in magnitude, which makes the problem significantly easier to solve. Therefore, computationally speaking, we can view the LWE problem as solved once we have found the LSBs of $\mathbf{s}$.

Unlike the FFT based approach, the binary approach requires that we reduce all positions in the $\mathbf{a}$ vectors. On the other hand, the approach allows us to correctly distinguish the secret with a larger magnitude on the $\mathbf{a}$ vectors. In Paper 2 we show that the binary approach is superior in some settings [BGJ+20].

Chapter 9

Some Concluding Remarks

> I've always pursued my interests without much regard for financial value or value to the world. I've spent lots of time on totally useless things.
>
> — *Claude Shannon*

This chapter concludes the research and points towards potential future research directions. It also covers some ideas that were tried, but that did not seem to work.

9.1 A General BKW Algorithm Framework

All suggested versions of the BKW algorithm are following the following framework.

1. Given m LWE samples.
2. Apply sample amplification if more samples are needed.
3. Apply secret-noise transform if the secret has larger standard deviation than the noise.
4. Apply pre-processing with plain BKW steps.
5. Apply other reduction steps. If the binary distinguisher is used, then multiply all samples by 2 at some point during this stage.
6. Apply distinguisher to guess the last positions or, if the binary distinguisher is used, guess the LSBs of all positions being reduced after the multiplication by 2.
7. Backtrack to whatever previous step is appropriate.

Paper 1 studies the sample complexity of Point 6, both with unlimited number of samples and limited number of samples requiring the sample amplification of Point 2.

Paper 2 implements all these points. It introduces the idea of multiplying with 2 at Point 5 in order to use the binary distinguisher at Point 6. The

smooth-LMS and file-based storage solution are two ways in which it improves the reduction of Point 4 and 5.

The quantum k-sieving of Paper 3 can be applied to the reduction of Point 5 to improve the k-BKW algorithm discussed in Section 7.5. See Section 9.4 for a bit of discussion on this.

Paper 5 and 6 both improve the reduction of Point 5 by using lattice sieving techniques. They also study the performance of this algorithm when the number of given samples of Point 1 is limited, requiring the sample amplification of Point 2.

9.2 A Generic BKW Reduction Step Framework

All the different reduction steps of the BKW algorithm can be viewed from a shared framework[1]. We describe the idea asymptotically, but it can of course also be analyzed for a concrete parameter setting.

Assume that we have taken $i-1$ reduction steps and reduced the first N_{i-1} positions to a magnitude B_{i-1}. Reduction step number i combines pairs[2] of samples to produce new samples with the first N_i positions reduced to the magnitude B_i, where

- A plain BKW step [ACF+15], corresponds to letting $B_i = B_{i-1} = 0$.
- A coded-BKW/LMS step [KF15, GJS15], corresponds to letting $B_i = \sqrt{2}B_{i-1}$.
- The basic coded-BKW with sieving algorithm [GJMS17a], corresponds to letting $B_i = B_{i-1} \neq 0$.
- The improvement in Paper 5 [GJMS19b], corresponds to letting $B_i = \gamma B_{i-1}$ for a constant γ. The other suggested versions in Paper 5 only use steps from the previous bullet points and are thus also covered.
- The slight improvement of Paper 6 [Må19] corresponds to letting $B_i = \gamma_i B_{i-1}$ for γ_i values that are different in different steps.
- The k-BKW algorithm [EHK+18] and the suggested improvements of it in this thesis, correspond to the previous bullet points, but combining k samples at the time instead of 2.

The plain reduction steps are a bit of a special case, where each step corresponds quite clearly to a generalized step of Gaussian elimination. These steps should, usually, be used at the beginning to decrease the dimension at a cost of increased noise. The large advantage of these plain steps is that positions reduced to 0 stayed reduced to 0, unlike partially reduced positions.

For all the other steps, it is more helpful to view them as modified version of a sieving iteration. The first discovered approaches of coded-BKW/LMS just correspond to letting $\gamma = \sqrt{2}$ and the basic coded-BKW with sieving corresponds to $\gamma = 1$. The connection between lattice sieving and BKW is thus

[1] The author would like to thank Elena Kirshanova and Gottfried Herold for pointing out that a coded-BKW with sieving step can be viewed from this perspective. This section generalizes and studies that realization.

[2] or k-tuples more generally.

no coincidence. Even though the methods are actually quite similar, there are also some differences. Compared to sieving BKW has both advantages and disadvantages.

The two advantages for BKW are

- We do not need to reduce all positions in every step. This is of course due to the fact that positions not being reduced yet do not increase in magnitude, due to the reduction modulo q.
- We are working in a smaller dimension compared to lattice-based approaches.

There are however also some disadvantages.

- We are more limited in the number of iterations we can make. For typical parameters we only get $\Theta(\log(n))$ iterations.
- We also depend heavily on the number of samples provided. While the performance of BKW is great asymptotically even with just $\Theta(n \log(n))$ samples, for concrete instances BKW suffers from having only a limited number of available samples.

Which of the algorithms is superior depends on the parameters being used, the number of available samples and whether the comparison is made asymptotically, concretely or in implementation. See Section 9.5 for a discussion.

9.3 Potential for Improvement of the Reduction Steps

There are essentially two different possible ways of improving BKW reduction steps. Either we pick the different steps to take in a smarter way or we improve the individual step as described in Section 9.2. Let us first consider the $k = 2$ setting.

Since the position values in the previous N_{i-1} positions and the current n_i positions are independent of each other, the cost for coded-BKW with sieving-style algorithms is mulitplicative in the cost of keeping the previous N_{i-1} positions small and the cost of reducing the current n_i positions. These two costs can therefore be analyzed independently when trying to find an improvement of the algorithm.

9.3.1 Trying to Improve What Steps to Take

Let us first look into the possibility of picking the steps in a smarter way. If plain BKW is considered one extreme, then the opposite extreme reduction step is to apply a sieving iteration on all n positions in each reduction step. Since we only have a logarithmic number of steps (assuming that $q = \text{poly}(n)$) we need to reduce the positions by a pretty large factor in each step. It is possible to tweak the LSF technique to achieve this. The complexity of doing BKW this way is actually quite easy to work out, but leads to an algorithm that is just slightly better than plain BKW. The LSF part can of course be sped-up using

Grover's algorithm, but the resulting algorithm is still not better than the other BKW approaches.

In Paper 5 [GJMS19b], other than varying the reduction factor, we also tried doing coded-BKW steps followed by coded-BKW with sieving steps and vice versa. This corresponds to using $\gamma = \sqrt{2}$ followed by $\gamma = 1$ and vice versa. Doing coded-BKW with sieving steps followed by coded-BKW steps turned out to be slightly better than just doing coded-BKW with sieving and optimized γ steps, for parameter settings where the optimal γ value is close to 1. In other words the settings where optimizing for γ leads to very small improvements.

The algorithm of Paper 6 beats all the algorithms of Paper 5 for all parameter settings tested in the papers. This is not so strange. Instead of abruptly switching from doing steps with $\gamma = 1$ to steps with $\gamma = \sqrt{2}$, the algorithm of Paper 6 starts and ends at optimally picked values γ_1 and γ_t, and makes a smooth transition between them.

9.3.2 Trying to Improve the Individual Step

A possible idea for improving the individual steps is to merge the coded part of two steps of length n_1 and n_2 into reducing $n_1 + n_2$ partially two times. This can of course be generalized to merging at least a constant number of steps r. This might be interesting concretely, but at least asymptotically it is possible to show that this does not lead to an improvement.

Another possible idea is to reduce the current n_i positions by a super-constant magnitude using sieving techniques. This is unlikely to be an improvement though. It is quite easy to show that the coded approach uses a minimum amount of memory. Since the approach uses the same amount of time as the amount of memory, its time complexity cannot be improved.

9.4 Improvements of k-BKW Reduction Steps

The possible time/memory tradeoffs from k-BKW steps are not a particularly explored area of research. The techniques from coded-BKW and coded-BKW with sieving can be combined with the k-BKW idea to achieve a good improvement of the complexities for solving LWE reported in [EHK$^+$18].

The complexity of a coded version of k-BKW is pretty straightforward to derive. However, deriving the complexity when adding classical k-sieving techniques of [HK17, HKL18] or quantum k-sieving techniques of Paper 3 [KMPM19a], is more challenging. Especially if we want to add these techniques in a non-blackbox manner.

Possible further improvements should be possible to achieve by applying the techniques introduced in Papers 5 and 6 to the k-BKW setting. This is likely to lead to very messy derivations of complexity though.

9.5 BKW vs. Lattice-based Approaches

In this section we will compare the performance of BKW and lattice-based approaches.

9.5.1 Asymptotic Comparison

For comparisons between the asymptotic complexity of BKW and lattice-based approaches when solving LWE, see Figure 5 and 8 of Paper 5 [GJMS19b]. If we would use the method from Paper 6, that algorithm would cover the full area where BKW algorithms are the fastest and a small adjacent area where lattice-based algorithms are the fastest in Paper 5. In Figure 8 the number of samples is unlimited, while in Figure 5 it is $\Theta(n \log(n))$.

In general, we notice that BKW algorithms are fastest for small values of q, while lattice-based methods are best for large values of q. Counterintuitively, when having access to only a limited number of samples, BKW algorithms outperform lattice-based methods asymptotically in a larger part of the parameter space.

9.5.2 Concrete Complexity and Implementation

So far, even though BKW algorithms are asymptotically faster for some settings, they are slower in practice than lattice-based algorithms, even when using an unlimited number of samples.

Estimating the more precise bit complexity of the BKW algorithm is tricky. With recent developments, the complexity depends inherently on the complexity of the lattice sieving iterations of Definition 5.1.1, a complexity which in turn is tricky to estimate precisely. Calculating the complexity of the BKW algorithm when combining all the tricks used in Papers 2, 5 and 6 remains to be done. As discussed in Paper 2 [BGJ+20], when using an unlimited number of samples, there are settings where BKW-type algorithms beat lattice-based approaches, in terms of bit complexity.

Recently, the concrete complexity of lattice sieving using quantumer computing was studied in [AGPS19]. A possible future research idea is to do a similar analysis for the BKW-type algorithms of Papers 5 or 6. More generally, quantum improvements of the BKW algorithm is a possible area to study.

Implementation-wise, BKW algorithms are slower than lattice-based methods. Partly the reason is that much less effort has been spent on making efficient implementations of BKW and running large-scale simulations. Partly, the large memory requirement is another problem.

From the publication of the NV sieve in 2008 [NV08], it took roughly 10 years before sieving beat enumeration, even though it was asymptotically faster. It is possible that something similar will happen with BKW for the settings where it is asymptotically faster, at least assuming an unlimited number of samples.

Number of Available Samples

The main practical limitation of BKW is the number of available samples. Many practical schemes based on LWE only provide the attacker with n or $2n$ samples, which clearly is very restricting for the current approaches of BKW. There are however some possible scenarios and/or techniques that can improve this. The performance of all of these remain to be investigated.

Many constructions use versions of the LWE problem with more structure than the one from Definition 1.4.1, the main one being ring-LWE. In [Sta19] Stange shows how the BKW algorithm can take advantage of this structure,

including a rotation technique that allows the attacker to increase the initial number of samples without increasing the noise level. Similar approaches might be possible for other types of structured LWE problems.

Some constructions provide the attacker with a bit more samples. A recent example is [KKPP20]. They construct a post-quantum version of an mKEM (multiple Key Encapsulation Mechanism), which can be based on the LWE problem. Here the attacker is provided with $\Theta(m \cdot n)$ samples, where m is the number of users, a number which can be up to 50000.

Another possible improvement when having a limited number of initial samples is keeping track of the error terms each sample has and prioritize combining samples that cancel error terms [Ngu20]. Yet another possibility to increase the number of samples is to use plain k-BKW steps to increase the number of samples. Similarly we can reduce fewer than the maximal number of positions using plain BKW steps, thereby gradually increasing the sample count for each iteration.

As a final note, if BKW can become faster than lattice-based methods only with a very large number of samples, that would still be an interesting result, leading to some natural follow-up questions. What number of samples does this happen at? Is it possible to improve upon both BKW and lattice sieving by some hybrid approach?

References

[AAB+19] F. Arute, K. Arya, R. Babbush, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505—-510, October 2019.

[Aar18] Scott Aaronson. Introduction to quantum information science lecture notes. `https://www.scottaaronson.com/qclec.pdf`, 2018. Accessed: 2020-11-09.

[ACF+15] Martin R. Albrecht, Carlos Cid, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. On the complexity of the BKW algorithm on LWE. *Designs, Codes and Cryptography*, 74(2):325–354, 2015.

[ACFP14] Martin R. Albrecht, Carlos Cid, Jean-Charles Faugère, and Ludovic Perret. Algebraic algorithms for LWE. Cryptology ePrint Archive, Report 2014/1018, 2014. `http://eprint.iacr.org/2014/1018`.

[ACKS20] Divesh Aggarwal, Yanlin Chen, Rajendra Kumar, and Yixin Shen. Improved (provable) algorithms for the shortest vector problem via bounded distance decoding. arXiv, Report 2002.07955, 2020. `https://arxiv.org/abs/2002.07955`.

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 595–618, Santa Barbara, CA, USA, August 16–20, 2009. Springer, Heidelberg, Germany.

[ADH+19] Martin R. Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn W. Postlethwaite, and Marc Stevens. The general sieve kernel and new records in lattice reduction. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 717–746, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany.

[ADRS15] Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. Solving the shortest vector problem in 2^n time using discrete Gaussian sampling: Extended abstract. In

Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 733–742, Portland, OR, USA, June 14–17, 2015. ACM Press.

[AFFP14] Martin R. Albrecht, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. Lazy modulus switching for the BKW algorithm on LWE. In Hugo Krawczyk, editor, *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pages 429–445, Buenos Aires, Argentina, March 26–28, 2014. Springer, Heidelberg, Germany.

[AFG14] Martin R. Albrecht, Robert Fitzpatrick, and Florian Göpfert. On the efficacy of solving LWE by reduction to unique-SVP. In Hyang-Sook Lee and Dong-Guk Han, editors, *ICISC 13: 16th International Conference on Information Security and Cryptology*, volume 8565 of *Lecture Notes in Computer Science*, pages 293–310, Seoul, Korea, November 27–29, 2014. Springer, Heidelberg, Germany.

[AG11] Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In Luca Aceto, Monika Henzinger, and Jiri Sgall, editors, *ICALP 2011: 38th International Colloquium on Automata, Languages and Programming, Part I*, volume 6755 of *Lecture Notes in Computer Science*, pages 403–415, Zurich, Switzerland, July 4–8, 2011. Springer, Heidelberg, Germany.

[Age] National Security Agency. NSA Statement on the threat from quantum computers. `https://apps.nsa.gov/iaarchive/programs/iad-initiatives/cnsa-suite.cfm`. Accessed: 2020-10-24.

[AGPS19] Martin R. Albrecht, Vlad Gheorghiu, Eamonn W. Postlethwaite, and John M. Schanck. Estimating quantum speedups for lattice sieves. Cryptology ePrint Archive, Report 2019/1161, 2019. `https://eprint.iacr.org/2019/1161`.

[AKS01] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *33rd Annual ACM Symposium on Theory of Computing*, pages 601–610, Crete, Greece, July 6–8, 2001. ACM Press.

[Alb17] Martin R. Albrecht. On dual lattice attacks against small-secret LWE and parameter choices in HElib and SEAL. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 103–129, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany.

[AOAGC18] Eric R. Anschuetz, Jonathan P. Olson, Alán Aspuru-Guzik, and Yudong Cao. Variational quantum factoring. arXiv, Report 1808.08927, 2018. `https://arxiv.org/abs/1808.08927`.

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On The Concrete Hardness Of Learning With Errors. *J. Mathematical Cryptology*, 9(3):169–203, 2015.

[ASK19] Mirko Amico, Zain H. Saleem, and Muir Kumph. Experimental study of Shor's factoring algorithm using the IBM Q Experience. *Physical Review A*, 100(1), July 2019.

[BBBV97] Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM Journal on Computing*, 26(5):1510–1523, Oct 1997.

[BBC08] Marco Baldi, Marco Bodrato, and Franco Chiaraluce. A new analysis of the McEliece cryptosystem based on QC-LDPC codes. In Rafail Ostrovsky, Roberto De Prisco, and Ivan Visconti, editors, *SCN 08: 6th International Conference on Security in Communication Networks*, volume 5229 of *Lecture Notes in Computer Science*, pages 246–262, Amalfi, Italy, September 10–12, 2008. Springer, Heidelberg, Germany.

[BBD08] Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen. *Post-Quantum Cryptography*. Springer Publishing Company, Incorporated, 1st edition, 2008.

[BBHT98] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(4-5):493–505, Jun 1998.

[BC07] M. Baldi and F. Chiaraluce. Cryptanalysis of a new instance of McEliece cryptosystem based on QC-LDPC Codes. In *2007 IEEE International Symposium on Information Theory*, pages 2591–2595, 2007.

[BDGL16] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In Robert Krauthgamer, editor, *27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 10–24, Arlington, VA, USA, January 10–12, 2016. ACM-SIAM.

[Ber10] Daniel J. Bernstein. Grover vs. McEliece. In Nicolas Sendrier, editor, *The Third International Workshop on Post-Quantum Cryptography, PQCRYPTO 2010*, pages 73–80, Darmstadt, Germany, May 25–28 2010. Springer, Heidelberg, Germany.

[BGJ15] Anja Becker, Nicolas Gama, and Antoine Joux. Speeding-up lattice sieving without increasing the memory, using sub-quadratic nearest neighbor search. Cryptology ePrint Archive, Report 2015/522, 2015. `http://eprint.iacr.org/2015/522`.

[BGJ+20] Alessandro Budroni, Qian Guo, Thomas Johansson, Erik Mårtensson, and Paul Stankovski Wagner. Making the BKW algorithm practical for LWE. In Karthikeyan Bhargavan, Elisabeth Oswald,

and Manoj Prabhakaran, editors, *Progress in Cryptology – INDOCRYPT 2020*, pages 417–439, Cham, 2020. Springer International Publishing.

[BHMT02] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. In *Quantum Computation and Information, Contemporary Mathematics Series. AMS*, 2002.

[BHT97] Gilles Brassard, Peter Hoyer, and Alain Tapp. Quantum algorithm for the collision problem. *ACM SIGACT News*, 28:14 – 19, 1997.

[BJMM12] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in $2^{n/20}$: How 1 + 1 = 0 improves information set decoding. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 520–536, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

[BJV04] Thomas Baignères, Pascal Junod, and Serge Vaudenay. How far can we go beyond linear cryptanalysis? In Pil Joong Lee, editor, *Advances in Cryptology – ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 432–450, Jeju Island, Korea, December 5–9, 2004. Springer, Heidelberg, Germany.

[BKR11] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full AES. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 344–371, Seoul, South Korea, December 4–8, 2011. Springer, Heidelberg, Germany.

[BKW00] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. In *32nd Annual ACM Symposium on Theory of Computing*, pages 435–440, Portland, OR, USA, May 21–23, 2000. ACM Press.

[BL17] D.J. Bernstein and T. Lange. Post-quantum cryptography. *Nature*, 549(7671):188–194, September 2017.

[BLP93] J. P. Buhler, H. W. Lenstra, and Carl Pomerance. Factoring integers with the number field sieve. In Arjen K. Lenstra and Hendrik W. Lenstra, editors, *The development of the number field sieve*, pages 50–94, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[BLP11] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Smaller decoding exponents: Ball-collision decoding. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 743–760, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Heidelberg, Germany.

[BLS16] Shi Bai, Thijs Laarhoven, and Damien Stehlé. Tuple lattice sieving. *LMS Journal of Computation and Mathematics*, 19(A):146–162, January 2016.

[BM17] Leif Both and A. May. Optimizing BJMM with nearest neighbors : Full decoding in $2^{2n/21}$ and mceliece security. In *WCC workshop on coding and cryptography*, 2017.

[BMPC16] Marco Baldi, Nicola Maturo, Enrico Paolini, and Franco Chiaraluce. On the use of ordered statistics decoders for low-density parity-check codes in space telecommand links. *EURASIP Journal on Wireless Communications and Networking*, 2016, 12 2016.

[BMv78] E. Berlekamp, R. McEliece, and H. van Tilborg. On the inherent intractability of certain coding problems (corresp.). *IEEE Transactions on Information Theory*, 24(3):384–386, 1978.

[Bos99] Martin Bossert. *Channel Coding for Telecommunications*. John Wiley & Sons, Inc., USA, 1st edition, 1999.

[BSC16] M. Baldi, P. Santini, and F. Chiaraluce. Soft McEliece: MDPC code-based McEliece cryptosystems with very compact keys through real-valued intentional errors. In *2016 IEEE International Symposium on Information Theory (ISIT)*, pages 795–799, 2016.

[BTV16] Sonia Bogos, Florian Tramèr, and Serge Vaudenay. On solving LPN using BKW and variants - implementation and analysis. *Cryptography and Communications*, 8(3):331–369, 2016.

[CC98] Anne Canteaut and Florent Chabaud. A new algorithm for finding minimum-weight words in a linear code: Application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511. *Information Theory, IEEE Transactions on*, 44:367 – 378, 02 1998.

[Che14] Yuanmi Chen. *Lattice Reduction and Concrete Security of Fully Homomorphic Encryption*. PhD thesis, Paris Diderot University, France, 2014.

[Chu01] Robert Churhhouse. *Codes and Ciphers: Julius Caesar, the Enigma, and the Internet*. Cambridge University Press, 2001.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, page 151–158, New York, NY, USA, 1971. Association for Computing Machinery.

[CT65] J. Cooley and John W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19:297–301, 1965.

[Dar] TU Darmstadt Learning with Errors Challenge. `https://www.latticechallenge.org/lwe_challenge/challenge.php`. Accessed: 2020-10-16.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DSvW20] L. Ducas, M. Stevens, and W.P.J. van Woerden. New practical advances in lattice sieving, using GPU tensor cores, 2020. Under submission.

[DTV15] Alexandre Duc, Florian Tramèr, and Serge Vaudenay. Better algorithms for LWE and LWR. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 173–202, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

[Dum91] I. Dumer. The use of information sets in decoding of linear codes. *In Proceedings of 5th Joint Soviet-Swedish International Workshop on Information Theory*, pages 50 – 52, 1991.

[dW19] Ronald de Wolf. Quantum computing: Lecture notes. `https://homepages.cwi.nl/~rdewolf/qcnotes.pdf`, 2019. Accessed: 2020-11-09.

[EHK+18] Andre Esser, Felix Heuer, Robert Kübler, Alexander May, and Christian Sohler. Dissection-BKW. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 638–666, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

[FHS+17] Tomás Fabsic, Viliam Hromada, Paul Stankovski, Pavol Zajac, Qian Guo, and Thomas Johansson. A reaction attack on the QC-LDPC McEliece cryptosystem. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017*, pages 51–68, Utrecht, The Netherlands, June 26–28 2017. Springer, Heidelberg, Germany.

[Fri20] Lex Fridman. Scott Aaronson: Quantum computing | episode #72. In *Lex Fridman Podcast*, February 2020. `https://www.youtube.com/watch?v=uX5t8EivCaM`.

[FS95] M. P. C. Fossorier and Shu Lin. Soft-decision decoding of linear block codes based on ordered statistics. *IEEE Transactions on Information Theory*, 41(5):1379–1396, 1995.

[FS03] Niels Ferguson and Bruce Schneier. *Practical Cryptography*. John Wiley & Sons, Inc., USA, 1st edition, 2003.

[FS09] Matthieu Finiasz and Nicolas Sendrier. Security bounds for the design of code-based cryptosystems. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 88–105, Tokyo, Japan, December 6–10, 2009. Springer, Heidelberg, Germany.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press.

[GJMS17a] Qian Guo, Thomas Johansson, Erik Mårtensson, and Paul Stankovski. Coded-BKW with sieving. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 323–346, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany.

[GJMS17b] Qian Guo, Thomas Johansson, Erik Mårtensson, and Paul Stankovski. Information set decoding with soft information and some cryptographic applications. In *2017 IEEE International Symposium on Information Theory (ISIT)*. IEEE, jun 2017.

[GJMS19a] Qian Guo, Thomas Johansson, Erik Mårtensson, and Paul Stankovski. Some cryptanalytic and coding-theoretic applications of a soft stern algorithm. *Advances in Mathematics of Communications*, 2 2019.

[GJMS19b] Qian Guo, Thomas Johansson, Erik Mårtensson, and Paul Stankovski Wagner. On the asymptotics of solving the LWE problem using coded-bkw with sieving. *IEEE Trans. Information Theory*, 65(8):5243–5259, 2019.

[GJS15] Qian Guo, Thomas Johansson, and Paul Stankovski. Coded-BKW: Solving LWE using lattice codes. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 23–42, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[GJS16] Qian Guo, Thomas Johansson, and Paul Stankovski. A key recovery attack on MDPC with CCA security using decoding errors. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 789–815, Hanoi, Vietnam, December 4–8, 2016. Springer, Heidelberg, Germany.

[GMS20] Qian Guo, Erik Mårtensson, and Paul Stankovski Wagner. On the sample complexity of solving LWE using BKW-style algorithms. Under submission, 2020.

[GNR10] Nicolas Gama, Phong Q. Nguyen, and Oded Regev. Lattice enumeration using extreme pruning. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 257–278, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany.

[Gop70] V. D. Goppa. A new class of linear correcting codes. *Probl. Peredachi Inf.*, 6:24–30, 1970.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *28th Annual ACM Symposium on Theory of Computing*, pages 212–219, Philadephia, PA, USA, May 22–24, 1996. ACM Press.

[HK17] Gottfried Herold and Elena Kirshanova. Improved algorithms for the approximate k-list problem in euclidean norm. In Serge Fehr, editor, *PKC 2017: 20th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 10174 of *Lecture Notes in Computer Science*, pages 16–40, Amsterdam, The Netherlands, March 28–31, 2017. Springer, Heidelberg, Germany.

[HKL18] Gottfried Herold, Elena Kirshanova, and Thijs Laarhoven. Speed-ups and time-memory trade-offs for tuple lattice sieving. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018: 21st International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 10769 of *Lecture Notes in Computer Science*, pages 407–436, Rio de Janeiro, Brazil, March 25–29, 2018. Springer, Heidelberg, Germany.

[HKM18] Gottfried Herold, Elena Kirshanova, and Alexander May. On the asymptotic complexity of solving LWE. *Des. Codes Cryptogr.*, 86(1):55–83, 2018.

[HPS11] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 447–464, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Heidelberg, Germany.

[Ker83a] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX:5–83, jan 1883.

[Ker83b] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX:161–191, feb 1883.

[KF15] Paul Kirchner and Pierre-Alain Fouque. An improved BKW algorithm for LWE with applications to cryptography and lattices. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 43–62, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[Kha96] David Khan. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. Scribner, 1996.

[Kir11] Paul Kirchner. Improved generalized birthday attack. Cryptology ePrint Archive, Report 2011/377, 2011. `http://eprint.iacr.org/2011/377`.

[Kir18] Elena Kirshanova. Improved quantum information set decoding. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*, pages 507–527, Fort Lauderdale, Florida, United States, April 9–11 2018. Springer, Heidelberg, Germany.

[Kit96] A. Kitaev. Quantum measurements and the abelian stabilizer problem. *Electron. Colloquium Comput. Complex.*, 3, 1996.

[KKPP20] Shuichi Katsumata, Kris Kwiatkowski, Federico Pintore, and Thomas Prest. Scalable ciphertext compression techniques for post-quantum kems and their applications. Cryptology ePrint Archive, Report 2020/1107, 2020. `https://eprint.iacr.org/2020/1107`.

[Kle00] Philip N. Klein. Finding the closest lattice vector when it's unusually close. In David B. Shmoys, editor, *11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 937–941, San Francisco, CA, USA, January 9–11, 2000. ACM-SIAM.

[KLM17] Phillip Kaye, Raymond Laflamme, and Michele Mosca. *An Introduction to Quantum Computing*. Oxford University Press, 1st edition, 2017.

[KMPM19a] Elena Kirshanova, Erik Mårtensson, Eamonn W. Postlethwaite, and Subhayan Roy Moulik. Quantum algorithms for the approximate k-list problem and their application to lattice sieving. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 521–551, Kobe, Japan, December 8–12, 2019. Springer, Heidelberg, Germany.

[KMPM19b] Elena Kirshanova, Erik Mårtensson, Eamonn W. Postlethwaite, and Subhayan Roy Moulik. Quantum algorithms for the approximate k-list problem and their application to lattice sieving. Cryptology ePrint Archive, Report 2019/1016, 2019. `https://eprint.iacr.org/2019/1016`.

[KT17] Ghazal Kachigar and Jean-Pierre Tillich. Quantum information set decoding algorithms. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017*, pages 69–89, Utrecht, The Netherlands, June 26–28 2017. Springer, Heidelberg, Germany.

[Laa15] Thijs Laarhoven. *Search problems in cryptography*. PhD thesis, Eindhoven University of Technology, 2015.

[LB88] Pil Joong Lee and Ernest F. Brickell. An observation on the security of McEliece's public-key cryptosystem. In C. G. Günther, editor, *Advances in Cryptology – EUROCRYPT'88*, volume 330 of *Lecture Notes in Computer Science*, pages 275–280, Davos, Switzerland, May 25–27, 1988. Springer, Heidelberg, Germany.

[LC04] Shu Lin and Daniel Costello. *Error Control Coding*. Prentice Hall, 2nd edition, 01 2004.

[LdW15] Thijs Laarhoven and Benne de Weger. Faster sieving for shortest lattice vectors using spherical locality-sensitive hashing. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *Progress in Cryptology - LATINCRYPT 2015: 4th International Conference on Cryptology and Information Security in Latin America*, volume 9230 of *Lecture Notes in Computer Science*, pages 101–118, Guadalajara, Mexico, August 23–26, 2015. Springer, Heidelberg, Germany.

[LF06] Éric Levieil and Pierre-Alain Fouque. An improved LPN algorithm. In Roberto De Prisco and Moti Yung, editors, *SCN 06: 5th International Conference on Security in Communication Networks*, volume 4116 of *Lecture Notes in Computer Science*, pages 348–359, Maiori, Italy, September 6–8, 2006. Springer, Heidelberg, Germany.

[LLL82] A.K. Lenstra, H.W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515 – 534, 1982.

[LN13] Mingjie Liu and Phong Q. Nguyen. Solving BDD by enumeration: An update. In Ed Dawson, editor, *Topics in Cryptology – CT-RSA 2013*, volume 7779 of *Lecture Notes in Computer Science*, pages 293–309, San Francisco, CA, USA, February 25 – March 1, 2013. Springer, Heidelberg, Germany.

[LP92] H.W. Lenstra and Carl Pomerance. A rigorous time bound for factoring integers. *J. Amer. Math. Soc. 5*, 5, 09 1992.

[LP11] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In Aggelos Kiayias, editor, *Topics in Cryptology – CT-RSA 2011*, volume 6558 of *Lecture Notes in Computer Science*, pages 319–339, San Francisco, CA, USA, February 14–18, 2011. Springer, Heidelberg, Germany.

[Lö14] Carl Löndahl. *Some Notes on Code-Based Cryptography*. PhD thesis, Lund University, 2014.

[McE78] R. J. McEliece. A Public-Key Cryptosystem Based On Algebraic Coding Theory. *Deep Space Network Progress Report*, 44:114–116, January 1978.

[ME99] Michele Mosca and Artur Ekert. The hidden subgroup problem and eigenvalue estimation on a quantum computer. In Colin P. Williams, editor, *Quantum Computing and Quantum Communications*, pages 174–188, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[Mer78] Ralph C. Merkle. Secure communications over insecure channels. *Communications of the ACM*, 21:294–299, 1978.

[MMT11] Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $\tilde{\mathcal{O}}(2^{0.054n})$. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 107–124, Seoul, South Korea, December 4–8, 2011. Springer, Heidelberg, Germany.

[MO15] Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 203–228, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

[MR09] Daniele Micciancio and Oded Regev. Lattice-based cryptography. In *Post-Quantum Cryptography*, pages 147–191. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[MTSB13] R. Misoczki, J. Tillich, N. Sendrier, and P. S. L. M. Barreto. MDPC-McEliece: New McEliece variants from moderate density parity-check codes. In *2013 IEEE International Symposium on Information Theory*, pages 2069–2073, 2013.

[MV10] Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the shortest vector problem. In Moses Charika, editor, *21st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1468–1480, Austin, TX, USA, January 17–19, 2010. ACM-SIAM.

[MvOV01] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1st edition, 2001.

[Må16] Erik Mårtensson. Solving NTRU Challenges Using the New Progressive BKZ Library. Master's thesis, Lund University, Sweden, 2016.

[Må19] Erik Mårtensson. The asymptotic complexity of Coded-BKW with sieving using increasing reduction factors. In *IEEE International Symposium on Information Theory, ISIT 2019, Paris, France, July 7-12, 2019*, pages 2579–2583. IEEE, 2019.

[NC10] Michael Nielsen and Isaac Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2nd edition, 2010.

[Ngu20] Vu Nguyen. Personal communication, 2020.

[NISa] NIST Lightweight Cryptography Competition. `https://csrc.nist.gov/projects/lightweight-cryptography`. Accessed: 2020-10-23.

[NISb] NIST Post-quantum Cryptography Competition. `https://csrc.nist.gov/projects/post-quantum-cryptography`. Accessed: 2020-10-23.

[NISc] NIST Post-quantum Cryptography Competition Round 1. `https://csrc.nist.gov/Projects/post-quantum-cryptography/Round-1-Submissions`. Accessed: 2020-10-23.

[NISd] NIST Post-quantum Cryptography Competition Round 2. `https://csrc.nist.gov/Projects/post-quantum-cryptography/round-2-submissions`. Accessed: 2020-10-23.

[NISe] NIST Post-quantum Cryptography Competition Round 3. `https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions`. Accessed: 2020-10-23.

[NV08] Phong Nguyen and Thomas Vidick. Sieve algorithms for the shortest vector problem are practical. *Journal of Mathematical Cryptology*, 2:181–207, 07 2008.

[OS09] Raphael Overbeck and Nicolas Sendrier. Code-based cryptography. In *Post-Quantum Cryptography*, pages 95–145. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[Pat75] N. Patterson. The algebraic decoding of goppa codes. *IEEE Transactions on Information Theory*, 21(2):203–207, 1975.

[PBY17] Peter Pessl, Leon Groot Bruinderink, and Yuval Yarom. To bliss-b or not to be: Attacking strongswan's implementation of post-quantum signatures. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1843–1855, New York, NY, USA, 2017. Association for Computing Machinery.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 333–342, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press.

[PGN+19] Edwin Pednault, John A. Gunnels, Giacomo Nannicini, Lior Horesh, and Robert Wisnieff. Leveraging secondary storage to simulate deep 54-qubit sycamore circuits, 2019.

[PM16] Peter Pessl and Stefan Mangard. Enhancing side-channel analysis of binary-field multiplication with bit reliability. In Kazue Sako, editor, *Topics in Cryptology – CT-RSA 2016*, volume 9610 of *Lecture Notes in Computer Science*, pages 255–270, San Francisco, CA, USA, February 29 – March 4, 2016. Springer, Heidelberg, Germany.

[Pol78] J. M. Pollard. Monte carlo methods for index computation ($\mathrm{mod}\, p$). *Mathematics of Computation*, 32(143):918–924, 1978.

[Pra62] E. Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, 1962.

[Qua] Quantum algorithm zoo. https://quantumalgorithmzoo.org/. Accessed: 2020-10-24.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 84–93, Baltimore, MA, USA, May 22–24, 2005. ACM Press.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.

[SB93] H. V. Sorensen and C. S. Burrus. Efficient computation of the DFT with only a subset of input or output points. *IEEE Transactions on Signal Processing*, 41(3):1184–1200, 1993.

[Sch87] C.P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theor. Comput. Sci.*, 53(2):201–224, June 1987.

[SE94] C. P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical Programming*, 66:181 – 199, 1994.

[Sha49] Claude E. Shannon. Communication theory of secrecy systems. *Bell Systems Technical Journal*, 28(4):656–715, 1949.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science*, pages 124–134, Santa Fe, NM, USA, November 20–22, 1994. IEEE Computer Society Press.

[Sin99] Simon Singh. *The Code Book: The Evolution of Secrecy from Mary, Queen of Scots, to Quantum Cryptography.* Doubleday, USA, 1st edition, 1999.

[Sma16] Nigel Smart. *Cryptography Made Simple.* Springer International Publishing, 1st edition, 2016.

[SP18] Douglas Robert Stinson and Maura Paterson. *Cryptography : Theory and Practice.* Boca Raton : Chapman and Hall/CRC, 4th edition, 2018.

[Sta19] Katherine E. Stange. Algebraic aspects of solving ring-LWE, including ring-based improvements in the blum-kalai-wasserman algorithm. Cryptology ePrint Archive, Report 2019/183, 2019. https://eprint.iacr.org/2019/183.

[Ste89] Jacques Stern. A method for finding codewords of small weight. In Gérard Cohen and Jacques Wolfmann, editors, *Coding Theory and Applications*, pages 106–113, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg.

[SVP] SVP Challenge. https://www.latticechallenge.org/svp-challenge/#. Accessed: 2020-10-16.

[VF01] Antoine Valembois and Marc Fossorier. Generation of binary vectors that optimize a given weight function with application to soft-decision decoding. In *Proceedings 2001 IEEE Information Theory Workshop*, pages 138–140, 02 2001.

[VF04] A. Valembois and M. Fossorier. Box and match techniques applied to soft-decision decoding. *IEEE Transactions on Information Theory*, 50(5):796–810, 2004.

[Wik20a] Wikipedia contributors. Combination - number of combinations with repetition — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Combination#Number_of_combinations_with_repetition`, 2020. [Online; accessed 2020-09-29].

[Wik20b] Wikipedia contributors. Coupon collector's problem — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Coupon_collector%27s_problem`, 2020. [Online; accessed 2020-10-02].

[Wik20c] Wikipedia contributors. Post-quantum cryptography — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Post-quantum_cryptography`, 2020. [Online; accessed 2020-09-28].

[Wik20d] Wikipedia contributors. Quantum algorithm for linear systems of equations — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Quantum_algorithm_for_linear_systems_of_equations`, 2020. [Online; accessed 2020-11-21].

[Wik20e] Wikipedia contributors. Quantum annealing — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Quantum_annealing`, 2020. [Online; accessed 2020-11-21].

[Wik20f] Wikipedia contributors. Quantum simulator — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Quantum_simulator`, 2020. [Online; accessed 2020-11-21].

[ZWD+20] Han-Sen Zhong, Hui Wang, Yu-Hao Deng, Ming-Cheng Chen, Li-Chao Peng, Yi-Han Luo, Jian Qin, Dian Wu, Xing Ding, Yi Hu, Peng Hu, Xiao-Yan Yang, Wei-Jun Zhang, Hao Li, Yuxuan Li, Xiao Jiang, Lin Gan, Guangwen Yang, Lixing You, Zhen Wang, Li Li, Nai-Le Liu, Chao-Yang Lu, and Jian-Wei Pan. Quantum computational advantage using photons. *Science*, 2020.

Part II

Included Papers

Paper I

On the Sample Complexity of solving LWE using BKW-Style Algorithms

The Learning with Errors problem (LWE) receives more and more attention in cryptography, mainly due to its fundamental significance in post-quantum cryptography. Among the extensive studies on its solving algorithms, the Blum-Kalai-Wasserman algorithm (BKW), originally proposed for solving the Learning Parity with Noise problem (LPN), performs well, especially for certain parameter settings with cryptographic importance. The BKW algorithm generally consists of two phases, the reduction phase and the solving phase. In this work, we study the performance of distinguishers used in the solving phase. We show that the Fast Fourier Transform distinguisher (FFT) from Eurocrypt'15 has the same sample complexity as the optimal distinguisher, when making the same number of hypotheses. We also show that it performs much better than theory predicts and introduces an improvement of it called the pruned FFT distinguisher. Finally, we indicate, via extensive experiments, that the sample dependency due to both LF2 and sample amplification is limited.

Keywords: LWE, BKW, FFT distinguisher, Hypothesis testing.

Qian Guo, Erik Mårtensson and Paul Stankovski Wagner, "On the Sample Complexity of solving LWE using BKW-Style Algorithms". Under Submission, 2020.

1 Introduction

Post-quantum cryptography is a central area in recent cryptographic research, studying replacements of cryptographic primitives based on the factoring and the discrete-log problems, both of which could efficiently be solved by a quantum computer [30]. Among all of its research branches, lattice-based cryptography is the most promising one; for instance, in the NIST Post-Quantum Cryptography (PQC) Standardization project [1], twelve out of twenty-six round-2 candidates are lattice-based.

The *Learning with Errors* problem (LWE), originally proposed by Regev [29], is the major problem in lattice-based cryptography. Its average-case hardness can be based on the worst-case hardness of some standard lattice problems, which is extremely interesting in theoretical crypto. Its cryptographic applications are very versatile, and among them, the most famous is the design of Fully Homomorphic Encryption (FHE) schemes. Its binary counterpart, the *Learning Parity with Noise* problem (LPN), also plays an significant role in cryptography (see [10]), especially in light-weight cryptography for very constrained environments such as RFID tags and low-power devices.

Due to the significance of LWE, a number of works have contributed to its solving algorithms. These approaches can be generally categorized into three classes – the lattice reduction, the algebraic, and the combinatorial approaches. The last class of algorithms all inherit from the famous Blum-Kalai-Wasserman (BKW) algorithm [11, 12], and are the most relevant to our study. We refer interested readers to [6] for concrete complexity estimation for solving LWE instances, and to [22, 24] for asymptotic complexity estimations.

The BKW-type algorithms include two phases, the reduction phase and the solving phase. The prior consists of a series of operations, called BKW steps, iteratively reducing the dimension of the problem at the cost of increasing its noise level. At the end of the reduction phase, the original LWE problem is transformed to a new problem with a much smaller dimension. The new problem can be solved efficiently by a procedure called distinguishing in the solving phase.

One of the main challenges in understanding the precise performance of BKW variants on solving the LWE problem comes from the lack of extensive experimental studies, especially on the various distinguishers proposed for the solving phase. Firstly, we have borrowed many heuristics from BKW variants on the LPN problem, but only very roughly or not at all verified them for the LWE problem. Secondly, the tightness of the nice theoretical bound in [16] on the sample complexity of the FFT distinguisher also needs to be experimentally checked. Lastly, a performance comparison of the different known distinguishers is still lacking.

1.1 Related Work

The BKW algorithm proposed by Blum et al. [11,12] is the first sub-exponential algorithm for solving the LPN problem. Its initial distinguisher, an exhaustive search method in the binary field, recovers one bit of the secret by employing majority voting. Later, Levieil and Fouque [27] applied the fast Walsh-Hadamard transform (FWHT) technique to accelerate the distinguishing process and recovered a number of secret bits in one pass. They also proposed some heuristic versions and tested these assumptions by experiments. In [25] Kirchner pro-

posed a secret-noise transform technique to change the secret distribution to be sparse. This technique is an application of the transform technique proposed in [7] for solving LWE. Bernstein and Lange [9] further instantiated an attack on the Ring-LPN problem, a variant of LPN with algebraic ring structures. In [19,20], Guo, Johansson, and Löndahl proposed a new distinguishing method called subspace hypothesis testing. Though this distinguisher can handle an instance with larger dimension by using covering codes, its inherent nature is still an FWHT distinguisher. Improvements of the BKW algorithm was further studied by Zhang et al. [32] and Bogos-Vaudenay [14]. An elaborate survey with experimental results on the BKW algorithm for solving LPN can be found in [13].

We see a similar research line when applying the BKW idea for solving LWE. Albrecht et al. initiated the study in [3]. In PKC 2014 [5], a new reduction technique called lazy modulus switching was proposed. In both works, an exhaustive search approach is employed in the solving phase. In [16] Duc et al. introduced the fast Fourier transform (FFT) technique in the distinguishing process and bounded the sample complexity theoretically from the Hoeffding inequality. Note that the actual performance regarding the bound is not experimentally verified and the information loss in the FFT distinguisher is unclear. There are new reduction methods in [21, 23, 26], and in [23], the authors also proposed a new method with polynomial reconstruction in the solving phase. This method has the same sample complexity as that of the exhaustive search approach but requires $(q+1)$ FFT operations rather than only one FFT in [16].

The BKW variants with memory constraints were recently studied in [15, 17, 18].

1.2 Contributions

In the paper, we perform a thorough empirical study to compare the performance of the known distinguishers. We investigate the performance of the optimal distinguisher and the FFT distinguisher. We also test the sample dependency when using LF2 or sample amplification. The following is a list of our contributions.

1. We show that the FFT method performs equally well as the optimal distinguisher in terms of number of samples needed for correctly guessing the secret, if we make sure that the distinguishers make the same number of hypotheses. This means that, except for very sparse secrets, the FFT distinguisher is always preferable. This also makes the polynomial reconstruction method of [23] obsolete.

2. We indicate that the formula from [16] for the number of samples needed for distinguishing is off by roughly an order of magnitude.

3. We introduce a pruned FFT method. By limiting the number of hypotheses to only the reasonable ones, we show that this method can improve the performance of the FFT distinguisher from [16] without any computational overhead.

4. We indicate that the sample dependency due to using LF2 or sample amplification is limited.

1.3 Organization

The rest of the paper is organized in the following way. We begin by introducing some necessary background in Section 2. In Section 3 we cover some basics about the BKW algorithm. In Section 4 we go over different distinguishers used for hypoethesis testing when solving LWE using BKW and introduce the pruned FFT method. Next, in Section 5 we show why the FFT distinguisher and the optimal distinguisher perform identically for our setting, which is followed by simulation results in Section 6. Finally, we conclude the paper in Section 7.

2 Background

We start by introducing some notation. We use a bold small letter to denote a vector. Let $\langle \cdot, \cdot \rangle$ denote the scalar products of two vectors with the same dimension. By $|x|$ we denote the absolute value of x for a real number $x \in \mathbb{R}$. We also denote by $\Re(y)$ the real part and $\|y\|$ the absolute value of a complex number $y \in \mathbb{C}$.

2.1 LWE

Let us define the LWE problem.

Definition 2.1 (LWE)**.** *Let n be a positive integer, q an odd prime. Let $\mathbf{s}$ be a uniformly random secret vector in $\mathbb{Z}_q^n$. Assume access to m noisy scalar products between $\mathbf{s}$ and known vectors $\mathbf{a_i}$, i.e.*

$$b_i = \langle \mathbf{a_i}, \mathbf{s} \rangle + e_i, \tag{1}$$

for $i = 1, \ldots, m$. The small error terms e_i are drawn from a distribution χ. The (search) LWE problem is to find the secret vector $\mathbf{s}$.

In other words, when solving LWE you have access to a large set of pairs $(\mathbf{a_i}, b_i)$ and want to find the corresponding secret vector $\mathbf{s}$. In some versions there are restrictions on the number of samples you have access to.

If we let $\vec{b} = (b_1, b_2, \ldots, b_m)$, $\vec{e} = (e_1, e_2, \ldots, e_m)$ and $\mathbf{A} = [\vec{a}_1^T, \vec{a}_2^T \cdots \vec{a}_m^T]$ we can write the whole problem on matrix form as

$$\vec{b} = \vec{s}\mathbf{A} + \vec{e}. \tag{2}$$

2.2 Rounded Gaussian Distribution

For the error we use the rounded Gaussian distribution[1]. Let $f(x|0, \sigma^2)$ denote the PDF of the normal ditribution with mean 0 and standard deviation σ, this distribution in turn being denoted as $\mathcal{N}(0, \sigma^2)$. Depending on the two parameters q and σ the rounded Gaussian distribution samples values from $\mathcal{N}(0, \sigma^2)$, rounds to the nearest integer and wraps the value to the interval $[-(q-1)/2, (q-1)/2]$. In other words, the probability of picking a certain error e is equal to

[1] Also common is to use the Discrete Gaussian distribution, which is similar.

$$\sum_{k=-\infty}^{\infty} \int_{e-1/2+k\cdot q}^{e+1/2+k\cdot q} f(x|0,\sigma^2)dx,$$

for $e \in [-(q-1)/2, (q-1)/2]$. We denote this distribution by $\bar{\Psi}_{\sigma,q}$. We use the well-known heuristic approximation that the sum of two independent distributions X_1 and X_2, drawn from $\bar{\Psi}_{\sigma_1,q}$ and $\bar{\Psi}_{\sigma_2,q}$, is drawn from $\bar{\Psi}_{\sqrt{\sigma_1^2+\sigma_2^2},q}$. We also use the notation $\alpha = \sigma/q$.

Finally, we let $\mathcal{U}(a,b)$ denote the discrete uniform distribution taking values from a up to b.

3 BKW

The BKW algorithm was originally invented to solve LPN. It was first used for LWE in [3]. The BKW algorithm consists of two parts, reduction and hypothesis testing. Let us begin by describing the reduction part.

3.1 Reduction

We divide samples into categories based on b position values in the $\mathbf{a}$ vectors. Two samples should be in the same category if and only if the b position values get canceled when adding or subtracting the $\mathbf{a}$ vectors. Given two samples $([\pm\mathbf{a_0}, \mathbf{a_1}], b_1)$ and $([\pm\mathbf{a_0}, \mathbf{a_2}], b_2)$ within the same category. By adding/subtracting the $\mathbf{a}$ vectors we get

$$\mathbf{a_{1,2}} = [\underbrace{0 \quad 0 \quad \cdots \quad 0}_{b \text{ symbols}} \quad * \quad * \quad \cdots \quad *].$$

By also calculating the corresponding b value we get $b_{1,2} = b_1 \pm b_2$. Now we have a new sample $(\mathbf{a_{1,2}}, b_{1,2})$. The corresponding noise variable is $e_{1,2} = e_1 \pm e_2$. Thus the variance of the new noise is $2\sigma^2$, where σ^2 is the variance of the originial noise. By going through all categories and calculating a suitable number of new samples we have reduced the dimensionality of the problem by b, at the cost of increasing the noise variance to $2\sigma^2$. If we repeat the reduction process t times we end up with a dimensionality of $n - tb$, and a noise variance of $2^t \cdot \sigma^2$.

LF1 and LF2

LF1 and LF2 are two implementation tricks originally proposed for solving LPN in [27]. Both can naturally be generalized for solving LWE.

In LF1 we pick one representative per category. We form new samples by subtracting or adding the other samples in this category from the representative. This process makes sure that all samples at the hypothesis testing stage are independent of each other. However, it also means that the sample size shrinks by $(q^b - 1)/2$ samples per generation, requiring a large initial sample size.

In LF2 we allow forming samples by combining any pair of samples within a category. This allows us to form much more samples. In LF2, if we form every possible sample, it is enough with a sample size of $3(q^b - 1)/2$ to form a new generation of samples of the same size. The disadvantage of this approach is that the samples are no longer independent, leading to higher noise levels in the

hypothesis stage of BKW. It is generally assumed that this effect is quite small, and this heuristic method is well tested for solving the LPN problem [27].

Sample Amplification

In some versions of LWE the number of initial samples is limited. To get more samples we can use sample amplification. For example, by adding/subtracting triples of samples we can increase the initial sample size m up to a maximum of $4 \cdot \binom{m}{3}$. This does however increase the noise by a factor of $\sqrt{3}$. It also leads to an increased dependency between samples in the hypothesis testing phase, similar in principle to LF2.

Secret-Noise Transformation

There is a transformation of the LWE problem that makes sure that the distribution of the secret vector follows the distribution of the noise [7, 25]. Assume for simplicity that the first n columns in $\mathbf{A}$ are linearly independent and form the invertible matrix $\mathbf{A}_0$. Now replace $\mathbf{A}$ by $\hat{\mathbf{A}} = \mathbf{A}_0^{-1}\mathbf{A} = [I \vec{a}_{n+1}^T \vec{a}_{n+2}^T \cdots \vec{a}_m^T]$ and $\mathbf{b}$ by $\hat{\mathbf{b}} = \mathbf{b} - (b_1, b_2, \ldots, b_n)\hat{\mathbf{A}} = (\mathbf{0}\hat{b}_{n+1}, \hat{b}_{n+2}, \hat{b}_m)$. Now we can search for the transformed secret vector $\hat{\vec{s}} = \vec{s}\mathbf{A}_0 - (b_1, b_2, \ldots, b_n)$. Once we have found $\hat{\vec{s}}$ we can find $\vec{s}$ via a simple inverse transformation.

Improved Reduction Methods

There are improvements of the basic reduction steps of the plain BKW algorithm. In [5] lazy modulus switching (LMS) was introduced. The main idea is to map vectors that almost, but not completely, cancel each other out when added/subtracted, into the same category. This leads to an extra error term, but allows us to take longer step. By correctly managing the total noise LMS makes it possible to solve larger instances. The problem with the approach of [5] is that the noise from the previously reduced positions increases in each new reduction step, leading to an uneven noise distribution.

This idea was improved in [23, 26]. The new idea was to vary the length of the reduction steps and the strictness of the reduction, leading to an evenly spread final noise. The next improvement was in [21]. To handle the problem of the increasing noise of the previously reduced positions, only the pairs creating the smallest noise levels were used within each category. This selection was done using techniques from lattice sieving. The asymptotics of the idea of combining BKW and lattice sieving was subsequently improved in [22, 28].

For the sake of comparing distinguishers it does not matter how the reduction is done. What matters is only the size of the final noise. Thus, for the sake of simplicity we only use plain reduction steps in this paper.

3.2 Hypothesis Testing

Assume that we have reduced all positions down to 0 except for k positons, leaving k positions for the hypothesis testing phase. After the reduction phase of t steps of plain BKW we end up with samples on the form

$$b = \sum_{i=1}^{k} a_i \cdot s_i + e \Leftrightarrow b - \sum_{i=1}^{k} a_i \cdot s_i = e, \tag{3}$$

where e is (approximately) rounded Gaussian distributed with a standard deviation of $\sigma_f = 2^{t/2} \cdot \sigma$ and mean 0. Now the problem is to distinguish the correct guess $\mathbf{s} = (s_1, s_2, \ldots, s_k)$ from all the incorrect guesses, among the total q^k different hypotheses[2]. For each guess $\hat{\mathbf{s}}$ we calculate the corresponding error terms in (3). For the correct guess the observed values of e are rounded Gaussian distributed, while for the wrong guess they are uniformly random.

How the hypothesis testing is done to distinguish the right guess from all the wrong ones is explained in Section 4.

4 Distinguishers

For the hypothesis testing we look at two different distinguishers, the optimal distinguisher, which is an exhaustive search method; and a faster method based on the fast Fourier transform.

For the optimal distinguisher, for each hypothesis we go through (3) for each sample and calculate a metric for how close the e values are to a rounded Gaussian distribution.

4.1 Optimal Distinguisher

Let $D_{\hat{\mathbf{s}}}$ denote the distribution of the e values for a given guess of the secret vector $\hat{\mathbf{s}}$. As is shown in [8, Prop. 1] to optimally distinguish the hypothesis $D_{\hat{\mathbf{s}}} = \mathcal{U}(0, q-1)$ against $D_{\hat{\mathbf{s}}} = \bar{\Psi}_{\sigma_f, q}$ we calculate the log-likelihood ratio

$$\sum_{e=0}^{q-1} N(e) \log \frac{\Pr_{\bar{\Psi}_{\sigma_f,q}}(e)}{\Pr_{\mathcal{U}(0,q-1)}(e)} = \sum_{e=0}^{q-1} N(e) \log \left(q \cdot \Pr_{\bar{\Psi}_{\sigma_f,q}}(e) \right), \tag{4}$$

where $N(e)$ denotes the number of times e occurs for the guess $\hat{\mathbf{s}}$, σ_f denotes the standard deviation of the samples after the reduction phase and $\Pr_D(e)$ denotes the probability of drawing e from the distribution D. We pick the value $\hat{\mathbf{s}}$ that maximizes (4). The time complexity of the optimal distinguisher is

$$\mathcal{O}(m \cdot q^k), \tag{5}$$

if we try all possible hypotheses. After performing the secret-noise transformation of Section 3.1 we can limit ourselves to assuming that the k values in $\mathbf{s}$ have an absolute value of at most d, reducing the complexity to

$$\mathcal{O}(m \cdot (2d+1)^k). \tag{6}$$

Limiting ourselves to only testing the likely hypotheses also have another advantage. The fewer hypotheses we test the less probable it is that an incorrect one gets picked [3]. This trick of limiting the number of hypotheses can of course also be applied to the FFT method of Section 4.2, which we do in Section 4.4.

[2] After the secret-noise transforming most of these hypotheses are almost guaranteed to be incorrect, simplifying the hypothesis testing a bit.

[3] as long as the correct one is among our hypotheses.

4.2 Fast Fourier Transform Method

For LWE, the idea of using a transform to speed up the distinguishing was introduced in [16]. Consider the function

$$f(\mathbf{x}) = \sum_{j=1}^{m} \mathbb{1}_{\mathbf{a}_j=\mathbf{x}} \theta_q^{b_j}, \tag{7}$$

where $\mathbf{x} \in \mathbb{Z}_q^k$, $\mathbb{1}_{\mathbf{a}_j=\mathbf{x}}$ is equal to 1 if and only if $\mathbf{x} = \mathbf{a}_j$ and 0 otherwise, and θ_q denotes the q-th root of unity. The idea of the FFT distinguisher is to calculate the FFT of f, that is

$$\hat{f}(\boldsymbol{\alpha}) = \sum_{\mathbf{x}\in\mathbb{Z}_q^k} f(\mathbf{x})\theta_q^{-\langle \mathbf{x},\boldsymbol{\alpha}\rangle} = \sum_{j=1}^{m} \theta_q^{-(\langle \vec{a}_j,\boldsymbol{\alpha}\rangle - b_j)}. \tag{8}$$

For any vector different from the secret $\boldsymbol{\alpha} \neq \vec{s}$, the exponents in (8) are uniformly random and (8) corresponds to a random walk of m steps in the complex plane along directions on the unit circle. If $\boldsymbol{\alpha} = \vec{s}$, then the exponents are equal to e_j. These values are more centered around 0 than uniformly random values. This will bias the random walk along the positive direction of the real axis. Thus, if m is large enough compared to the noise level, the secret $\mathbf{s}$ is the value that maximizes $\Re(\hat{f}(\boldsymbol{\alpha}))$ from (8).

Time Complexity

The time complexity for building the function f from (7) is $\mathcal{O}(m)$. The time complexity of calculating the FFT of f is $\mathcal{O}(q^k \cdot \log(q^k))$ leading to a total time complexity of

$$\mathcal{O}(m + k \cdot q^k \cdot \log(q)). \tag{9}$$

In general this complexity is much lower than the one in (5). However, it does depend on the sparsity of the secret $\vec{s}$. For a binary $\vec{s}$, the exhaustive methods are better.

Sample Complexity

From [16, Thm. 16] we have the following (upper limit) formula for the number of samples needed for the FFT distinguisher

$$8 \cdot \ln\left(\frac{q^k}{\epsilon}\right)\left(\frac{q}{\pi}\sin\left(\frac{\pi}{q}\right)e^{-2\pi^2\sigma^2/q^2}\right)^{-2^{t+1}}, \tag{10}$$

where ϵ is the probability of guessing $\vec{s}$ incorrectly. Notice that the expression is slightly modified to fit our notation and that a minor error in the formula is corrected[4].

[4] Using our notation k should be within the logarithm and not as a factor in front of it like in [16].

4.3 Polynomial Reconstruction Method

In [23], a method combining exhaustive search and the fast Fourier transform was introduced. It can achieve the optimal distinguishing from the information-theoretical perspective. On the other hand, it is much more efficient compared to the optimal distinguisher. Compared to the complexity of the FFT methods (9), however, the complexity is higher by a factor of roughly q.

4.4 Pruned FFT Distinguisher

Just like for the optimal distinguisher we can limit the number of hypotheses when using the FFT distinguisher. Since we only need a small subset of the output values of the FFT distinguisher in (8), we can speed-up the calculations using a pruned FFT. In general, if we only need K out of all N output values, the time complexity for calculating the FFT improves from $\mathcal{O}(N\log(N))$ to $\mathcal{O}(N\log(K))$ [31]. Limiting the magnitude when guessing the last k positions of $\mathbf{s}$ to d, this changes the time complexity from (9) to

$$\mathcal{O}(m + k \cdot q^k \cdot \log(2d+1)). \tag{11}$$

The more important gain of this method is in the sample complexity. In the formula for sample complexity (10), the numerator q^k corresponds to the number of values of $\mathbf{s}$ can take on the last k positions. Re-doing the proofs of [16, Thm. 16], but limiting the magnitude of the guess in each position to d, we get the formula

$$8 \cdot \ln\left(\frac{(2d+1)^k}{\epsilon}\right)\left(\frac{q}{\pi}\sin\left(\frac{\pi}{q}\right)e^{-2\pi^2\sigma^2/q^2}\right)^{-2^{t+1}}. \tag{12}$$

Notice that this reduced sample complexity comes at no computational cost.

5 Equal Performance of Optimal and FFT Distinguishers

When starting to run simulations, we noticed that the FFT distinguisher and the optimal distinguisher performed identically, in terms of number of samples to correctly guess the secret. We explain this phenomenon in Appendix A[5].

There are two immediate effects of this finding.

- The polynomial reconstruction method from Section 4.3 is now obsolete.
- Unless the secret is very sparse, the FFT distinguisher is strictly better than the optimal distinguisher, since it is computationally cheaper.

Hence we limit ourselves to investigating the FFT distinguisher in Section 6.

Notice that we do not make any wider claims about the equivalance between the sample complexity of the two distinguishers outside of our context of solving LWE using BKW, when having large rounded (or Discrete) Gaussian noise [6].

[5] We do, of course, not claim that this is true in general for distinguishing distributions outside of our context of solving LWE using BKW.

[6] Although it could be interesting to investigate.

6 Simulations and Results

This section covers the simulations we ran and the results they yielded. For all the figures, each data point corresponds to running plain BKW plus distinguishing at least 30 times. For most data points we ran slightly more iterations. See Appendix B for details on the number of iterations for all the data points. We picked our parameters inspired by the so called Darmstadt LWE Challenge [2].

The Darmstadt LWE Challenge

The Darmstadt LWE Challenge is a set of (search) LWE problem instances with the goal of comparing the efficacy of different methods of solving LWE. You have access to the problem dimension n, the modulus $q \approx n^2$, the relative error size α and $m \approx n^2$ equations of the form (1). In our simulations we mostly use parameters inspired by the LWE challenges, that is, we mostly let $q = 1601$ (corresponding to $n = 40$) and vary α to get problem instances that require a suitable number of samples for simulating hypothesis testing. Currently the records for the LWE challenges are set using lattice sieving [4].

6.1 Varying Noise Level

In Figure 1a we compare the theoretical expressions for sample complexity from (10) with our implementation results. We compare it against an implementation of the FFT distinguisher of [16] and an implementation of the pruned FFT distinguisher suggested in this paper. The latter distinguisher guesses values of absolute value up to 3σ, rounded upwards. The simulated data points are the median values of our simulations and the theoretical values correspond to setting $\epsilon = 0.5$ in (10). We use $q = 1601$, $n = 28$, we take $t = 13$ steps of plain BKW, reducing 2 positions per step. Finally we guess the last 2 positions and measure the minimum number of samples to correctly guess the secret. We vary α between 0.005 and 0.006. We use LF1 to guarantee that the samples are independent.

We notice that there is a consistant gap of roughly a factor 10 between theory and simulation. More exactly, the gap is a factor [10.8277, 8.6816, 10.1037, 8.6776, 10.5218, 10.1564] for the six data points, counting in increasing order of noise level.

We also see a gap between the FFT distinguisher and the pruned FFT distinguisher. We can estimate how much better the pruned FFT should be compared to the standard version by comparing (12) and (10). Counting in increasing level of noise by theory we expect the pruned version to need [1.8056, 1.8056, 1.7895, 1.7743, 1.7598, 1.7461] times less samples for the 6 data points. The numbers from the simulation were [2.0244, 1.8610, 1.8433, 2.1905, 2.0665, 2.2060], pretty close to theory.

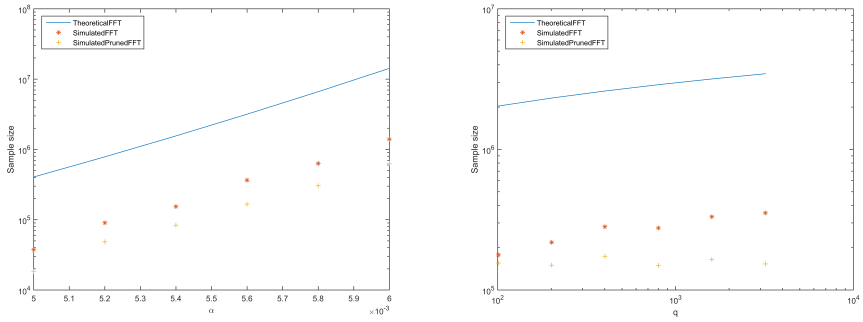
6.2 Varying q

In Figure 1b we show how the number of samples needed for distinguishing varies as we vary q. For q we use the values [101, 201, 401, 801, 1601, 3201], for α we use the values [0.0896, 0.0448, 0.0224, 0.0112, 0.0056, 0.0028] and the number of steps were [5, 7, 9, 11, 13, 15]. This way we make sure that both the

final noise level and the original $\mathbf{s}$ vectors have almost the same distribution, making the values of q the only varying factor. We use LF1 to guarantee that the samples are independent.

Notice first of all that the number of samples needed to guess the secret is roughly an order of magnitude lower than theory predicts, counting in increasing order of q, the gain is a factor [11.4537, 10.6112, 9.2315, 10.4473, 9.5561, 9.7822] for the six data points.

Also notice that the pruned version is an improvement, that increases with q. The explanation for this is that the total number of hypotheses divided by the number of hypotheses we make increases with q. By comparing (12) and (10), we expect the improvement to be a factor [1.1303, 1.2871, 1.4563, 1.6152, 1.7743, 1.9334]. This is pretty close to the factors 1.1435, 1.4551, 1.6215, 1.8507, 2.0121, 2.3045] from simulation.



(a) Varying α (b) Varying q

Figure 1: Comparing the theoretical values with our simulated values.

6.3 LF1 vs LF2

We investigate the increased number of samples needed due to dependencies, when using LF2. Here we limit ourselves to using the FFT distinguisher. For LF2, depending on the number of samples needed for guessing, we used either the minimum number of samples to produce a new generation of the same size or a sample size roughly equal to the size needed for guessing at the end. To test the limit of LF2 we made sure to produce every possible sample from each category. See Figure 2a for details and the setting is the same as in Section 6.1. We limit ourselves to using the pruned FFT distinguisher.

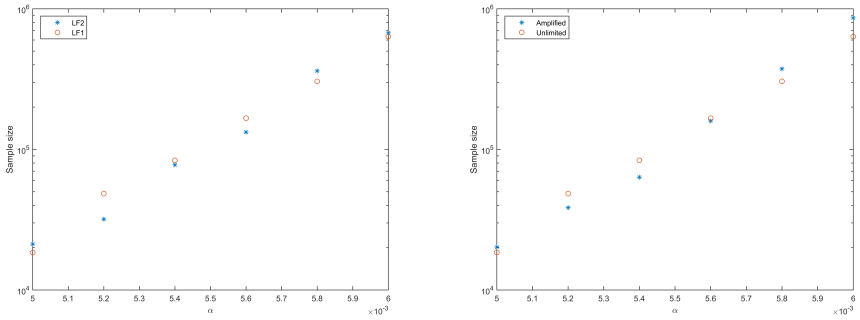
Notice that the performance of the distinguisher is almost exactly the same in both the LF1 and the LF2 cases, as is generally assumed [27].

6.4 Sample Amplification

In Figure 2b we investigate the increased number of samples needed due to dependencies, when using sample amplification. We use $q = 1601$ and start with 1600 samples. We form new samples by randomly adding/subtracting triples of samples to get a large enough set of samples. We vary the noise level between $\alpha = 0.005/\sqrt{3}$ and $\alpha = 0.006/\sqrt{3}$. We take 13 steps of plain

BKW, reducing 2 positions per step. Finally we guess the last 2 positions and measure the minimum number of samples needed to correctly guess the secret. We compare the results against starting with as many samples as we want and noise levels between $\alpha = 0.005$ and $\alpha = 0.006$ to isolate the effect of the increased dependencies due to sample amplification. We use LF1 to isolate the sample dependency due to sample amplification. We limit ourselves to using the pruned FFT distinguisher.

There does not seem to be a clear difference between the data points. This implies that the dependency caused by sample amplification is rather limited.



(a) LF1 vs. LF2 (b) Unlimited vs. sample amplification

Figure 2: Testing the increased number of samples needed due to dependencies.

6.5 Implementation

The source code for the simulation is part of an ongoing implementation project. It will be made available when it is in a more mature state.

7 Conclusions

In this paper we have shown that, in terms of sample complexity, the FFT distinguisher performs as well as the optimal distinguisher for solving LWE using the BKW algorithm. We have also showed that it performs roughly an order of magnitude better than the upper limit formula from [16, Thm. 16]. By introducing a pruned version of the FFT method we improved the sample complexity of the FFT solver, with no computational overhead.

Finally, we have also indicated that the sample dependency due to using

both LF2 and sample amplification is limited.

References

[1] NIST Post-Quantum Cryptography Standardization. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization, accessed: 2019-09-24

[2] TU Darmstadt Learning with Errors Challenge. https://www.latticechallenge.org/lwe_challenge/challenge.php, accessed: 2020-09-30

[3] Albrecht, M.R., Cid, C., Faugère, J.C., Fitzpatrick, R., Perret, L.: On the complexity of the BKW algorithm on LWE. Designs, Codes and Cryptography 74(2), 325–354 (2015)

[4] Albrecht, M.R., Ducas, L., Herold, G., Kirshanova, E., Postlethwaite, E.W., Stevens, M.: The general sieve kernel and new records in lattice reduction. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2019, Part II. Lecture Notes in Computer Science, vol. 11477, pp. 717–746. Springer, Heidelberg, Germany, Darmstadt, Germany (May 19–23, 2019)

[5] Albrecht, M.R., Faugère, J.C., Fitzpatrick, R., Perret, L.: Lazy modulus switching for the BKW algorithm on LWE. In: Krawczyk, H. (ed.) PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography. Lecture Notes in Computer Science, vol. 8383, pp. 429–445. Springer, Heidelberg, Germany, Buenos Aires, Argentina (Mar 26–28, 2014)

[6] Albrecht, M.R., Player, R., Scott, S.: On The Concrete Hardness Of Learning With Errors. J. Mathematical Cryptology 9(3), 169–203 (2015)

[7] Applebaum, B., Cash, D., Peikert, C., Sahai, A.: Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In: Halevi, S. (ed.) Advances in Cryptology – CRYPTO 2009. Lecture Notes in Computer Science, vol. 5677, pp. 595–618. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2009)

[8] Baignères, T., Junod, P., Vaudenay, S.: How far can we go beyond linear cryptanalysis? In: Lee, P.J. (ed.) Advances in Cryptology – ASIACRYPT 2004. Lecture Notes in Computer Science, vol. 3329, pp. 432–450. Springer, Heidelberg, Germany, Jeju Island, Korea (Dec 5–9, 2004)

[9] Bernstein, D.J., Lange, T.: Never trust a bunny. Cryptology ePrint Archive, Report 2012/355 (2012), http://eprint.iacr.org/2012/355

[10] Blum, A., Furst, M.L., Kearns, M.J., Lipton, R.J.: Cryptographic primitives based on hard learning problems. In: Stinson, D.R. (ed.) Advances in Cryptology – CRYPTO'93. Lecture Notes in Computer Science, vol. 773, pp. 278–291. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 22–26, 1994)

[11] Blum, A., Kalai, A., Wasserman, H.: Noise-tolerant learning, the parity problem, and the statistical query model. In: 32nd Annual ACM Symposium on Theory of Computing. pp. 435–440. ACM Press, Portland, OR, USA (May 21–23, 2000)

[12] Blum, A., Kalai, A., Wasserman, H.: Noise-tolerant learning, the parity problem, and the statistical query model. J. ACM 50(4), 506–519 (2003), https://doi.org/10.1145/792538.792543

[13] Bogos, S., Tramèr, F., Vaudenay, S.: On solving L P N using B K W and variants - implementation and analysis. Cryptography and Communications 8(3), 331–369 (2016), https://doi.org/10.1007/s12095-015-0149-2

[14] Bogos, S., Vaudenay, S.: Optimization of LPN solving algorithms. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology – ASIACRYPT 2016, Part I. Lecture Notes in Computer Science, vol. 10031, pp. 703–728. Springer, Heidelberg, Germany, Hanoi, Vietnam (Dec 4–8, 2016)

[15] Delaplace, C., Esser, A., May, A.: Improved low-memory subset sum and LPN algorithms via multiple collisions. In: Albrecht, M. (ed.) 17th IMA International Conference on Cryptography and Coding. Lecture Notes in Computer Science, vol. 11929, pp. 178–199. Springer, Heidelberg, Germany, Oxford, UK (Dec 16–18, 2019)

[16] Duc, A., Tramèr, F., Vaudenay, S.: Better algorithms for LWE and LWR. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 173–202. Springer, Heidelberg, Germany, Sofia, Bulgaria (Apr 26–30, 2015)

[17] Esser, A., Heuer, F., Kübler, R., May, A., Sohler, C.: Dissection-BKW. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018, Part II. Lecture Notes in Computer Science, vol. 10992, pp. 638–666. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2018)

[18] Esser, A., Kübler, R., May, A.: LPN decoded. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology – CRYPTO 2017, Part II. Lecture Notes in Computer Science, vol. 10402, pp. 486–514. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2017)

[19] Guo, Q., Johansson, T., Löndahl, C.: Solving LPN using covering codes. In: Sarkar, P., Iwata, T. (eds.) Advances in Cryptology – ASIACRYPT 2014,

Part I. Lecture Notes in Computer Science, vol. 8873, pp. 1–20. Springer, Heidelberg, Germany, Kaoshiung, Taiwan, R.O.C. (Dec 7–11, 2014)

[20] Guo, Q., Johansson, T., Löndahl, C.: Solving LPN using covering codes. J. Cryptology 33(1), 1–33 (2020), https://doi.org/10.1007/s00145-019-09338-8

[21] Guo, Q., Johansson, T., Mårtensson, E., Stankovski, P.: Coded-BKW with sieving. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology – ASIACRYPT 2017, Part I. Lecture Notes in Computer Science, vol. 10624, pp. 323–346. Springer, Heidelberg, Germany, Hong Kong, China (Dec 3–7, 2017)

[22] Guo, Q., Johansson, T., Mårtensson, E., Stankovski Wagner, P.: On the asymptotics of solving the LWE problem using coded-bkw with sieving. IEEE Trans. Information Theory 65(8), 5243–5259 (2019), https://doi.org/10.1109/TIT.2019.2906233

[23] Guo, Q., Johansson, T., Stankovski, P.: Coded-BKW: Solving LWE using lattice codes. In: Gennaro, R., Robshaw, M.J.B. (eds.) Advances in Cryptology – CRYPTO 2015, Part I. Lecture Notes in Computer Science, vol. 9215, pp. 23–42. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2015)

[24] Herold, G., Kirshanova, E., May, A.: On the asymptotic complexity of solving LWE. Des. Codes Cryptogr. 86(1), 55–83 (2018), https://doi.org/10.1007/s10623-016-0326-0

[25] Kirchner, P.: Improved generalized birthday attack. Cryptology ePrint Archive, Report 2011/377 (2011), http://eprint.iacr.org/2011/377

[26] Kirchner, P., Fouque, P.A.: An improved BKW algorithm for LWE with applications to cryptography and lattices. In: Gennaro, R., Robshaw, M.J.B. (eds.) Advances in Cryptology – CRYPTO 2015, Part I. Lecture Notes in Computer Science, vol. 9215, pp. 43–62. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2015)

[27] Levieil, É., Fouque, P.A.: An improved LPN algorithm. In: Prisco, R.D., Yung, M. (eds.) SCN 06: 5th International Conference on Security in Communication Networks. Lecture Notes in Computer Science, vol. 4116, pp. 348–359. Springer, Heidelberg, Germany, Maiori, Italy (Sep 6–8, 2006)

[28] Mårtensson, E.: The asymptotic complexity of coded-bkw with sieving using increasing reduction factors. In: IEEE International Symposium on Information Theory, ISIT 2019, Paris, France, July 7-12, 2019. pp. 2579–2583. IEEE (2019), https://doi.org/10.1109/ISIT.2019.8849218

[29] Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th Annual ACM Symposium on Theory of Computing. pp. 84–93. ACM Press, Baltimore, MA, USA (May 22–24, 2005)

[30] Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: 35th Annual Symposium on Foundations of Computer Science. pp. 124–134. IEEE Computer Society Press, Santa Fe, NM, USA (Nov 20–22, 1994)

[31] Sorensen, H.V., Burrus, C.S.: Efficient computation of the dft with only a subset of input or output points. IEEE Transactions on Signal Processing 41(3), 1184–1200 (1993)

[32] Zhang, B., Jiao, L., Wang, M.: Faster algorithms for solving LPN. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology – EUROCRYPT 2016, Part I. Lecture Notes in Computer Science, vol. 9665, pp. 168–195. Springer, Heidelberg, Germany, Vienna, Austria (May 8–12, 2016)

A Explaining the Optimimality of the FFT Distinguisher

Consider a sample on the form (3). By making a guess $\hat{\mathbf{s}}$ we calculate the corresponding error term $\hat{e}$. The Fourier transform of the FFT distinguisher in (8) can now be written as

$$\sum_{j=1}^{m} \theta_q^{\hat{e}_j}. \tag{13}$$

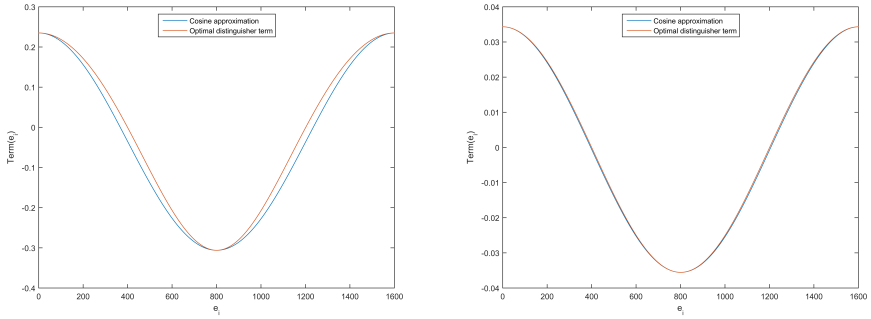
The real part (13) is equal to

$$\sum_{j=1}^{m} \cos\left(2\pi \hat{e}_j/q\right). \tag{14}$$

The FFT distinguisher picks the guess that maximizes (14). Now, let us rewrite (4) for the optimal distinguisher as

$$\sum_{j=1}^{m} \log\left(q \cdot \Pr_{\bar{\Psi}_{\sigma_f,q}}(\hat{e}_j)\right). \tag{15}$$

It turns out that with increasing noise level, the terms in (15) can be approximated as cosine functions with a period of q, as illustrated in Figure 3. The terms correspond to $q = 1601$, starting with rounded Gaussian noise with $\alpha = 0.005$, $\sigma = \alpha \cdot q = 8.005$ and taking 12 or 13 steps of plain BKW respectively. Notice that the approximation gets drastically better with increasing noise level[7]. The 13 step picture corresponds to the setting used in most of the experiments in Section 6. For a large-scale problem, the noise level would of course be much larger, resulting in an even better cosine approximation.



(a) Taking 12 plain BKW steps (b) Taking 13 plain BKW steps

Figure 3: Approximating the terms in (15) as cosine functions.

Since both distinguishers pick the $\hat{\mathbf{s}}$ that minimizes a sum of cosine functions with the same period, they will pick the same $\hat{\mathbf{s}}$, hence they will perform identically.

[7] Also notice that the approximation is not necessarily the best cosine approximation. It is simple the approximation that matches the largest and the smallest value of the curve.

B Number of Iterations in the Simulations

The following is a collection of lists of the number of iterations used for each data point to get the estimations of the median values in Figures 1b-2b. For each figure and curve we list the number iterations from left to right in, in other words in increasing level of noise level α or modulus q.

Figure 1a

Simulated FFT	31	51	52	59	50	52
Simulated Pruned FFT	33	41	56	35	30	49

Figure 1b

Simulated FFT	100	100	95	80	67	82
Simulated Pruned FFT	100	100	95	80	67	82

Figure 2a

LF1	33	41	56	35	30	49
LF2	43	46	69	37	69	50

Figure 2b

Unlimited samples	33	41	56	35	30	49
Sample Amplification	37	59	38	45	47	40

Paper II

Making the BKW Algorithm Practical for LWE

The Learning with Errors (LWE) problem is one of the main mathematical foundations of post-quantum cryptography. One of the main groups of algorithms for solving LWE is the Blum-Kalai-Wasserman (BKW) algorithm. This paper presents new improvements for BKW-style algorithms for solving LWE instances. We target minimum concrete complexity and we introduce a new reduction step where we partially reduce the last position in an iteration and finish the reduction in the next iteration, allowing non-integer step sizes. We also introduce a new procedure in the secret recovery by mapping the problem to binary problems and applying the Fast Walsh Hadamard Transform. The complexity of the resulting algorithm compares favourably to all other previous approaches, including lattice sieving. We additionally show the steps of implementing the approach for large LWE problem instances. The core idea here is to overcome RAM limitations by using large file-based memory.

Keywords: BKW, LWE, Lattice-Based Cryptography, FWHT, Post-Quantum Cryptography.

Alessandro Budroni, Qian Guo, Thomas Johansson, Erik Mårtensson and Paul Stankovski Wagner, "Making the BKW Algorithm Practical for LWE", in *21st International Conference on Cryptology in India (INDOCRYPT 2020)*, pp. 417-439, 2020, Bangalore, India.

1 Introduction

Since a large-scale quantum computer easily breaks both the problem of integer factoring and the discrete logarithm problem [34], public-key cryptography needs to be based on other underlying mathematical problems. In post-quantum cryptography - the research area studying such replacements - lattice-based problems are the most promising candidates. In the NIST post-quantum standardization competition, 5 out of 7 finalists and 2 out of 8 alternates are lattice-based [1].

The Learning with Errors problem (LWE) introduced by Regev in [33], is the main problem in lattice-based cryptography. It has a theoretically very interesting average-case to worst-case reduction to standard lattice-based problems. It has many cryptographic applications, including but not limited to, design of Fully Homomorphic Encryption Schemes (FHE). An interesting special case of LWE is the Learning Parity with Noise problem (LPN), introduced in [12], which has interesting applications in light-weight cryptography.

Considerable cryptanalytic effort has been spent on algorithms for solving LWE. These can be divided into three categories: lattice-reduction, algebraic methods and combinatorial methods. The algebraic methods were introduced by Arora and Ge in [9] and further considered in [3]. For very small noise these methods perform very well, but otherwise the approach is inefficient. The methods based on lattice-reduction are currently the most efficient ones in practise. One way of comparing the different approaches is through the Darmstadt LWE Challenges [2], where the lattice-based approach called General Sieve Kernel (G6K) is the currently most successful algorithm in breaking challenges [5]. The combinatorial algorithms are all based on the Blum-Kalai-Wasserman (BKW) algorithm and algorithms in this direction will be the focus of this paper.

For surveys on the concrete and asymptotic complexity of solving LWE, see [7] and [22, 24], respectively. In essence, BKW-style algorithms have a better asymptotic performance than lattice-based approaches for parameter choices with large noise. Unlike lattice-based approaches, BKW-style algorithms pay a penalty when the number of samples is limited (like in the Darmstadt challenges).

1.1 Related Work

The BKW algorithm was originally developed as the first subexponential algorithm for solving the LPN problem [13]. In [27] the algorithm was improved, introducing new concepts like LF2 and the use of the fast Walsh-Hadamard transform (FWHT) for the distinguishing phase. A new distinguisher using subspace hypothesis testing was introduced in [19, 20].

The BKW algorithm was first applied to the LWE problem in [4]. This idea was improved in [6], where the idea of Lazy Modulus Switching (LMS) was introduced. The idea was improved in [23, 26], where [23] introduced so called coded-BKW steps. The idea of combining coded-BKW or LMS with techniques from lattice sieving [11] lead to the next improvement [21]. This combined approach was slightly improved in [22, 30]. The distinguishing part of the BKW algorithm for solving LWE was improved by using the Fast Fourier Transform (FFT) in [16]. One drawback of BKW is its high memory-usage. To remedy this, time-memory trade-offs for the BKW algorithm were recently

studied in [15, 17, 18].

1.2 Contributions

In this paper we introduce a new BKW-style algorithm including the following.

- A generalized reduction step that we refer to as smooth-LMS, allowing us to use non-integer step sizes. These steps allow us to use the same time, space and sample complexity in each reduction step of the algorithm, which improves performance compared to previous work.
- A binary-oriented method for the guessing phase, transforming the LWE problem into an LPN problem. While the previous FFT method guesses a few positions of the secret vector and finds the correct one, this approach instead finds the least significant bits of a large amount of positions using the FWHT. This method allows us to correctly distinguish the secret with a larger noise level, generally leading to an improved performance compared to the FFT based method. In addition, the FWHT is much faster in implementation.
- Concrete complexity calculations for the proposed algorithm showing the lowest known complexity for some parameter choices selected as in the Darmstadt LWE Challenge instances, but with unrestricted number of samples.
- An implementation approach for the algorithm that allows larger instances to be solved. The implementation is file-based and stores huge tables on disk and not in RAM only. The file read/write is minimized by implementing the algorithm in a clever way. Simulation results on solving larger instances are presented and verifies the previous theoretical arguments.

1.3 Organization

We organize the rest of the paper as follows. We introduce some necessary background in Section 2. In Section 3 we cover previous work on applying the BKW algorithm to the LWE problem. Then in Section 4 we introduce our new Smooth-LMS reduction method. Next, in Section 5 we go over our new binary-oriented guessing procedure. Sections 6 and 7 cover the complexity analysis and implementation of our algorithm, respectively. Section 8 describes our experimental results using the implementation. Finally, the paper is concluded in Section 9.

2 Background

2.1 Notation

Throughout the paper we use the following notations.

- We write $\log(\cdot)$ for the base 2 logarithm.

- In the n-dimensional Euclidean space $\mathbb{R}^n$, by the norm of a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ we consider its L_2-norm, defined as

$$\|\mathbf{x}\| = \sqrt{x_1^2 + \cdots + x_n^2}.$$

  The Euclidean distance between vectors $\mathbf{x}$ and $\mathbf{y}$ in $\mathbb{R}^n$ is defined as $\|\mathbf{x} - \mathbf{y}\|$.

- Elements in $\mathbb{Z}_q$ are represented by the set of integers in $[-\frac{q-1}{2}, \frac{q-1}{2}]$.

- For an $[N, k]$ linear code, N denotes the code length and k denotes the dimension.

2.2 The LWE and LPN Problems

The LWE problem [33] is defined as follows.

Definition 2.1. *Let n be a positive integer, q a prime, and let $\mathcal{X}$ be an error distribution selected as the discrete Gaussian distribution on $\mathbb{Z}_q$ with variance σ^2. Fix $\mathbf{s}$ to be a secret vector in $\mathbb{Z}_q^n$, chosen from some distribution (usually the uniform distribution). Denote by $L_{\mathbf{s},\mathcal{X}}$ the probability distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ obtained by choosing $\mathbf{a} \in \mathbb{Z}_q^n$ uniformly at random, choosing an error $e \in \mathbb{Z}_q$ from $\mathcal{X}$ and returning*

$$(\mathbf{a}, z) = (\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e)$$

in $\mathbb{Z}_q^n \times \mathbb{Z}_q$. The (search) LWE *problem is to find the secret vector $\mathbf{s}$ given a fixed number of samples from $L_{\mathbf{s},\mathcal{X}}$.*

The definition above gives the *search* LWE problem, as the problem description asks for the recovery of the secret vector $\mathbf{s}$. Another version is the *decision* LWE problem, in which case the problem is to distinguish between samples drawn from $L_{\mathbf{s},\mathcal{X}}$ and a uniform distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$.

Let us also define the LPN problem, which is a binary special case of LWE.

Definition 2.2. *Let k be a positive integer, let $\mathbf{x}$ be a secret binary vector of length k and let $X \sim \mathsf{Ber}_\eta$ be a Bernoulli distributed error with parameter $\eta > 0$. Let $L_{\mathbf{x},X}$ denote the probability distribution on $\mathbb{F}_2^k \times \mathbb{F}_2$ obtained by choosing $\mathbf{g}$ uniformly at random, choosing $e \in \mathbf{F}_2$ from X and returning*

$$(\mathbf{g}, z) = (\mathbf{g}, \langle \mathbf{g}, \mathbf{x} \rangle + e)$$

The (search) LPN *problem is to find the secret vector $\mathbf{s}$ given a fixed number of samples from $L_{\mathbf{x},X}$.*

Just like for LWE, we can also, analogously, define *decision* LPN.

Previously, analysis of algorithms solving the LWE problem have used two different approaches. One being calculating the number of operations needed to solve a certain instance for a particular algorithm, and then comparing the different complexity results. The other being asymptotic analysis. Solvers for the LWE problem with suitable parameters are expected to have fully exponential complexity, bounded by 2^{cn} as n tends to infinity, where the value of c depends on the algorithms and the parameters of the involved distributions. In this paper, we focus on the complexity computed as the number of arithmetic operations in $\mathbb{Z}_q$, for solving particular LWE instances (and we do not consider the asymptotics).

2.3 Discrete Gaussian Distributions

We define the discrete Gaussian distribution over $\mathbb{Z}$ with mean 0 and variance σ^2, denoted $D_{\mathbb{Z},\sigma}$ as the probability distribution obtained by assigning a probability proportional to $\exp(-x^2/(2\sigma^2))$ to each $x \in \mathbb{Z}$. Then, the discrete Gaussian distribution $\mathcal{X}$ over $\mathbb{Z}_q$ with variance σ^2 (also denoted $\mathcal{X}_\sigma$) can be defined by folding $D_{\mathbb{Z},\sigma}$ and accumulating the value of the probability mass function over all integers in each residue class modulo q. It makes sense to consider the noise level as α, where $\sigma = \alpha q$. We also define the rounded Gaussian distribution on Z_q. This distribution samples values by sampling values from the continuous Gaussian distribution with mean 0 and variance σ^2, rounding to the closest integer and then folding the result to the corresponding value in $\mathbb{Z}_q$. We denote it by $\bar{\Psi}_{\sigma,q}$.

If two independent X_1 and X_2 are drawn from $\mathcal{X}_{\sigma_1}$ and $\mathcal{X}_{\sigma_2}$ respectively, we make the heuristic assumption that their sum is drawn from $\mathcal{X}_{\sqrt{\sigma_1^2+\sigma_2^2}}$. We make the corresponding assumption for the rounded Gaussian distribution.

3 A Review of BKW-style Algorithms

3.1 The LWE Problem Reformulated

Assume that m samples

$$(\mathbf{a}_1, z_1), (\mathbf{a}_2, z_2), \ldots, (\mathbf{a}_m, z_m),$$

are collected from the LWE distribution $L_{\mathbf{s},\mathcal{X}}$, where $\mathbf{a}_i \in \mathbb{Z}_q^n, z_i \in \mathbb{Z}_q$. Let $\mathbf{z} = (z_1, z_2, \ldots, z_m)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_m) = \mathbf{sA}$. We have

$$\mathbf{z} = \mathbf{sA} + \mathbf{e},$$

where $\mathbf{A} = \begin{bmatrix}\mathbf{a}_1^{\mathrm{T}} & \mathbf{a}_2^{\mathrm{T}} & \cdots & \mathbf{a}_m^{\mathrm{T}}\end{bmatrix}$, $z_i = y_i + e_i = \langle \mathbf{s}, \mathbf{a}_i \rangle + e_i$ and $e_i \xleftarrow{\$} \mathcal{X}$. The search LWE problem is a decoding problem, where $\mathbf{A}$ serves as the generator matrix for a linear code over $\mathbb{Z}_q$ and $\mathbf{z}$ is a received word. Finding the secret vector $\mathbf{s}$ is equivalent to finding the codeword $\mathbf{y} = \mathbf{sA}$ for which the Euclidean distance $||\mathbf{y} - \mathbf{z}||$ is minimal. In the sequel, we adopt the notation $\mathbf{a}_i = (a_{i1}, a_{i2}, \ldots, a_{in})$.

3.2 Transforming the Secret Distribution

A transformation [8, 25] can be applied to ensure that the secret vector follows the same distribution $\mathcal{X}$ as the noise. It is done as follows. We write $\mathbf{A}$ in systematic form via Gaussian elimination. Assume that the first n columns are linearly independent and form the matrix $\mathbf{A_0}$. Define $\mathbf{D} = {\mathbf{A_0}}^{-1}$ and write $\hat{\mathbf{s}} = \mathbf{sD}^{-1} - (z_1, z_2, \ldots, z_n)$. Hence, we can derive an equivalent problem described by $\hat{\mathbf{A}} = (\mathbf{I}, \hat{\mathbf{a}}_{n+1}^{\mathrm{T}}, \hat{\mathbf{a}}_{n+2}^{\mathrm{T}}, \cdots, \hat{\mathbf{a}}_m^{\mathrm{T}})$, where $\hat{\mathbf{A}} = \mathbf{DA}$. We compute

$$\hat{\mathbf{z}} = \mathbf{z} - (z_1, z_2, \ldots, z_n)\hat{\mathbf{A}} = (\mathbf{0}, \hat{z}_{n+1}, \hat{z}_{n+2}, \ldots, \hat{z}_m).$$

Using this transformation, each entry in the secret vector $\mathbf{s}$ is now distributed according to $\mathcal{X}$. The fact that entries in $\mathbf{s}$ are small is a very useful property in several of the known reduction algorithms for solving LWE.

The noise distribution $\mathcal{X}$ is usually chosen as the discrete Gaussian distribution or the rounded Gaussian Distribution from Section 2.3.

3.3 Sample Amplification

In some versions of the LWE problem, such as the Darmstadt Challenges [2], the number of available samples is limited. To get more samples, sample amplification can be used. For example, assume that we have M samples $(\mathbf{a}_1, b_1)$, $(\mathbf{a}_2, b_2)$, ..., $(\mathbf{a}_M, b_M)$. Then we can form new samples, using an index set I of size k, as

$$\left(\sum_{j \in I} \pm \mathbf{a}_j, \sum_{j \in I} \pm b_j \right). \tag{1}$$

Given an initial number of samples M we can produce up to $2^{k-1}\binom{M}{k}$ samples. This comes at a cost of increasing the noise level (standard deviation) to $\sqrt{k} \cdot \sigma$. This also increases the sample dependency.

3.4 Iterating and Guessing

BKW-style algorithms work by combining samples in many steps in such a way that we reach a system of equations over $\mathbb{Z}_q$ of the form $\mathbf{z} = \mathbf{sA} + \mathbf{E}$, where $\mathbf{E} = (E_1, E_2, \ldots, E_m)$ and the entries $E_i, i = 1, 2, \ldots, m$ are sums of not too many original noise vectors, say $E_i = \sum_{j=1}^{2^t} e_{i_j}$, and where t is the number of iterations. The process also reduces the norm of column vectors in $\mathbf{A}$ to be small. Let $n_i, i = 1, 2, \ldots, t$ denote the number of reduced positions in step i and let $N_i = \sum_{j=1}^{i} n_j$. If $n = N_t$, then every reduced equation is of form

$$z_i = \langle \mathbf{a_i}, \mathbf{s} \rangle + E_i, \tag{2}$$

for $i = 1, 2, \ldots, m$. The right hand side can be approximated as a sample drawn from a discrete Gaussian and if the standard deviation is not too large, then the sequence of samples $z_1, z_2, \ldots$ can be distinguished from a uniform distribution. We will then need to determine the number of required samples to distinguish between the uniform distribution on $\mathbb{Z}_q$ and $\mathcal{X}_\sigma$. Relying on standard theory from statistics, using either previous work [28] or Bleichenbacher's definition of bias [32], we can find that the required number of samples is roughly

$$C \cdot e^{2\pi \left(\frac{\sigma\sqrt{2\pi}}{q} \right)^2}, \tag{3}$$

where C is a small positive constant. Initially, an optimal but exhaustive distinguisher was used [10]. While minimizing the sample complexity, it was slow and limited the number of positions that could be guessed. This basic approach was improved in [16], using the FFT. This was in turn a generalization of the corresponding distinguisher for LPN, which used the FWHT [27].

3.5 Plain BKW

The basic BKW algorithm was originally developed for solving LPN in [13]. It was first applied to LWE in [4]. The reduction part of this approach means that we reduce a fixed number b of positions in the column vectors of $\mathbf{A}$ to zero in each step. In each iteration, the dimension of $\mathbf{A}$ is decreased by b and after t iterations the dimension has decreased by bt.

3.6 Coded-BKW and LMS

LMS was introduced in [6] and improved in [26]. Coded-BKW was introduced in [23]. Both methods reduce positions in the columns of $\mathbf{A}$ to a small magnitude, but not to zero, allowing reduction of more positions per step. In LMS this is achieved by mapping samples to the same category if the n_i considered positions give the same result when integer divided by a suitable parameter p. In coded-BKW this is instead achieved by mapping samples to the same category if they are close to the same codeword in an $[n_i, k_i]$ linear code, for a suitable value k_i. Samples mapped to the same category give rise to new samples by subtracting them. The main idea [23, 26] is that positions in later iterations do not need to be reduced as much as the first ones, giving different n_i values in different steps.

3.7 LF1, LF2, Unnatural Selection

Each step of the reduction part of the BKW algorithm consists of two parts. First samples are mapped to categories depending on their position values on the currently relevant n_i positions. Next, pairs of samples within the categories are added/subtracted to reduce the current n_i positions to form a new generation of samples. This can be done in a couple of different ways.

Originally this was done using what is called LF1. Here we pick a representative from each category and form new samples by adding/subtracting samples to/from this sample. This approach makes the final samples independent, but also gradually decreases the sample size. In [27] the approach called LF2 was introduced. Here we add/subtract every possible pair within each category to form new samples. This approach requires only 3 samples within each category to form a new generation of the same size. The final samples are no longer independent, but experiments have shown that this effect is negligible.

In [6] unnatural selection was introduced.The idea is to produce more samples than needed from each category, but only keep the best samples, typically the ones with minimum norm on the current N_i positions in the columns of $\mathbf{A}$.

3.8 Coded-BKW with Sieving

When using coded-BKW or LMS, the previously reduced N_{i-1} positions of the columns of $\mathbf{A}$ increase in magnitude with an average factor $\sqrt{2}$ in each reduction step. This problem was addressed in [21] by using unnatural selection to only produce samples that kept the magnitude of the previous N_{i-1} positions small. Instead of testing all possible pairs of samples within the categories, this procedure was sped-up using lattice sieving techniques of [11]. This approach was slightly improved in [22, 30].

4 BKW-style Reduction Using Smooth-LMS

In this section we introduce a new reduction algorithm solving the problem of having the same complexity and memory usage in each iteration of a BKW-style reduction. The novel idea is to use simple LMS to reduce a certain number of positions and then partially reduce one extra position. This allows for balancing the complexity among the steps and hence to reduce more positions in total.

4.1 A New BKW-style Step

Assume having a large set of samples written as before in the form $\mathbf{z} = \mathbf{sA} + \mathbf{e} \bmod q$. Assume also that the entries of the secret vector $\mathbf{s}$ are drawn from some restricted distribution with small standard deviation (compared to the alphabet size q). If this is not the case, the transformation from Section 3.2 should be applied. Moreover, in case the later distinguishing process involves some positions to be guessed or transformed, we assume that this has been already considered and all positions in our coming description should be reduced.

The goal of this BKW-type procedure is to make the norms of the column vectors of $\mathbf{A}$ small by adding and subtracting equations together in a number of steps. Having expressions of the form $z_i = \mathbf{sa}_i + E_i \bmod q$, if we can reach a case where $||\mathbf{a}_i||$ is not too large, then $\mathbf{sa}_i + E_i$ can be considered as a random variable drawn from a discrete Gaussian distribution $\mathcal{X}_\sigma$. Furthermore, $\mathcal{X}_\sigma \bmod q$ can be distinguished from a uniform distribution over $\mathbb{Z}_q$ if σ is not too large.

Now let us describe the new reduction procedure. Fix the number of reduction steps to be t. We will also fix a maximum list size to be 2^v, meaning that $\mathbf{A}$ can have at most 2^v columns. In each iteration i, we are going to reduce some positions to be upper limited in magnitude by C_i, for $i = 1, ..., t$. Namely, these positions that are fully treated in iteration i will only have values in the set $\{-C_i + 1, \ldots, 0, 1, \ldots, C_i - 1\}$ of size $2C_i - 1$. We do this by dividing up the q possible values into intervals of length C_i. We also adopt the notation $\beta_i = q/C_i$, which describes the number of intervals we divide up the positions into. We assume that $\beta_i > 2$.

First step. In the first iteration, assume that we have stored $\mathbf{A}$. We first compute the required compression starting in iteration 1 by computing C_1 (we will explain how later). We then evaluate how many positions n_1 that can be fully reduced by computing $n_1 = \lfloor v/\log \beta_1 \rfloor$. The position $n_1 + 1$ can be partially reduced to be in an interval of size C_1' fulfilling $\beta_1' \cdot \beta_1^{n_1} \cdot 3/2 \leq 2^v$, where $\beta_1' = q/C_1'$. Now we do an LMS step that "transfers between iterations" in the following way.

We run through all the columns of $\mathbf{A}$. For column i, we simply denote it as $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and we compute:

$$k_j = \begin{cases} x_j \text{ div } C_1, & x_1 \geq 0 \\ -x_j \text{ div } C_1, & x_1 < 0 \end{cases}, \quad \text{for} \quad j = 1, \ldots, n_1,$$

$$k_{n_1+1} = \begin{cases} x_{n_1+1} \text{ div } C_1', & x_1 \geq 0 \\ -x_{n_1+1} \text{ div } C_1', & x_1 < 0 \end{cases}.$$

The vector $K_i = (k_1, k_2, \ldots, k_{n_1+1})$ is now an index to a sorted list $\mathcal{L}$, storing these vectors[1]. Except for the inverting of values if $x_1 < 0$, samples should have the same index if and only if all position values are the same when integer divided by C_1 (C_1' for the last position). So we assign $\mathcal{L}(K_i) = \mathcal{L}(K_i) \cup \{i\}$. After we have inserted all columns into the list $\mathcal{L}$, we go to the combining part.

[1]The point of inverting all position values if $x_1 < 0$ is to make sure that samples that get reduced when added should be given the same index. For example $(x_1, x_2, \ldots, x_{n_1+1})$ and $(-x_1, -x_2, \ldots, -x_{n_1+1})$ are mapped to the same category.

We build a new matrix $\mathbf{A}$ in the following way. Run through all indices K and if $|\mathcal{L}(K)| \geq 2$ combine every pair of vectors in $\mathcal{L}(K)$ by subtracting/adding[2] them to form a new column in the new matrix $\mathbf{A}$. Stop when the number of new columns has reached 2^v. For each column in $\mathbf{A}$ we have that:

- the absolute value of each position $j \in \{1, \ldots, n_1\}$ is $< C_1$,
- the absolute value of position $n_1 + 1$ is $< C_1'$.

Next steps. We now describe all the next iterations, numbered as $l = 2, 3, \ldots, t$. Iteration l will involve positions from $N_{l-1} + 1 = \sum_{i=1}^{l-1} n_i + 1$ to $N_l + 1$. The very first position has possibly already been partially reduced and its absolute value is $< C_{l-1}'$, so the interval for possible values is of size $2C_{l-1}' - 1$. Assume that the desired interval size in iteration l is C_l. In order to achieve the corresponding reduction factor β_l, we split this interval in $\beta_l'' = (2C_{l-1}' - 1)/C_l$ subintervals. We then compute how many positions n_l that can be fully reduced by computing $n_l = \lfloor (v - \log \beta_l'')/\log \beta_l \rfloor$. The position $N_l + 1$ can finally be partially reduced to be in an interval of size C_l' fulfilling $\beta_l' \cdot \beta_l^{n_l - 1} \beta_l'' \cdot 3/2 \leq 2^v$, where $\beta_l' = q/C_l'$.

Similar to iteration 1, we run through all the columns of $\mathbf{A}$. For each column i in the matrix $\mathbf{A}$ denoted as $\mathbf{x}$ we do the following. For each vector position in $\{N_{l-1} + 1, \ldots, N_l + 1\}$, we compute (here div means integer division)

$$
\begin{aligned}
k_j &= \begin{cases} x_{N_{l-1}+j} \text{ div } C_l, & x_{N_{l-1}+1} \geq 0 \\ -x_{N_{l-1}+j} \text{ div } C_l, & x_{N_{l-1}+1} < 0 \end{cases}, \quad \text{for} \quad j = 1, \ldots, n_l, \\
k_{n_l} &= \begin{cases} x_{N_l+1} \text{ div } C_l', & x_{N_{l-1}+1} \geq 0 \\ -x_{N_l+1} \text{ div } C_l', & x_{N_{l-1}+1} < 0 \end{cases}.
\end{aligned} \tag{4}
$$

The vector $K = (k_1, k_2, \ldots, k_{n_{l+1}})$ is again an index to a sorted list $\mathcal{L}$, keeping track of columns[3]. So again we assign $\mathcal{L}(K) = \mathcal{L}(K) \cup \{i\}$. After we have inserted all column indices into the list $\mathcal{L}$, we go to the combining part.

As in the first step, we build a new $\mathbf{A}$ as follows. Run through all indices K and if $|\mathcal{L}(K)| \geq 2$ combine every pair of vectors by adding/subtracting them to form a column in the new matrix $\mathbf{A}$. Stop when the number of new columns has reached 2^v.

For the last iteration, since N_t is the last row of $\mathbf{A}$, one applies the same step as above but without reducing the extra position. After t iterations, one gets equations on the form (2), where the $\mathbf{a}_i$ vectors in $\mathbf{A}$ have reduced norm.

4.2 Smooth-Plain BKW

The procedure described above also applies to plain BKW steps. For example, if in the first iteration one sets $C_1 = 1$ and $C_1' > 1$, then each column vector $\mathbf{x}$ of $\mathbf{A}$ will be reduced such that $x_1 = \ldots = x_{n_1} = 0$ and $x_{n_1+1} \in \{-C_1' + 1, \ldots, C_1' - 1\}$. Thus, one can either continue with another smooth-Plain BKW step by setting also $C_2 = 1$ in the second iteration, or switch to smooth-LMS. In both cases,

[2] Depending on what reduces the sample the most.

[3] Also here the point of inverting all position values if $x_{N_{l-1}+1} < 0$ is to make sure that samples that get reduced when added should be given the same index. For example $(x_{N_{l-1}+1}, x_{N_{l-1}+2}, \ldots, x_{N_l+1})$ and $(-x_{N_{l-1}+1}, -x_{N_{l-1}+2}, \ldots, -x_{N_l+1})$ are mapped to the same category.

we have the advantage of having x_{n_1} already partially reduced. Using these smooth-Plain steps we can reduce a couple of extra positions in the plain pre-processing steps of the BKW algorithm.

4.3 How to Choose the Interval Sizes C_i

To achieve as small norm of the vectors as possible, we would like the variance of all positions to be equally large, after completing all iterations. Assume that a position x takes values uniformly in the set $\{-(C-1)/2, \ldots, 0, 1, \ldots, (C-1)/2\}$, for $C > 0$. Then, we have that in $\mathrm{Var}(x) = (C-1)(C+1)/12$. Assuming C is somewhat large, we approximately get $\mathrm{Var}(x) = C^2/12$. When subtracting/adding two such values, the variance increases to $2\mathrm{Var}(x)$ in each iteration. Therefore, a reduced position will have an expected growth of $\sqrt{2}$. For this reason, we choose a relation for the interval sizes of the form

$$C_i = 2^{-(t-i)/2} C_t, \quad i = 1, \ldots, t-1.$$

This makes the variance of each position roughly the same, after completing all iterations. In particular, our vectors $||\mathbf{a}_i||$ in $\mathbf{A}$ are expected to have norm at most $\sqrt{n}C_t/\sqrt{12}$, and C_t is determined according to the final noise allowed in the guessing phase. Ignoring the pre-processing step with smooth-Plain BKW steps, the maximum dimension n that can be reduced is then $n = N_t = \sum_{i=1}^{t} n_i$.

Example 4.1. *Let $q = 1601$ and $\alpha = 0.005$, so $\sigma = \alpha q \approx 8$. Let us compute how many positions that can be reduced using $2^v = 2^{28}$ list entries. The idea is that the variance of the right hand side in (2) should be minimized by making the variance of the two terms roughly equal. The error part E_i is the sum of 2^t initial errors, so its variance is $\mathrm{Var}(E_i) = 2^t\sigma^2$. In order to be able to distinguish the samples according to (3), we set $\mathrm{Var}(E_i) < q^2/2$. This will give us the number of iterations possible as $2^t\sigma^2 \approx q^2/2$ or $2^t \approx 1601^2/(2 \cdot 8^2)$ leading to $t = 14$. Now we bound the variance of the scalar product part of (2) also to be $< q^2/2$, so $n\sigma^2 C_t^2/12 \approx q^2/2$ leading to $C_t^2 \approx 12q^2/(2n\sigma^2)$ and $C_t^2 \approx 12 \cdot 1601^2/(2n \cdot 8^2)$ or $C_t \approx 80$ if $n < 38$. Then one chooses $C_{t-1} = \lfloor C_t/\sqrt{2} \rceil = 57$ and so on.*

4.4 Unnatural Selection

We can improve performance by using the unnatural selection discussed in Section 3.7. Let us make some basic observations. Combining n_l positions using interval size C gives as previously described a value in the set $\{-(C-1)/2, \ldots, 0, 1, \ldots (C-1)/2\}$, and results in $\mathrm{Var}(x) = (C-1)(C+1)/12$. Combining two vectors from the same category, a position value $y = x_1 + x_2$, where x_1, x_2 are as above, results in a value in the interval $\{-(C-1), \ldots, 0, 1, \ldots (C-1)\}$ with variance $\mathrm{Var}(y) = (C-1)(C+1)/6$. Now observe that for the resulting reduced positions, smaller values are much more probable than larger ones.

5 A Binary Partial Guessing Approach

In this section we propose a new way of reducing the guessing step to a binary version. This way, we are able to efficiently use the FWHT to guess many entries in a small number of operations. In Section 6 we do the theoretical analysis and show that this indeed leads to a more efficient procedure than all previous ones.

5.1 From LWE to LPN

First, we need to introduce a slight modification to the original system of equations *before* the reduction part. Assume that we have turned the distribution of $\mathbf{s}$ to be the noise distribution, through the standard transformation described in Section 3.2. The result after this is written as before

$$\mathbf{z} = \mathbf{sA} + \mathbf{e}. \tag{5}$$

Now we perform a multiplication by 2 to each equation, resulting in

$$\mathbf{z}' = \mathbf{sA}' + 2\mathbf{e},$$

since when multiplied with a known value, we can compute the result modulo q.

Next, we apply the reduction steps and make the values in $\mathbf{A}'$ as small as possible by performing BKW-like steps. In our case we apply the smooth-LMS step from the previous section, but any other reduction method like coded-BKW with sieving would be possible. If $\mathbf{A}' = \begin{bmatrix}\mathbf{a}_1^{\mathrm{T}} & \mathbf{a}_2^{\mathrm{T}} & \cdots & \mathbf{a}_m^{\mathrm{T}}\end{bmatrix}$ the output of this step is a matrix where the Euclidean norm of each $\mathbf{a}_i$ is small. The result is written as

$$\mathbf{z}'' = \mathbf{sA}'' + 2\mathbf{E}, \tag{6}$$

where $\mathbf{E} = (E_1, E_2, \ldots, E_m)$ and $E_i = \sum_{j=1}^{2^t} e_{i_j}$ as before.

Finally, we transform the entire system to the binary case by considering

$$\mathbf{z}_0'' = \mathbf{s}_0\mathbf{A}_\mathbf{0}'' + \mathbf{e} \bmod 2, \tag{7}$$

where $\mathbf{z}_0''$ is the vector of least significant bits in $\mathbf{z}''$, $\mathbf{s}_0$ the vector of least significant bits in $\mathbf{s}$, $\mathbf{A}_\mathbf{0}'' = (\mathbf{A}'' \bmod 2)$ and $\mathbf{e}$ denotes the binary error introduced.

We can now examine the error e_j in position j of $\mathbf{e}$. In (6) we have equations of the form $z_j = \sum_i s_i a_{ij} + 2E_j$ in $\mathbb{Z}_q$, which can be written on integer form as

$$z_j = \sum_i s_i a_{ij} + 2E_j + k_j \cdot q. \tag{8}$$

Now if $|\sum_i s_i a_{ij} + 2E_j| < q/2$ then $k_j = 0$. In this case (8) can be reduced mod 2 without error and $e_j = 0$. In general, the error is computed as $e_j = k_j \bmod 2$. So one can compute a distribution for $e_j = k_j \bmod 2$ by computing $P(k_j = x)$. It is possible to compute such distribution either making a general approximation or precisely for each specific position j using the known values $\mathbf{a}_j$ and z_j. Note that the distribution of e_j depends on z_j. Also note that if $\mathbf{a}_j$ is reduced to a small norm and the number of steps t is not too large, then it is quite likely that $|\sum_i s_i a_{ij} + 2E_j| < q/2$ leading to $P(e_j = 0)$ being large.

For the binary system, we finally need to find the secret value $\mathbf{s}_0$. Either

1. there are no errors (or almost no errors), corresponding to $P(e_j = 0) \approx 1$. Then one can solve for $\mathbf{s}_0$ directly using Gaussian elimination (or possibly some information set decoding algorithm in the case of a few possible errors).
2. or the noise is larger. The binary system of equations corresponds to the situation of a fast correlation attack [31], or secret recovery in an LPN problem [13]. Thus, one may apply an FWHT to recover the binary secret values.

5.2 Guessing $\mathbf{s}_0$ Using the FWHT

The approach of using the FWHT to find the most likely $\mathbf{s}_0$ in the binary system in (7) comes directly from previous literature on Fast Correlation Attacks [14].

Let k denote an n-bit vector $(k_0, k_1, \ldots, k_{n-1})$ (also considered as an integer) and consider a sequence $X_k, k = 0, 1, \ldots, N-1$, $N = 2^n$. It can for example be a sequence of occurrence values in the time domain, e.g. $X_k =$ the number of occurrences of $X = k$. The Walsh-Hadamard Transform is defined as

$$\hat{X}_w = \sum_{k=0}^{N-1} X_k \cdot (-1)^{w \cdot k},$$

where $w \cdot k$ denotes the bitwise dot product of the binary representation of the n-bit indices w and k. There exists an efficient method (FWHT) to compute the WHT in time $O(N \log N)$. Given the matrix $\mathbf{A}''_\mathbf{0}$, we define $X_k = \sum_{j \in J} (-1)^{z''_j}$, where J is the set of all columns of the matrix $\mathbf{A}''_\mathbf{0}$ that equal k. Then, one computes $\max_w |\hat{X}_w|$, and we have that $\mathbf{s}_0$ corresponds to $\bar{w}$ such that $|\hat{X}_{\bar{w}}| = \max_w |\hat{X}_w|$. In addition, $\hat{X}_w$ is simply the (biased) sum of the noise terms.

Soft Received Information

The bias of $\hat{X}_w$ actually depends on the value of z''_j. So a slightly better approach is to use "soft received information" by defining $X_k = \sum_{j \in J} (-1)^{z''_j} \cdot \epsilon_{z''_j}$, where $\epsilon_{z''_j}$ is the bias corresponding to z''_j. For each $x \in \{-(q-1)/2, ..., (q-1)/2\}$, the bias ϵ_x can be efficiently pre-computed so that its evaluation does not affect the overall complexity of the guessing procedure.

Hybrid Guessing

One can use hybrid approach to balance the overall complexity among reduction and guessing phases. Indeed, it is possible to leave some rows of the matrix $\mathbf{A}$ unreduced and apply an exhaustive search over the corresponding positions in combination with the previously described guessing step.

5.3 Retrieving the Original Secret

Once $\mathbf{s}_0$ is correctly guessed, it is possible to obtain a new LWE problem instance with the secret half as big as follows. Write $\mathbf{s} = 2\mathbf{s}' + \mathbf{s}_0$. Define $\hat{\mathbf{A}} = 2\mathbf{A}$ and $\hat{\mathbf{z}} = \mathbf{z} - \mathbf{s}_0\mathbf{A}$. Then we have that

$$\hat{\mathbf{z}} = \mathbf{s}'\hat{\mathbf{A}} + \mathbf{e}. \tag{9}$$

The entries of $\mathbf{s}'$ have a bit-size half as large as the entries of $\mathbf{s}$, therefore (9) is an easier problem than (5). One can apply the procedure described above to (9) and guess the new binary secret $\mathbf{s}_1$, i.e. the least significant bits of $\mathbf{s}'$. The cost of doing this will be significantly smaller as shorter secret translates to computationally easier reduction steps. Thus, computationally speaking, the LWE problem can be considered solved once we manage to guess the least significant bits of $\mathbf{s}$. Given the list of binary vectors $\mathbf{s}_0, \mathbf{s}_1, \mathbf{s}_2, ...$, it is easy to retrieve the original secret $\mathbf{s}$.

Algorithm 1 BKW-FWHT with smooth reduction (main framework)

Input: Matrix $\mathbf{A}$ with n rows and m columns, received vector $\mathbf{z}$ of length m and algorithm parameters $t_1, t_2, t_3, n_{limit}, \sigma_{set}$

Step 0: Use Gaussian elimination to change the distribution of the secret vector;
Step 1: Use t_1 smooth-plain BKW steps to remove the bottom n_{pbkw} entries;
Step 2: Use t_2 smooth-LMS steps to reduce n_{cod1} more entries;
Step 3: Perform the multiplying-2 operations;
Step 4: Use t_3 smooth-LMS steps to reduce the remaining $n_t \leq n_{limit}$ entries;
Step 5: Transform all the samples to the binary field and recover the partial secret key by the FWHT. We can exhaustively guess some positions.

6 Analysis of the Algorithm and its Complexity

In this section, we describe in detail the newly-proposed algorithm called BKW-FWHT with smooth reduction (BKW-FWHT-SR).

6.1 The Algorithm

The main steps of the new BKW-FWHT-SR algorithm are described in Algorithm 1. We start by changing the distribution of the secret vector with the secret-noise transformation [8], if necessary.

The general framework is similar to the coded-BKW with sieving procedure proposed in [21]. In our implementation, we instantiated coded-BKW with sieving steps with smooth-LMS steps discussed before for the ease of implementation.

The different part of the new algorithm is that after certain reduction steps, we perform a multiplication by 2 to each reduced sample as described in Section 5. We then continue reducing the remain positions and perform the mod 2 operations to transform the entire system to the binary case. Now we obtain a list of LPN samples and solve the corresponding LPN instance via known techniques such as FWHT or partial guessing.

One high level description is that we aim to input an LWE instance to the LWE-to-LPN transform developed in Section 5, and solve the instance by using a solver for LPN. To optimize the performance, we first perform some reduction steps to have a new LWE instance with reduced dimension but larger noise. We then feed the obtained instance to the LWE-to-LPN transform.

6.2 The Complexity of Each Step

From now on we assume that the secret is already distributed as the noise distribution or that the secret-noise transform is performed. We use the LF2 heuristics and assume the the sample size is unchanged before and after each reduction step. We now start with smooth-plain BKW steps and let l_{red} be the number of positions already reduced.

Smooth-Plain BKW steps.

Given m initial samples, we could on average have $\lfloor \frac{2m}{3} \rfloor$ categories[4] for one plain BKW step in the LF2 setting. Instead we could assume for 2^{b_0} categories, and thus the number of samples m is $1.5 \cdot 2^{b_0}$. Let C_{pBKW} be the cost of all smooth-plain BKW steps, whose initial value is set to be 0. If a step starts with a position never being reduced before, we can reduce l_p positions, where $l_p = \left\lfloor \frac{b}{\log_2(q)} \right\rfloor$. Otherwise, when the first position is partially reduced in the previous step and we need β' categories to further reduce this position, we can in total fully reduce l_p positions, where $l_p = 1 + \left\lfloor \frac{b - \log_2(\beta')}{\log_2(q)} \right\rfloor$.

For this smooth-plain BKW step, we compute

$$C_{pbkw} \mathrel{+}= ((n + 1 - l_{red}) \cdot m + C_{d,pbkw}),$$

where $C_{d,pbkw} = m$ is the cost of modulus switching for the last partially reduced position in this step. We then update the number of the reduced positions, $l_{red} \mathrel{+}= l_p$.

After iterating for t_1 times, we could compute C_{pbkw} and l_{red}. We will continue updating l_{red} and denote n_{pbkw} the length reduced by the smooth-plain BKW steps.

Smooth-LMS steps before the multiplication of 2.

We assume that the final noise contribution from each position reduced by LMS is similar, bounded by a preset value σ_{set}. Since the noise variable generated in the i-th $(0 \leq i \leq t_2 - 1)$ Smooth-LMS step will be added by $2^{t_2+t_3-i}$ times and also be multiplied by 2, we compute $\sigma_{set}^2 = \frac{2^{t_2+t_3-i} \times 4C_{i,LMS1}^2}{12}$, where $C_{i,LMS1}$ is the length of the interval after the LMS reduction in this step. We use $\beta_{i,LMS1}$ categories for one position, where $\beta_{i,LMS1} = \lceil \frac{q}{C_{i,LMS1}} \rceil$. Similar to smooth-plain BKW steps, if this step starts with an new position, we can reduce l_p positions, where $l_p = \lfloor \frac{b}{\log_2(\beta_{i,LMS1})} \rfloor$. Otherwise, when the first position is partially reduced in the previous step and we need $\beta'_{p,i,LMS1}$ categories to further reduce this position, we can in total fully reduce l_p positions, where $l_p = 1 + \lfloor \frac{b - \log_2(\beta'_{p,i,LMS1})}{\log_2(\beta_{i,LMS1})} \rfloor$. Let C_{LMS1} be the cost of Smooth-LMS steps before the multiplication of 2, which is innitialized to 0. For this step, we compute

$$C_{LMS1} \mathrel{+}= (n + 1 - l_{red}) \cdot m,$$

and then update the number of the reduced positions, $l_{red} \mathrel{+}= l_p$.

After iterating t_2 times, we compute C_{LMS1} and l_{red}. We expect $l_{red} = n - n_t$ $(n_t \leq n_{limit})$ positions have been fully reduced and will continue updating l_{red}.

Smooth-LMS steps after the multiplication of 2.

The formulas are similar to those for Smooth-LMS steps before the multiplication of 2. The difference is that the noise term is no longer multiplied by 2, so

[4]The number of categories is doubled compared with the LF2 setting for LPN. The difference is that we could add and subtract samples for LWE.

σ_f	q	$D(\mathcal{X}_{\sigma_f,2q}\|\|U_{2q})$	$\mathbb{E}_{z=t}[D(e_{z=t}\|\|U_b)]$
$0.5q$	1601	-2.974149	-2.974995
$0.6q$	1601	-4.577082	-4.577116
$0.7q$	1601	-6.442575	-6.442576
$0.8q$	1601	-8.582783	-8.582783

Table 1: The comparison between $D(\mathcal{X}_{\sigma_f,2q}||U_{2q})$ and $\mathbb{E}_{z=t}[D(e_{z=t}||U_b)]$

we have $\sigma^2_{set} = \frac{2^{t_3-i}C^2_{i,LMS2}}{12}$, for $0 \leq i \leq t_3 - 1$. Also, we need to track the a vector of length n_t for the later distinguisher. The cost is

$$C_{LMS2} = t_3 \cdot (n_t + 1) \cdot m.$$

We also need to count the cost for multiplying samples by 2 and the mod2 operations, and the LMS decoding cost, which are

$$C_{mulMod} = 2 \cdot (n_t + 1) \cdot m,$$
$$C_{dec} = (n - n_{pbkw} + t_2 + t_3) \cdot m.$$

FWHT distinguisher and partial guessing.

After the LWE-to-LPN transformation, we have an LPN problem with dimension n_t and m instance. We perform partial guessing on n_{guess} positions, and use FWHT to recover the remaining $n_{FWHT} = n_t - n_{guess}$ positions. The cost is,

$$C_{distin} = 2^{n_{guess}} \cdot ((n_{guess} + 1) \cdot m + n_{FWHT} \cdot 2^{n_{FWHT}}).$$

6.3 The Data Complexity

We now discuss the data complexity of the new FWHT distinguisher. In the integer form, we have the following equation,

$$z_j = \sum_{i=0}^{n_t-1} s_i a_{ij} + 2E_j + k_j \cdot q.$$

If $|\sum s_i a_{ij} + 2E_j| < q/2$ then $k_j = 0$. Then the equation can be reduced mod 2 without error. In general, the error is $e_j = k_j \bmod 2$.

We employ a smart FWHT distinguisher with soft received information, as described in Section 5. From [29], we know the sample complexity can be approximated as $m \approx \frac{4 \ln n_t}{\mathbb{E}_{z=t}[D(e_{z=t}||U_b)]}$.

For different value of z_j, the distribution of e_j is different. The maximum bias is achieved when $z_j = 0$. In this sense, we could compute the divergence as

$$\begin{aligned}\mathbb{E}_{z=t}[D(e_{z=t}||U_b)] &= \sum_{t\in\mathbb{Z}_q} \Pr[z=t]\, D(e_{z=t}||U_b) \\ &= \sum_{t\in\mathbb{Z}_q} \Pr[z=t]\,(\sum_{i=0}^{1} \Pr[e_{z=t}=i]\log(2\cdot\Pr[e_{z=t}=i]))\end{aligned}$$

where e_z is the Bernoulli variable conditioned on the value of z and U_b the uniform distribution over the binary field.

Following the previous research [4], we approximate the noise $\sum s_i a_{ij} + 2E_j$ as discrete Gaussian with standard deviation σ_f. If σ_f is large, the probability $\Pr[z = t]$ is very close to $1/q$. Then, the expectation $\mathbb{E}_{z=t,t\in\mathbb{Z}_q}[D(e_{z=t}||U_b)]$ can be approximated as

$$\sum_{t\in\mathbb{Z}_q}\sum_{i=0}^{1}\Pr[z=t]\Pr[e_{z=t}=i]\log(2q\cdot\Pr[e_{z=t}=i,z=t]),$$

i.e., the divergence between a discrete Gaussian with the same standard deviation and a uniform distribution over $2q$, $D(\mathcal{X}_{\sigma_f,2q}||U_{2q})$. We numerically computed that the approximation is rather accurate when the noise is sufficiently large (see Table 1). In conclusion, we use the formula

$$m \approx \frac{4\ln n_t}{D(\mathcal{X}_{\sigma_f,2q}||U_{2q})},$$

to estimate the data complexity of the new distinguisher. It remains to control the overall variance σ_f^2. Since we assume that the noise contribution from each reduced position by LMS is the same and the multiplication of 2 will double the standard deviation, we can derive $\sigma_f^2 = 4 * 2^{t_1+t_2+t_3}\sigma^2 + \sigma^2\sigma_{set}^2(n - n_{pbkw})$.

Note: The final noise is a combination of three parts, the noise from the LWE problem, the LMS steps before the multiplication by 2, and the LMS steps after the multiplication by 2. The final partial key recovery problem is equivalent to distinguishing a discrete Gaussian from uniform with the alphabet size doubled. We see that with the multiplication by 2, the variances of the first and the second noise parts are increased by a factor of 4, but the last noise part does not expand. This intuitively explains the gain of the new binary distinguisher.

6.4 In Summary

We have the following theorem to estimate the complexity of the attack.

Theorem 6.1. *The time complexity of the new algorithm is*

$$C = C_{pbkw} + C_{LMS1} + C_{LMS2} + C_{dec} + C_{distin} + C_{mulMod},$$

under the condition that

$$m \geq \frac{4\ln n_t}{D(\mathcal{X}_{\sigma_f,2q}||U_{2q})},$$

where $\sigma_f^2 = 4 * 2^{t_1+t_2+t_3}\sigma^2 + \sigma^2\sigma_{set}^2(n - n_{pbkw})$.

6.5 Numerical Estimation

We numerically estimate the complexity of the new algorithm BKW-FWHT-SR (shown in Table 2). It improves the known approaches when the noise rate (represented by α) becomes larger. We should note that compared with the previous BKW-type algorithms, the implementation is much easier though the complexity gain might be mild.

n	q	α		LWE estimator [7]						
			BKW-	Coded-	usvp		dec		dual	
			FWHT-SR	BKW	ENU	Sieve	ENU	Sieve	ENU	Sieve
40	1601	0.005	34.4	42.6	**31.4**	41.5	34.7	44.6	39.1	47.5
		0.010	39.3	43.7	**34.0**	44.8	36.3	44.9	51.1	57.9
		0.015	**42.4**	52.6	42.5	54.2	43.1	50.6	61.5	64.4
		0.020	**46.2**	52.6	∞	∞	51.9	58.2	73.1	75.9
		0.025	**48.3**	52.7	∞	∞	59.2	66.1	84.7	85.4
		0.030	**50.0**	52.7	∞	∞	67.1	68.9	96.3	92.5
45	2027	0.005	37.7	55.2	**31.8**	41.9	35.0	44.8	41.5	51.6
		0.010	43.5	55.2	**39.5**	51.2	41.2	48.2	57.0	64.6
		0.015	**48.3**	55.2	50.4	61.3	51.2	58.3	74.3	74.9
		0.020	**51.2**	55.2	∞	∞	61.1	65.0	86.8	86.1
		0.025	**54.1**	55.3	∞	∞	71.0	71.4	100.7	95.0
		0.030	**56.3**	64.1	∞	∞	80.2	78.7	116.2	104.1
50	2503	0.005	41.8	46.4	**32.4**	42.6	35.5	45.1	46.7	58.0
		0.010	48.7	56.0	**46.0**	57.5	47.6	54.1	66.8	65.4
		0.015	**52.5**	56.8	∞	∞	60.8	63.6	84.9	83.5
		0.020	**56.4**	61.9	∞	∞	72.1	72.1	101.9	96.5
		0.025	**59.3**	66.1	∞	∞	83.5	80.8	120.0	105.7
		0.030	**63.3**	66.3	∞	∞	94.2	89.1	134.0	115.6
70	4903	0.005	58.3	62.3	**52.3**	54.2	55.2	63.3	76.2	75.9
		0.010	**67.1**	73.7	∞	∞	80.4	77.1	111.3	98.9
		0.015	**73.3**	75.6	∞	∞	102.5	93.2	146.0	118.0
120	14401	0.005	100.1	110.5	133.0	**93.2**	135.5	111.4	181.9	133.2
		0.010	**115.1**	124.0	∞	∞	195.0	150.4	266.2	165.7
		0.015	**127.0**	136.8	∞	∞	246.4	183.2	334.0	209.8

Table 2: Estimated time complexity comparison (in $\log_2(\cdot)$) for solving LWE instances in the TU Darmstadt LWE challenge [2]. Here unlimited number of samples are assumed. The last columns show the complexity estimation from the LWE estimator [7]. "ENU" represents the enumeration cost model is employed and "Sieve" represents the sieving cost model is used. Bold-faced numbers are the smallest among the estimations with these different approaches.

7 A New BKW Algorithm Implementation for Large LWE Problem Instances

We have a new implementation of the BKW algorithm that is able to handle very large LWE problem instances. The code is written in C, and much care has been taken to be able to handle instances with a large number of samples.

A key success factor in the software design was to avoid unnecessary reliance on RAM, so we have employed file-based storage where necessary and practically possible. The implementation includes most known BKW reduction step, FFT and FWHT-based guessing methods, and hybrid guessing approaches.

For our experiments, presented in Section 8, we assembled a machine with an ASUS PRIME X399-A motherboard, a 4.0GHz Ryzen Threadripper 1950X processor and 128GiB of 2666MHz DDR4 RAM. While the machine was built from standard parts with a limited budget, we have primarily attempted to maximize the amount of RAM and the size and read/write speeds of the fast SSDs for overall ability to solve large LWE problem instances. We will make the implementation available as an open source repository.

We describe below how we dealt with some interesting performance issues.

File-based Sample Storage

The implementation does not assume that all samples can be stored in RAM, so instead they are stored on file in a special way. Samples are stored sorted into their respective categories. For simplicity, we have opted for a fixed maximum number of samples per category. The categories are stored sequentially on file, each containing its respective samples (possibly leaving some space if the categories are not full). A category mapping, unique for each reduction type, defines what category index a given sample belongs to[5].

Optional Sample Amplification

We support optional sample amplification. That is, if a problem instance has a limited number of initial samples (e.g., the Darmstadt LWE challenge), then it is possible to combine several of these to produce new samples (more, but with higher noise).

While this is very straightforward in theory, we have noticed considerable performance effects when this recombination is performed naïvely. For example, combining triplets of initial samples using a nested loop is problematic in practice for some instances, since some initial samples become over-represented – Some samples are used more often than others when implemented this way.

We have solved this by using a Linear Feedback Shift Register to efficiently and pseudo-randomly distribute the selection of initial samples more evenly.

Employing Meta-Categories

For some LWE problem instances, using a very high number of categories with few samples in each is a good option. This can be problematic to handle in an implementation, but we have used meta-categories to handle this situation. For example, using plain BKW reduction steps with modulus q and three positions, we end up with q^3 different categories. With q large, an option is to use only two out of the three position values in a vector to first map it into one out of q^2 different meta-categories. When processing the (meta-)categories, one then needs an additional pre-processing in form of a sorting step in order to divide the samples into their respective (non-meta) categories (based on all three position values), before proceeding as per usual.

We have used this implementation trick to, for example, implement plain BKW reduction for three positions. One may think of the process as brute-forcing one out of three positions in the reduction step.

Secret Guessing with FFT and FWHT

The same brute-forcing techniques are also useful for speeding up the guessing part of the solver. We have used this to improve the FFT and FWHT solvers in the corresponding way.

[5]In this section a category is defined slightly differently from the rest of the paper. A category together with its adjacent category are together what we simply refer to as a category in the rest of the paper.

For the FWHT case, if the number of positions to guess is too large for the RAM to handle, we leave some of them to brute-force. This case differs from the above by the fact that binary positions are brute-forced (so more positions can be handled) and that the corresponding entries in the samples must be reduced.

8 Experimental Results

In this section we report the experimental results obtained in solving some LWE problems. Our main goal was to confirm our theory and to prove that BKW algorithms can be used in practice to solve relatively large instances. Therefore, there is still room to run a more optimized code (for example, we did not use any parallelization in our experiments) and to make more optimal parameter choices (we generally used more samples than required and no brute-force guessing techniques were used).

We considered two different scenarios. In the first case, we assumed for each LWE instance to have access to an arbitrary large number of samples. Here we create the desired amount of samples ourselves[6]. In the second case, we considered instances with a limited number of samples. An LWE problem is "solved" when the binary secret is correctly guessed, for the reasons explained in Section 5.3.

Unlimited Number of Samples

We targeted the parameter choices of the TU Darmstadt challenges [2]. For each instance, we generated as many initial samples as needed according to our estimations. In Table 3 we report the details of the largest solved instances. Moreover, in Example 8.1 we present our parameter choices for one of these.

n	q	α	number of samples	running time
40	1601	0.005	45 million	12 minutes
40	1601	0.01	1.6 billion	12 hours
45	2027	0.005	1.1 billion	13 hours

Table 3: Experimental results on target parameters.

Example 8.1. *Let us consider an LWE instance with $n = 40$, $q = 1601$ and $\sigma = 0.005 \cdot q$. To successfully guess the secret, we first performed 8 smooth-plain BKW steps reducing 18 positions to zero. We used the following parameters.*

$$\begin{aligned} n_i = 2, \quad C_i = 1, \qquad & \textit{for} \quad i = 1, \ldots, 8, \\ (C'_1, C'_2, C'_3, C'_4, C'_5, C'_6, C'_7, C'_8) = \quad & (165, 30, 6, 1, 165, 30, 6, 1). \end{aligned}$$

Note that $C'_4 = C'_8 = 1$. In this way, we exploited the smoothness to zero 9 positions every 4 steps. For this reason, we start steps 5 and 9 by skipping one

[6] we used rounded Gaussian noise for simplicity of implementation.

position. Finally, we did 5 smooth-LMS steps using the following parameters:

$$\begin{aligned}(n_9, n_{10}, n_{11}, n_{12}, n_{13}) &= (3, 4, 4, 5, 6)\\ (C_9, C_{10}, C_{11}, C_{12}, C_{13}) &= (17, 24, 34, 46, 66)\\ (C'_9, C'_{10}, C'_{11}, C'_{12}) &= (46, 66, 23, 81).\end{aligned}$$

These parameters are chosen in such a way that the number of categories within each step is $\approx 13M$ *and* $C_i \approx \sqrt{2}C_{i-1}$*. We used* $\approx 40M$ *samples in each step so that each category contained 3 samples in average. This way we are guaranteed to have enough samples in each step.*

Limited Number of Samples

As a proof-of-concept, we solved the original TU Darmstadt LWE challenge instance [2] with parameters $n = 40$, $\alpha = 0.005$ and the number of samples limited to $m = 1600$. We did this by sample amplifying with triples of samples, taking 7 steps of smooth-plain BKW on 17 entries, 5 steps of smooth-LMS on 22 entries and 1 position was left to brute-force. The overall running time was of 3 hours and 39 minutes.

9 Conclusions and Future Work

We introduced a novel and easy approach to implement the BKW reduction step which allows balancing the complexity among the iterations, and an FWHT-based guessing procedure able to correctly guess the secret with relatively large noise level. Together with a file-based approach of storing samples, the above define a new BKW algorithm specifically designed to solve practical LWE instances. We leave optimization of the implementation, including parallelization, for future work.

Acknowledgements

This work was supported in part by the Swedish Research Council (Grants No. 2015-04528 and 2019-04166), the Norwegian Research Council (Grant No. 247742/070), and the Swedish Foundation for Strategic Research (Grant No. RIT17-0005 and strategic mobility grant No. SM17-0062). This work was also partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

[1] NIST Post-Quantum Cryptography Standardization. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/

Post-Quantum-Cryptography-Standardization, accessed: 2018-09-24

[2] TU Darmstadt Learning with Errors Challenge. https://www.latticechallenge.org/lwe_challenge/challenge.php, accessed: 2020-05-01

[3] Albrecht, M., Cid, C., Faugere, J.C., Fitzpatrick, R., Perret, L.: On the complexity of the Arora-Ge algorithm against LWE. Cryptology ePrint Archive, Report 2014/1018 (2014), https://eprint.iacr.org/2014/1018

[4] Albrecht, M.R., Cid, C., Faugère, J., Fitzpatrick, R., Perret, L.: On the complexity of the BKW algorithm on LWE. Des. Codes Cryptogr. 74(2), 325–354 (2015)

[5] Albrecht, M.R., Ducas, L., Herold, G., Kirshanova, E., Postlethwaite, E.W., Stevens, M.: The general sieve kernel and new records in lattice reduction. In: Rijmen, V., Ishai, Y. (eds.) Advances in Cryptology – EUROCRYPT 2019, Part II. pp. 717–746. Lecture Notes in Computer Science, Springer, Heidelberg, Germany, Darmstadt, Germany (May 19–23, 2019)

[6] Albrecht, M.R., Faugère, J.C., Fitzpatrick, R., Perret, L.: Lazy modulus switching for the BKW algorithm on LWE. In: Krawczyk, H. (ed.) PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography. Lecture Notes in Computer Science, vol. 8383, pp. 429–445. Springer, Heidelberg, Germany, Buenos Aires, Argentina (Mar 26–28, 2014)

[7] Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. J. Mathematical Cryptology 9(3), 169–203 (2015)

[8] Applebaum, B., Cash, D., Peikert, C., Sahai, A.: Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In: Halevi, S. (ed.) Advances in Cryptology – CRYPTO 2009. Lecture Notes in Computer Science, vol. 5677, pp. 595–618. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2009)

[9] Arora, S., Ge, R.: New algorithms for learning in presence of errors. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011: 38th International Colloquium on Automata, Languages and Programming, Part I. Lecture Notes in Computer Science, vol. 6755, pp. 403–415. Springer, Heidelberg, Germany, Zurich, Switzerland (Jul 4–8, 2011)

[10] Baignères, T., Junod, P., Vaudenay, S.: How far can we go beyond linear cryptanalysis? In: Lee, P.J. (ed.) Advances in Cryptology – ASIACRYPT 2004. Lecture Notes in Computer Science, vol. 3329, pp. 432–450. Springer, Heidelberg, Germany, Jeju Island, Korea (Dec 5–9, 2004)

[11] Becker, A., Ducas, L., Gama, N., Laarhoven, T.: New directions in nearest neighbor searching with applications to lattice sieving. In: Krauthgamer, R. (ed.) 27th Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 10–24. ACM-SIAM, Arlington, VA, USA (Jan 10–12, 2016)

[12] Blum, A., Furst, M.L., Kearns, M.J., Lipton, R.J.: Cryptographic primitives based on hard learning problems. In: Stinson, D.R. (ed.) Advances in Cryptology – CRYPTO'93. Lecture Notes in Computer Science, vol. 773, pp. 278–291. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 22–26, 1994)

[13] Blum, A., Kalai, A., Wasserman, H.: Noise-tolerant learning, the parity problem, and the statistical query model. In: 32nd Annual ACM Symposium on Theory of Computing. pp. 435–440. ACM Press, Portland, OR, USA (May 21–23, 2000)

[14] Chose, P., Joux, A., Mitton, M.: Fast correlation attacks: An algorithmic point of view. In: Knudsen, L.R. (ed.) Advances in Cryptology — EUROCRYPT 2002. pp. 209–221. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)

[15] Delaplace, C., Esser, A., May, A.: Improved low-memory subset sum and LPN algorithms via multiple collisions. In: 17th IMA International Conference on Cryptography and Coding. pp. 178–199. Lecture Notes in Computer Science, Springer, Heidelberg, Germany, Oxford, UK (Dec 2019)

[16] Duc, A., Tramèr, F., Vaudenay, S.: Better algorithms for LWE and LWR. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 173–202. Springer, Heidelberg, Germany, Sofia, Bulgaria (Apr 26–30, 2015)

[17] Esser, A., Heuer, F., Kübler, R., May, A., Sohler, C.: Dissection-BKW. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018, Part II. Lecture Notes in Computer Science, vol. 10992, pp. 638–666. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2018)

[18] Esser, A., Kübler, R., May, A.: LPN decoded. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology – CRYPTO 2017, Part II. Lecture Notes in Computer Science, vol. 10402, pp. 486–514. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2017)

[19] Guo, Q., Johansson, T., Löndahl, C.: Solving LPN using covering codes. In: Sarkar, P., Iwata, T. (eds.) Advances in Cryptology – ASIACRYPT 2014, Part I. Lecture Notes in Computer Science, vol. 8873, pp. 1–20. Springer, Heidelberg, Germany, Kaoshiung, Taiwan, R.O.C. (Dec 7–11, 2014)

[20] Guo, Q., Johansson, T., Löndahl, C.: Solving LPN using covering codes. Journal of Cryptology 33(1), 1–33 (Jan 2020)

[21] Guo, Q., Johansson, T., Mårtensson, E., Stankovski, P.: Coded-BKW with sieving. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology – ASIACRYPT 2017, Part I. Lecture Notes in Computer Science, vol. 10624, pp. 323–346. Springer, Heidelberg, Germany, Hong Kong, China (Dec 3–7, 2017)

[22] Guo, Q., Johansson, T., Mårtensson, E., Stankovski Wagner, P.: On the asymptotics of solving the LWE problem using coded-bkw with sieving. IEEE Trans. Inf. Theory 65(8), 5243–5259 (2019)

[23] Guo, Q., Johansson, T., Stankovski, P.: Coded-BKW: Solving LWE using lattice codes. In: Gennaro, R., Robshaw, M.J.B. (eds.) Advances in Cryptology – CRYPTO 2015, Part I. Lecture Notes in Computer Science, vol. 9215, pp. 23–42. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2015)

[24] Herold, G., Kirshanova, E., May, A.: On the asymptotic complexity of solving LWE. Designs, Codes and Cryptography 86(1), 55–83 (2018)

[25] Kirchner, P.: Improved generalized birthday attack. Cryptology ePrint Archive, Report 2011/377 (2011), http://eprint.iacr.org/2011/377

[26] Kirchner, P., Fouque, P.A.: An improved BKW algorithm for LWE with applications to cryptography and lattices. In: Gennaro, R., Robshaw, M.J.B. (eds.) Advances in Cryptology – CRYPTO 2015, Part I. Lecture Notes in Computer Science, vol. 9215, pp. 43–62. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2015)

[27] Levieil, É., Fouque, P.A.: An improved LPN algorithm. In: Prisco, R.D., Yung, M. (eds.) SCN 06: 5th International Conference on Security in Communication Networks. Lecture Notes in Computer Science, vol. 4116, pp. 348–359. Springer, Heidelberg, Germany, Maiori, Italy (Sep 6–8, 2006)

[28] Lindner, R., Peikert, C.: Better key sizes (and attacks) for LWE-based encryption. In: Kiayias, A. (ed.) Topics in Cryptology – CT-RSA 2011. Lecture Notes in Computer Science, vol. 6558, pp. 319–339. Springer, Heidelberg, Germany, San Francisco, CA, USA (Feb 14–18, 2011)

[29] Lu, Y., Meier, W., Vaudenay, S.: The conditional correlation attack: A practical attack on Bluetooth encryption. In: Shoup, V. (ed.) Advances in Cryptology – CRYPTO 2005. Lecture Notes in Computer Science, vol. 3621, pp. 97–117. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2005)

[30] Mårtensson, E.: The Asymptotic Complexity of Coded-BKW with Sieving Using Increasing Reduction Factors. In: 2019 IEEE International Symposium on Information Theory (ISIT). pp. 2579–2583 (2019)

[31] Meier, W., Staffelbach, O.: Fast correlation attacks on certain stream ciphers. Journal of Cryptology 1(3), 159–176 (Oct 1989)

[32] Mulder, E.D., Hutter, M., Marson, M.E., Pearson, P.: Using bleichenbacher's solution to the hidden number problem to attack nonce leaks in 384-bit ECDSA: extended version. J. Cryptographic Engineering 4(1), 33–45 (2014)

[33] Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th Annual ACM Symposium on Theory of Computing. pp. 84–93. ACM Press, Baltimore, MA, USA (May 22–24, 2005)

[34] Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: 35th Annual Symposium on Foundations of Computer Science. pp. 124–134. IEEE Computer Society Press, Santa Fe, New Mexico (Nov 20–22, 1994)

Paper III

Quantum Algorithms for the Approximate k-List Problem and their Application to Lattice Sieving

Lattice sieve algorithms are amongst the foremost methods of solving SVP. The asymptotically fastest known classical and quantum sieves solve SVP in a d-dimensional lattice in $2^{\mathsf{c}d+o(d)}$ time steps with $2^{\mathsf{c}'d+o(d)}$ memory for constants c, c'. In this work, we give various quantum sieving algorithms that trade computational steps for memory. We first give a quantum analogue of the classical k-Sieve algorithm [Herold–Kirshanova–Laarhoven, PKC'18] in the Quantum Random Access Memory (QRAM) model, achieving an algorithm that heuristically solves SVP in $2^{0.2989d+o(d)}$ time steps using $2^{0.1395d+o(d)}$ memory. This should be compared to the state-of-the-art algorithm [Laarhoven, Ph.D Thesis, 2015] which, in the same model, solves SVP in $2^{0.2653d+o(d)}$ time steps and memory. In the QRAM model these algorithms can be implemented using $\mathrm{poly}(d)$ width quantum circuits. Secondly, we frame the k-Sieve as the problem of k-clique listing in a graph and apply quantum k-clique finding techniques to the k-Sieve. Finally, we explore the large quantum memory regime by adapting parallel quantum search [Beals et al., Proc. Roy. Soc. A'13] to the 2-Sieve and giving an analysis in the quantum circuit model. We show how to heuristically solve SVP in $2^{0.1037d+o(d)}$ time steps using $2^{0.2075d+o(d)}$ quantum memory.

Keywords: shortest vector problem (SVP), lattice sieving, Grover's algorithm, approximate k-list problem, nearest neighbour algorithms, distributed computation.

Elena Kirshanova, Erik Mårtensson, Eamonn W. Postlethwaite and Subhayan Roy Moulik, "Quantum Algorithms for the Approximate k-List Problem and their Application to Lattice Sieving", in *Advances in Cryptology–ASIACRYPT 2019, the 25th Annual International Conference on Theory and Application of Cryptology and Information Security*, pp. 521-551, 2019, Kobe, Japan.

1 Introduction

The Shortest Vector Problem (SVP) is one of the central problems in the theory of lattices. For a given d-dimensional Euclidean lattice, usually described by a basis, to solve SVP one must find a shortest non zero vector in the lattice. This problem gives rise to a variety of efficient, versatile, and (believed to be) quantum resistant cryptographic constructions [AD97, Reg05]. To obtain an estimate for the security of these constructions it is important to understand the complexities of the fastest known algorithms for SVP.

There are two main families of algorithms for SVP, (1) algorithms that require $2^{\omega(d)}$ time and poly(d) memory; and (2) algorithms that require $2^{\Theta(d)}$ time and memory. The first family includes lattice enumeration algorithms [Kan83, GNR10]. The second contains sieving algorithms [AKS01,NV08,MV10], Voronoi cell based approaches [MV10] and others [ADRS15, BGJ14]. In practice, it is only enumeration and sieving algorithms that are currently competitive in large dimensions [ADH$^+$19, TKH18]. Practical variants of these algorithms rely on *heuristic* assumptions. For example we may not have a guarantee that the returned vector will solve SVP exactly (e.g. pruning techniques for enumeration [GNR10], lifting techniques for sieving [Duc18]), or that our algorithm will work as expected on arbitrary lattices (e.g. sieving algorithms may fail on orthogonal lattices). Yet these heuristics are natural for lattices often used in cryptographic constructions, and one does not require an exact solution to SVP to progress with cryptanalysis [ADH$^+$19]. Therefore, one usually relies on heuristic variants of SVP solvers for security estimates.

Among the various attractive features of lattice based cryptography is its potential resistance to attacks by quantum computers. In particular, there is no known quantum algorithm that solves SVP on an arbitrary lattice significantly faster than existing classical algorithms.[1] However, some quantum speed-ups for SVP algorithms are possible in general.

It was shown by Aono–Nguyen–Shen [ANS18] that enumeration algorithms for SVP can be sped up using the *quantum backtracking* algorithm of Montanaro [Mon18]. More precisely, with quantum enumeration one solves SVP on a d-dimensional lattice in time $2^{\frac{1}{4e}d\log d+o(d\log d)}$, a square root improvement over classical enumeration. This algorithm requires poly(d) classical and quantum memory. This bound holds for both provable and heuristic versions of enumeration. Quantum speed-ups for sieving algorithms have been considered by Laarhoven–Mosca–van de Pol [LMvdP15] and later by Laarhoven [Laa15]. The latter result presents various quantum sieving algorithms for SVP. One of them achieves time and classical memory of order $2^{0.2653d+o(d)}$ and requires poly(d) quantum memory. This is the best known quantum time complexity for heuristic sieving algorithms. Provable single exponential SVP solvers were considered in the quantum setting by Chen–Chang–Lai [CCL17]. Based on [ADRS15,DRS14], the authors describe a $2^{1.255d+o(d)}$ time, $2^{0.5d+o(d)}$ classical and poly(d) quantum memory algorithm for SVP. All heuristic and provable results rely on the classical memory being quantumly addressable.

A drawback of sieving algorithms is their large memory requirements. Initiated by Bai–Laarhoven–Stehlé, a line of work [BLS16, HK17, HKL18] gave a

[1] For some families of lattices, like ideal lattices, there exist quantum algorithms that solve a variant of SVP faster than classical algorithms, see [CDW17, PMHS19]. In this work, we consider arbitrary lattices.

family of heuristic sieving algorithms, called tuple lattice sieves, or k-Sieves for some fixed constant k, that offer time-memory trade-offs. Such trade-offs have proven important in the current fastest SVP solvers, as the ideas of tuple sieving offer significant speed-ups in practice, [ADH+19]. In this work, we explore various directions for *asymptotic* quantum accelerations of tuple sieves.

Our results.

1. In Section 4 we show how to use a quantum computer to speed up the k-Sieve of Bai–Laarhoven–Stehlé [BLS16] and its improvement due to Herold–Kirshanova–Laarhoven [HKL18] (Algorithms 2,3). One data point achieves a time complexity of $2^{0.2989d+o(d)}$, while requiring $2^{0.1395d+o(d)}$ classical memory and poly(d) width quantum circuits. In the $\texttt{Area} \times \texttt{Time}$ model this beats the previously best known algorithm [Laa15] of time and memory complexities $2^{0.2653d+o(d)}$; we almost halve the constant in the exponent for memory at the cost of a small increase in the respective constant for time.

2. Borrowing ideas from [Laa15] we give a quantum k-Sieve (Algorithm 7) that also exploits nearest neighbour techniques. For $k = 2$, we recover Laarhoven's $2^{0.2653d+o(d)}$ time and memory quantum algorithm.

3. In Section 5 the k-Sieve is reduced to listing k-cliques in a graph. By generalising the triangle finding algorithm of [BdWD+01] this approach leads to an algorithm that matches the performance of Algorithm 2, when optimised for time, for all k.

4. In Section 6 we specialise to listing 3-cliques (triangles) in a graph. Using the quantum triangle finding algorithm of [LGN17] allows us, in the *query model*,[2] to perform the 3-Sieve using $2^{0.3264d+o(d)}$ *queries*.

5. In Section 7 we describe a quantum circuit consisting only of gates from a universal gate set (e.g. CNOT and single qubit rotations) of depth $2^{0.1038d+o(d)}$ and width $2^{0.2075d+o(d)}$ that implements the 2-Sieve as proposed classically in [NV08]. In particular we consider exponential *quantum* memory to make significant improvements to the number of time steps. Our construction adapts the parallel search procedure of [BBG+13].

Our main results, quantum time-memory trade-offs for sieving algorithms, are summarised in Figure 1. When optimising for time a quantum 2-Sieve with locality sensitive filtering (LSF) remains the best algorithm. For $k \geq 3$ the speed-ups offered by LSF are less impressive, and one can achieve approximately the same asymptotic time complexity by considering quantum k-Sieve algorithms (without LSF) with $k \geq 10$ and far less memory.

All the results presented in this work are asymptotic in nature: our algorithms have time, classical memory, quantum memory complexities of orders $2^{\mathsf{c}d+o(d)}$, $2^{\mathsf{c}'d+o(d)}$, poly(d) or $2^{\mathsf{c}''d+o(d)}$ respectively, for $\mathsf{c}, \mathsf{c}', \mathsf{c}'' \in \Theta(1)$, which we aim to minimise. We do not attempt to specify the $o(d)$ or poly(d) terms.

[2] This means that the complexity of the algorithm is measured by the number of oracle calls to the adjacency matrix of a graph.

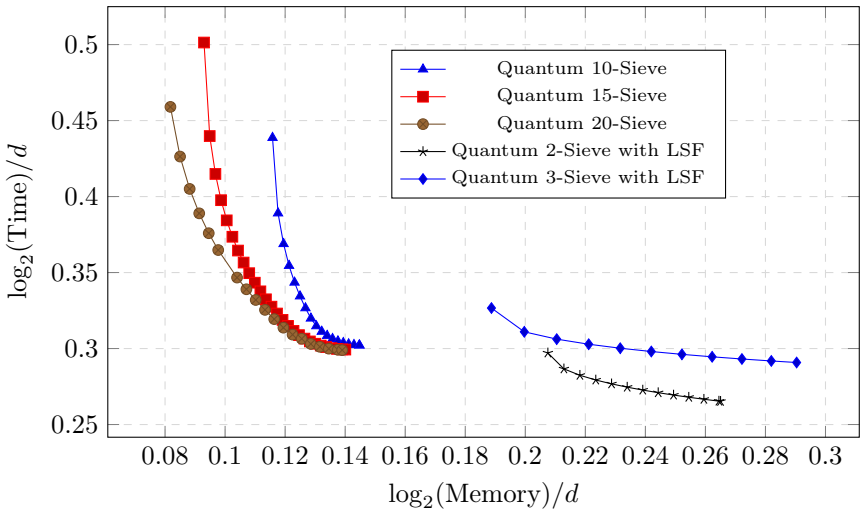


Figure 1: Time-memory trade-offs for Algorithm 2 with $k \in \{10, 15, 20\}$ and Algorithm 7 with $k \in \{2, 3\}$. Each curve provides time-memory trade-offs for a fixed k, either with nearest neighbour techniques (the right two curves) or without (the left three curves). Each point on a curve (x, y) represents (Memory, Time) values, obtained by numerically optimising for time while fixing available memory. For example, we build the leftmost curve (dotted brown) by computing the memory optimal (Memory, Time) value for the 20-Sieve and then repeatedly increase the available memory (decreasing time) until we reach the time optimal (Memory, Time) value. Increasing memory further than the rightmost point on each curve does not decrease time. The figures were obtained using optimisation package provided by Maple$^{\text{TM}}$ [Map].

Our techniques. We now briefly describe the main ingredients of our results.

1. A useful abstraction of the k-Sieve is the *configuration problem*, first described in [HK17]. It consists of finding k elements that satisfy certain pairwise inner product constraints from k exponentially large lists of vectors. Assuming $(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ is a solution tuple, the i^{th} element $\mathbf{x}_i$ can be obtained via a brute force search either over the i^{th} input list [BLS16], or over a certain sublist of the i^{th} list [HK17], see Figure 2b. We replace the brute force searches with calls to Grover's algorithm and reanalyse the configuration problem.

2. An alternative way to find the i^{th} element of a solution tuple for the configuration problem is to apply nearest neighbour techniques [Laa15, BDGL16]. This method sorts the i^{th} list into a specially crafted data structure that, for a given $(\mathbf{x}_1, \ldots, \mathbf{x}_{i-1})$, allows one to find the satisfying $\mathbf{x}_i$ faster than via brute force. The search for $\mathbf{x}_i$ within such a data structure can itself be sped up by Grover's algorithm.

3. The configuration problem can be reduced to the k-clique problem in a graph with vertices representing elements from the lists given by the configuration problem. Vertices are connected by an edge if and only if the

corresponding list elements satisfy some inner product constraint. Classically, this interpretation yields no improvements to configuration problem algorithms. However we achieve quantum speed-ups by generalising the triangle finding algorithm of Buhrman et al. [BdWD+01] and applying it to k-cliques.

4. We apply the triangle finding algorithm of Le Gall–Nakajima [LGN17] and exploit the structure of our graph instance. In particular we form many graphs from unions of sublists of our lists, allowing us to alter the sparsity of said graphs.

5. To make use of more quantum memory we run Grover searches in parallel. The idea is to allow simultaneous queries by several processors to a large, shared, quantum memory. Instead of looking for a "good" $\mathbf{x}_i$ for one *fixed* tuple $(\mathbf{x}_1, \ldots, \mathbf{x}_{i-1})$, one could think of parallel searches aiming to find a "good" $\mathbf{x}_i$ for several tuples $(\mathbf{x}_1, \ldots, \mathbf{x}_{i-1})$. The possibility of running several Grover's algorithms concurrently was shown in the work of Beals et al. [BBG+13]. Based on this result we specify all the subroutines needed to solve the shortest vector problem using large quantum memory.

Open questions.

1. The classical configuration search algorithms of [HKL18] offer time-memory trade-offs for SVP by varying k (larger k requires less memory but more time). We observe in Section 3 that time optimal classical algorithms for the configuration problem hit a certain point on the time-memory trade-off curve once k becomes large enough, see Table 1. The same behaviour is observed for our quantum algorithms for the configuration problem, see Table 2. Although we provide some explanation of this, we do not rigorously prove that the trade-off curve indeed stops on its time optimal side. We leave it as an open problem to determine the shape of the configuration problem for these time optimal instances of the algorithm. Another open problem originating from [HK17] is to extend the analysis to non constant k.

2. We do not give time complexities in Section 6, instead reporting the query complexity for listing triangles. We leave open the question of determining, e.g. the complexity of forming auxiliary databases used by the quantum random walks on Johnson graphs of [LGN17], which is not captured in the query model, as well as giving the (quantum) memory requirements of these methods in our setting. If the asymptotic time complexity does not increase (much) above the query complexity then the $2^{0.3264d+o(d)}$ achieved by the algorithm in Section 6 represents both an improvement over the best classical algorithms for the relevant configuration problem [HKL18] and an improvement over Algorithms 2, 3 for $k = 3$ in low memory regimes, see Table 2.

3. In Section 7 we present a parallel quantum version of a 2-Sieve. We believe that it should be possible to extend the techniques to k-Sieve for $k > 2$.

2 Preliminaries

We denote by $\mathsf{S}^d \subset \mathbb{R}^{d+1}$ the d-dimensional unit sphere. We use soft-$\mathcal{O}$ notation to denote running times, that is $T = \widetilde{\mathcal{O}}(2^{cd})$ suppresses subexponential factors in d. By $[n]$ we denote the set $\{1, \ldots, n\}$. The norm considered in this work is Euclidean and is denoted by $\|\cdot\|$.

For any set $\mathbf{x}_1, \ldots, \mathbf{x}_k$ of vectors in $\mathbb{R}^d$, the *Gram matrix* $C \in \mathbb{R}^{k \times k}$ is given by $C_{i,j} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$, the set of pairwise scalar products. For $I \subset [k]$, we denote by $C[I]$ the $|I| \times |I|$ submatrix of C obtained by restricting C to the rows and columns indexed by I. For a vector $\mathbf{x}$ and $i \in [k]$, $\mathbf{x}[i]$ denotes the i^{th} entry. For a function f, by O_f we denote a unitary matrix that implements f.

Lattices. Given a basis $B = \{\mathbf{b}_1, \ldots, \mathbf{b}_m\} \subset \mathbb{R}^d$ of linearly independent vectors $\mathbf{b}_i$, the lattice generated by B is defined as $\mathcal{L}(B) = \{\sum_{i=1}^m z_i \mathbf{b}_i \colon z_i \in \mathbb{Z}\}$. For simplicity we work with lattices of full rank ($d = m$). The Shortest Vector Problem (SVP) is to find, for a given B, a shortest non zero vector of $\mathcal{L}(B)$. Minkowski's theorem for the Euclidean norm states that a shortest vector of $\mathcal{L}(B)$ is bounded from above by $\sqrt{d} \cdot \det(B)^{1/d}$.

Quantum Search. Our results rely on Grover's quantum search algorithm [Gro96] which finds "good" elements in a (large) list. The analysis of the success probability of this algorithm can be found in [BBHT98]. We also rely on the generalisation of Grover's algorithm, called Amplitude Amplification, due to Brassard–Høyer–Mosca–Tapp [BHMT02] and a result on parallel quantum search [BBG+13].

Theorem 2.1 (Grover's algorithm [Gro96,BBHT98])**.** *Given quantum access to a list L that contains t marked elements (the value t is not necessarily known) and a function $f \colon L \to \{0,1\}$, described by a unitary O_f, which determines whether an element is "good" or not, we wish to find a solution $i \in [|L|]$, such that for $f(x_i) = 1, x_i \in L$. There exists a quantum algorithm, called Grover's algorithm, that with probability greater than $1 - t/|L|$ outputs one "good" element using $\mathcal{O}(\sqrt{|L|/t})$ calls to O_f.*

Theorem 2.2 (Amplitude Amplification [BHMT02, Theorem 2])**.** *Let $\mathcal{A}$ be any quantum algorithm that makes no measurements and let $\mathcal{A}|0\rangle = |\Psi_0\rangle + |\Psi_1\rangle$, where $|\Psi_0\rangle$ and $|\Psi_1\rangle$ are spanned by "bad" and "good" states respectively. Let further $a = \langle \Psi_1 | \Psi_1 \rangle$ be the success probability of $\mathcal{A}$. Given access to a function f that flips the sign of the amplitudes of good states, i.e. $f \colon |x\rangle \mapsto -|x\rangle$ for "good" $|x\rangle$ and leaves the amplitudes of "bad" $|x\rangle$ unchanged, the amplitude amplification algorithm constructs the unitary $Q = -\mathcal{A}R\mathcal{A}^{-1}O_f$, where R is the reflection about $|0\rangle$, and applies Q^m to the state $\mathcal{A}|0\rangle$, where $m = \lfloor \frac{\pi}{4} \arcsin(\sqrt{a}) \rfloor$. Upon measurement of the system, a "good" state is obtained with probability at least $\max\{a, 1-a\}$.*

Theorem 2.3 (Quantum Parallel Search [BBG+13])**.** *Given a list L, with each element of bit length d, and $|L|$ functions that take list elements as input $f_i \colon L \to \{0,1\}$ for $i \in [|L|]$, we wish to find solution vectors $\mathbf{s} \in [|L|]^{|L|}$. A solution has $f_i(\mathbf{x}_{\mathbf{s}[i]}) = 1$ for all $i \in [|L|]$. Given unitaries $U_{f_i} \colon |\mathbf{x}\rangle|b\rangle \to |\mathbf{x}\rangle|b \oplus f_i(\mathbf{x})\rangle$ there exists a quantum algorithm that, for each $i \in [|L|]$, either returns a solution $\mathbf{s}[i]$ or if there is no such solution, returns* no solution*. The algorithm succeeds with*

probability $\Theta(1)$ *and, given that each* U_{f_i} *has depth and width* $\operatorname{poly}\log(|L|, d)$*, can be implemented using a quantum circuit of width* $\widetilde{\mathcal{O}}(|L|)$ *and depth* $\widetilde{\mathcal{O}}(\sqrt{|L|})$*.*

Computational Models. Our algorithms are analysed in the quantum circuit model [KLM07]. We set each wire to represent a qubit, i.e. a vector in a two dimensional complex Hilbert space, and assert that we have a set of universal gates. We work in the noiseless quantum theory, i.e. we assume there is no (or negligible) decoherence or other sources of noise in the computational procedures.

The algorithms given in Sections 4 and 5 are in the QRAM model and assume quantumly accessible classical memory [GLM08]. More concretely in this model we store all data, e.g. the list of vectors, in classical memory and only demand that this memory is quantumly accessible, i.e. elements in the list can be efficiently accessed in coherent superposition. This enables us to design algorithms that, in principle, do not require large quantum memories and can be implemented with only $\operatorname{poly}(d)$ qubits and with the $2^{\Theta(d)}$ sized list stored in classical memory. Several works [BHT97, Kup13] suggest that this memory model is potentially easier to achieve than a full quantum memory.

In Section 6 we study the algorithms in the query model, which is the typical model for quantum triangle or k-clique finding algorithms. Namely, the complexity of our algorithm is measured in the number of oracle calls to the adjacency matrix of a graph associated to a list of vectors.

Acknowledging the arguments against the feasibility of QRAM and whether it can be meaningfully cheaper than quantum memory [AGJO$^+$15], in Section 7 we consider algorithms that use exponential quantum memory in the quantum circuit model without assuming QRAM.

3 Sieving as Configuration Search

In this section we describe previously known *classical* sieving algorithms. We will not go into detail or give proofs, which can be found in the relevant references.

Sieving algorithms receive on input a basis $B \in \mathbb{R}^{d\times d}$ and start by sampling an exponentially large list L of (long) lattice vectors from $\mathcal{L}(B)$. There are efficient algorithms for sampling lattice vectors, e.g. [Kle00]. The elements of L are then iteratively combined to form shorter lattice vectors, $\mathbf{x}_{\text{new}} = \mathbf{x}_1 \pm \mathbf{x}_2 \pm \ldots \pm \mathbf{x}_k$ such that $\|\mathbf{x}_{\text{new}}\| \leq \max_{i\leq k}\{\|\mathbf{x}_i\|\}$, for some $k \geq 2$. Newly obtained vectors $\mathbf{x}_{\text{new}}$ are stored in a new list and the process is repeated with this new list of shorter vectors. It can be shown [NV08, Reg09] that after $\operatorname{poly}(d)$ such iterations we obtain a list that contains a shortest vector. Therefore, the asymptotic complexity of sieving is determined by the cost of finding k-tuples whose combination produces shorter vectors. Under certain heuristics, specified below, finding such k-tuples can be formulated as the approximate k-List problem.

Definition 3.1 (Approximate k-List problem)**.** *Assume we are given k lists $L_1, \ldots, L_k$ of equal exponential (in d) size $|L|$ and whose elements are i.i.d. uniformly chosen vectors from S^{d-1}. The approximate k-List problem is to find $|L|$*

solutions, where a solution is a k-tuple $(x_1, \ldots, x_k) \in L_1 \times \ldots \times L_k$ *satisfying* $\|\mathbf{x}_1 + \ldots + \mathbf{x}_k\| \leq 1$.

The assumption made in analyses of heuristic sieving algorithms [NV08] is that the lattice vectors in the new list after an iteration are thought of as i.i.d. uniform vectors on a thin spherical shell (essentially, a sphere), and, once normalised, on S^{d-1}. Hence sieves do not "see" the discrete structure of the lattice from the vectors operated on. The heuristic becomes invalid when the vectors become short. In this case we assume we have solved SVP. Thus, we may not find a *shortest* vector, but an approximation to it, which is enough for most cryptanalytic purposes.

We consider k to be constant. The lists $L_1, \ldots, L_k$ in Definition 3.1 may be identical. The algorithms described below are applicable to this case as well. Furthermore, the approximate k-List problem only looks for solutions with $+$ signs, i.e. $\|\mathbf{x}_1 + \ldots + \mathbf{x}_k\| \leq 1$, while sieving looks for arbitrary signs. This is not an issue, as we may repeat an algorithm for the approximate k-List problem $2^k = \mathcal{O}(1)$ times in order to obtain all solutions.

Configuration Search. Using a concentration result on the distribution of scalar products of $\mathbf{x}_1, \ldots, \mathbf{x}_k \in \mathsf{S}^{d-1}$ shown in [HK17], the approximate k-List problem can be reduced to the configuration problem. In order to state this problem, we need a notion of configurations.

Definition 3.2 (Configuration)**.** *The configuration* $C = \mathrm{Conf}(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ *of* k *points* $\mathbf{x}_1, \ldots, \mathbf{x}_k \in \mathsf{S}^{d-1}$ *is the Gram matrix of the* $\mathbf{x}_i$*, i.e.* $C_{i,j} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$.

A configuration $C \in \mathbb{R}^{k \times k}$ is a positive semidefinite matrix. Rewriting the solution condition $\|\mathbf{x}_1 + \ldots + \mathbf{x}_k\|^2 \leq 1$, one can check that a configuration C for a solution tuple satisfies $\mathbf{1}^T C \mathbf{1} \leq 1$. We denote the set of such "good" configurations by

$$\mathscr{C} = \{C \in \mathbb{R}^{k \times k} \colon C \text{ is positive semidefinite and } \mathbf{1}^T C \mathbf{1} \leq 1\}.$$

It has been shown [HK17] that rather than looking for k-tuples that form a solution for the approximate k-List problem, we may look for k-tuples that satisfy a constraint on their configuration. It gives rise to the following problem.

Definition 3.3 (Configuration problem)**.** *Let* $k \in \mathbb{N}$ *and* $\varepsilon > 0$*. Suppose we are given a target configuration* $C \in \mathscr{C}$*. Given* k *lists* $L_1, \ldots, L_k$ *all of exponential (in* d*) size* $|L|$*, whose elements are i.i.d. uniform from* S^{d-1}*, the configuration problem consists of finding a* $1 - o(1)$ *fraction of all solutions, where a solution is a* k*-tuple* $(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ *with* $\mathbf{x}_i \in L_i$ *such that* $|\langle \mathbf{x}_i, \mathbf{x}_j \rangle - C_{i,j}| \leq \varepsilon$ *for all* i, j.

Solving the configuration problem for a $C \in \mathscr{C}$ gives solutions to the approximate k-List problem. For a given $C \in \mathbb{R}^{k \times k}$ the number of expected solutions to the configuration problem is given by $\det(C)$ as the following theorem shows.

Theorem 3.1 (Distribution of configurations [HK17, Theorem 1])**.** *If* $\mathbf{x}_1, \ldots, \mathbf{x}_k$ *are i.i.d. from* S^{d-1}*,* $d > k$*, then their configuration* $C = \mathrm{Conf}(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ *follows a distribution with density function*

$$\mu = W_{d,k} \cdot \det(C)^{\frac{1}{2}(d-k)} \mathrm{d}C_{1,2} \ldots \mathrm{d}C_{d-1,d}, \tag{1}$$

where $W_{d,k} = \mathcal{O}_k(d^{\frac{1}{4}(k^2-k)})$ *is an explicitly known normalisation constant that only depends on* d *and* k.

This theorem tells us that the expected number of solutions to the configuration problem for C is given by $\prod_i |L_i| \cdot (\det C)^{d/2}$. If we want to apply an algorithm for the configuration problem to the approximate k-List problem (and to sieving), we require that the expected number of output solutions to the configuration problem is equal to the size of the input lists. Namely, C and the input lists L_i of size $|L|$ should (up to polynomial factors) satisfy $|L|^k \cdot (\det C)^{d/2} = |L|$. This condition gives a lower bound on the size of the input lists. Using Chernoff bounds, one can show (see [HKL18, Lemma 2]) that increasing this bound by a poly(d) factor gives a sufficient condition for the size of input lists, namely

$$|L| = \widetilde{\mathcal{O}}\left(\left(\frac{1}{\det(C)}\right)^{\frac{d}{2(k-1)}}\right). \tag{2}$$

Classical algorithms for the configuration problem. The first classical algorithm for the configuration problem for $k \geq 2$ was given by Bai–Laarhoven–Stehlé [BLS16]. It is depicted in Figure 2a. It was later improved by Herold–Kirshanova [HK17] and by Herold–Kirshanova–Laarhoven [HKL18] (Figure 2b). These results present a family of algorithms for the configuration problem that offer time-memory trade-offs. In Section 4 we present quantum versions of these algorithms.

Both algorithms [BLS16, HKL18] process the lists from left to right but in a different manner. For each $\mathbf{x}_1 \in L_1$ the algorithm from [BLS16] applies a filtering procedure to L_2 and creates the "filtered" list $L_2(\mathbf{x}_1)$. This filtering procedure takes as input an element $\mathbf{x}_2 \in L_2$ and adds it to $L_2(\mathbf{x}_1)$ iff $|\langle \mathbf{x}_1 , \mathbf{x}_2 \rangle - C_{1,2}| \leq \varepsilon$. Having constructed the list $L_2(\mathbf{x}_1)$, the algorithm then iterates over it: for each $\mathbf{x}_2 \in L_2(\mathbf{x}_1)$ it applies the filtering procedure to L_3 with respect to $C_{2,3}$ and obtains $L_3(\mathbf{x}_1, \mathbf{x}_2)$. Throughout, vectors in brackets indicate fixed elements with respect to which the list has been filtered. Filtering of the top level lists $(L_1, \ldots, L_k)$ continues in this fashion until we have constructed $L_k(\mathbf{x}_1, \ldots, \mathbf{x}_{k-1})$ for fixed values $\mathbf{x}_1, \ldots, \mathbf{x}_{k-1}$. The tuples of the form $(\mathbf{x}_1, \ldots, \mathbf{x}_{k-1}, \mathbf{x}_k)$ for all $\mathbf{x}_k \in L_k(\mathbf{x}_1, \ldots, \mathbf{x}_{k-1})$ form solutions to the configuration problem.

The algorithms from [HK17, HKL18] apply more filtering steps. For a fixed $\mathbf{x}_1 \in L_1$, they not only create $L_2(\mathbf{x}_1)$, but also $L_3(\mathbf{x}_1), \ldots, L_k(\mathbf{x}_1)$. This speeds up the next iteration over all $\mathbf{x}_2 \in L_2(\mathbf{x}_1)$, where now the filtering step with respect to $C_{2,3}$ is applied not to L_3, but to $L_3(\mathbf{x}_1)$, as well as to $L_4(\mathbf{x}_1), \ldots, L_k(\mathbf{x}_1)$, each of which is smaller than L_i. This speeds up the construction of $L_3(\mathbf{x}_1, \mathbf{x}_2)$. The algorithm continues with this filtering process until the last inner product check with respect to $C_{k-1,k}$ is applied to all the elements from $L_k(\mathbf{x}_1, \ldots, \mathbf{x}_{k-2})$ and the list $L_k(\mathbf{x}_1, \ldots, \mathbf{x}_{k-1})$ is constructed. This gives solutions of the form $(\mathbf{x}_1, \ldots, \mathbf{x}_{k-1}, \mathbf{x}_k)$ for all $\mathbf{x}_k \in L_k(\mathbf{x}_1, \ldots, \mathbf{x}_{k-1})$. The concentration result, Theorem 3.1, implies the outputs of algorithms from [BLS16] and [HK17, HKL18] are (up to a subexponential fraction) the same. Pseudocode for [HK17] can be found in Appendix A.

Important for our analysis in Section 4 will be the the result of [HKL18] that describes the sizes of all the intermediate lists that appear during the configuration search algorithms via the determinants of submatrices of the target configuration C. The next theorem gives the expected sizes of these lists and

the time complexity of the algorithm from [HKL18].

Theorem 3.2 (Intermediate list sizes [HKL18, Lemma 1] and time compleixty of configuration search algorithm)**.** *During a run of the configuration search algorithms described in Figures 2a, 2b, given an input configuration* $C \in \mathbb{R}^{k\times k}$ *and lists* $L_1, \ldots, L_k \subset \mathsf{S}^{d-1}$ *each of size* $|L|$*, the intermediate lists for* $1 \leq i < j \leq k$ *are of expected sizes*

$$\mathbb{E}[|L_j(\mathbf{x}_1, \ldots, \mathbf{x}_i)|] = |L| \cdot \left(\frac{\det(C[1, \ldots, i, j])}{\det(C[1 \ldots i])}\right)^{d/2}. \tag{3}$$

The expected running time of the algorithm described in Figure 2b is

$$T^{\mathrm{C}}_{\text{k-Conf}} = \max_{1\leq i\leq k}\left[\prod_{r=1}^{i} |L_r(\mathbf{x}_1, \ldots, \mathbf{x}_{r-1})| \cdot \max_{i+1\leq j\leq k} |L_j(\mathbf{x}_1, \ldots, \mathbf{x}_{i-1})|\right]. \tag{4}$$

Finding a configuration for optimal runtime. For a given i the square bracketed term in Eq. (4) represents the expected time required to create all filtered lists on a given "level". Here "level" refers to all lists filtered with respect to the same fixed $\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}$, i.e. a row of lists in Figure 2b. In order to find an optimal configuration C that minimises Eq. (4), we perform numerical optimisations using the Maple$^{\mathrm{TM}}$ package [Map].[3] In particular, we search for $C \in \mathscr{C}$ that minimises Eq. (4) under the condition that Eq. (2) is satisfied (so that we actually obtain enough solutions for the k-List problem). Figures for the optimal runtime and the corresponding memory are given in Table 1. The memory is determined by the size of the input lists computed from the optimal C using Eq. (2). Since the k-List routine determines the asymptotic cost of k-Sieve, the figures in Table 1 are also the constants in the exponents for complexities of k-Sieves.

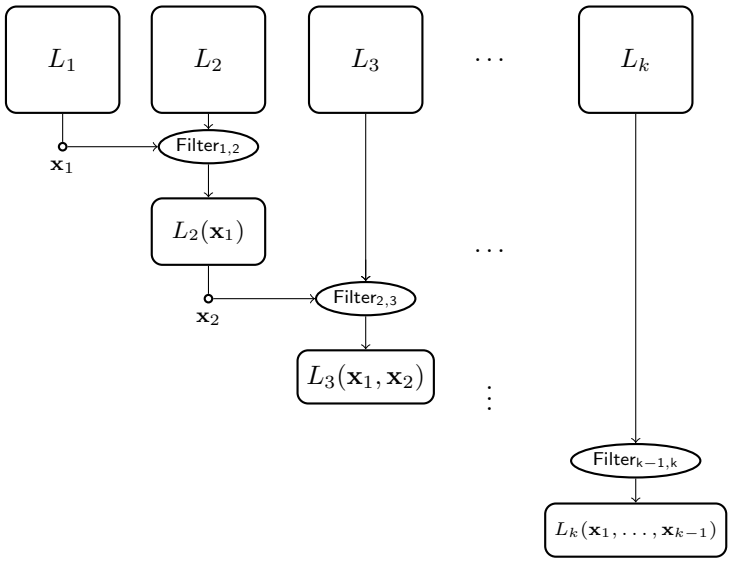
k	2	3	4	5	6	...	16	17	18
Time	0.4150	0.3789	0.3702	0.3707	0.3716		0.3728	0.37281	0.37281
Space	0.2075	0.1895	0.1851	0.1853	0.1858		0.1864	0.18640	0.18640

Table 1: Asymptotic complexity exponents for the approximate k-List problem, base 2. The table gives optimised runtime and the corresponding memory exponents for the classical algorithm from [HKL18], see Figure 2b and Algorithm 5.

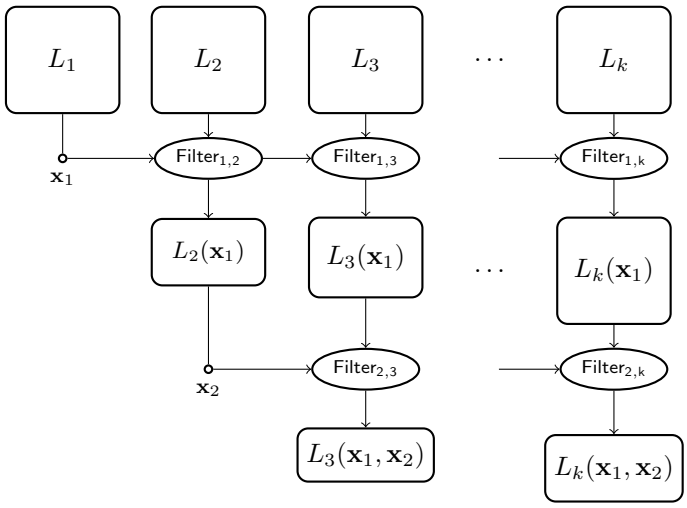
Interestingly, the optimal runtime constant turns out to be equal for large enough k. This can be explained as follows. The optimal C achieves the situation where all the expressions in the outer max in Eq. (4) are equal. This implies that creating all the filtered lists on level i asymptotically costs the same as creating all the filtered lists on level $i+1$ for $2 \leq i \leq k-1$. The cost of creating filtered lists $L_i(\mathbf{x}_1)$ on the second level (assuming that the first level consists of the input lists) is of order $|L|^2$. This value, $|L|^2$, becomes (up to poly(d) factors) the running time of the whole algorithm (compare the Time and Space constants for $k = 16, 17, 18$ in Table 1). The precise shape of $C \in \mathscr{C}$

[3] The code is available at https://github.com/ElenaKirshanova/QuantumSieve

Figure 2: Algorithms for the configuration problem. Procedures $\mathsf{Filter}_{i,j}$ receive as input a vector (e.g. $\mathbf{x}_1$), a list of vectors (e.g. L_2), and a real number $C_{i,j}$, the target inner product. It creates another shorter list (e.g. $L_2(\mathbf{x}_1)$) that contains all vectors from the input list whose inner product with the input vector is within some small ε from the target inner product.



(a) The algorithm of Bai et al. [BLS16] for the configuration problem.



(b) The algorithm of Herold et al. [HKL18] for the configuration problem.

that balances the costs per level can be obtained by equating all the terms in the max of Eq. (4) and minimising the value $|L|^2$ under these constraints. Even for small k these computations become rather tedious and we do not attempt to express $C_{i,j}$ as a function of k, which is, in principal, possible.

Finding a configuration for optimal memory. If we want to optimise for memory, the optimal configuration C has all its off diagonal elements $C_{i,j} = -1/k$. It is shown in [HK17] that such C maximises $\det(C)$ among all $C \in \mathscr{C}$, which, in turn, minimises the sizes of the input lists (but does not lead to optimal running time as the costs per level are not balanced).

4 Quantum Configuration Search

In this section we present several quantum algorithms for the configuration problem (Definition 3.3). As explained in Section 3, this directly translates to quantum sieving algorithms for SVP. We start with a quantum version of the BLS style configuration search [BLS16], then we show how to improve this algorithm by constructing intermediate lists. In Appendix B we show how nearest neighbour methods in the quantum setting speed up the latter algorithm.

Recall the configuration problem: as input we receive k lists $L_i, i \in [k]$ each of size a power of two,[4] a configuration matrix $C \in \mathbb{R}^{k\times k}$ and $\varepsilon \geq 0$. To describe our first algorithm we denote by $f_{[i],j}$ a function that takes as input $(i+1)$ many d-dimensional vectors and is defined as

$$f_{[i],j}(\mathbf{x}_1, \ldots, \mathbf{x}_i, \mathbf{x}) = \begin{cases} 1, & |\langle \mathbf{x}_\ell \, , \mathbf{x}\rangle - C_{\ell,j}| \leq \varepsilon, \quad \ell \in [i] \\ 0, & \text{else.} \end{cases}$$

A reversible embedding of $f_{[i],j}$ is denoted by $O_{f_{[i],j}}$. Using these functions we perform a check for "good" elements and construct the lists $L_j(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_i)$. Furthermore, we assume that any vector encountered by the algorithm fits into $\bar{d}$ qubits. We denote by $|\mathbf{0}\rangle$ the $\bar{d}$-tensor of 0 qubits, i.e. $|\mathbf{0}\rangle = |0^{\otimes \bar{d}}\rangle$.

The input lists, $L_i, i \in [k]$, are stored classically and are assumed to be quantumly accessible. In particular, we assume that we can efficiently construct a uniform superposition over all elements from a given list by first applying Hadamards to $|\mathbf{0}\rangle$ to create a superposition over all indices, and then by querying $L[i]$ for each i in the superposition. That is, we assume an efficient circuit for $\frac{1}{\sqrt{|L|}} \sum_i |i\rangle |\mathbf{0}\rangle \to \frac{1}{\sqrt{|L|}} \sum_i |i\rangle |L[i]\rangle$. For simplicity, we ignore the first qubit that stores indices and we denote by $|\Psi_L\rangle$ a uniform superposition over all the elements in L, i.e. $|\Psi_L\rangle = \frac{1}{\sqrt{|L|}} \sum_{\mathbf{x}\in L} |\mathbf{x}\rangle$.

The idea of our algorithm for the configuration problem is the following. We have a global *classical* loop over $\mathbf{x}_1 \in L_1$ inside which we run our quantum algorithm to find a $(k-1)$ tuple $(\mathbf{x}_2, \ldots, \mathbf{x}_k)$ that together with $\mathbf{x}_1$ gives a solution to the configuration problem. We expect to have $\mathcal{O}(1)$ such $(k-1)$ tuples per $\mathbf{x}_1$.[5] At the end of the algorithm we expect to obtain such a solution by

[4] This is not necessary but it enables us to efficiently create superpositions $|\Psi_{L_i}\rangle$ using Hadamard gates. Since our lists L_i are of sizes $2^{\mathsf{c}d+o(d)}$ for a large d and a constant $\mathsf{c} < 1$, this condition is easy to satisfy by rounding $\mathsf{c}d$.

[5] This follows by multiplying the sizes of the lists $L_i(\mathbf{x}_1, \ldots \mathbf{x}_{i-1})$ for all $2 \leq i \leq k$.

means of amplitude amplification (Theorem 2.2). In Theorem 4.1 we argue that this procedure succeeds in finding a solution with probability at least $1-2^{-\Omega(d)}$.

Inside the classical loop over $\mathbf{x}_1$ we prepare $(k-1)\bar{d}$ qubits, which we arrange into $k-1$ registers, so that each register will store (a superposition of) input vectors, see Figure 3. Each such register is set in uniform superposition over the elements of the input lists: $|\Psi_{L_2}\rangle \otimes |\Psi_{L_3}\rangle \otimes \cdots \otimes |\Psi_{L_k}\rangle$. We apply Grover's algorithm on $|\Psi_{L_2}\rangle$. Each Grover's iteration is defined by the unitary $Q_{1,2} = -H^{\otimes \bar{d}} R H^{\otimes \bar{d}} O_{f_{[1],2}}$. Here H is the Hadamard gate and R is the rotation around $|\mathbf{0}\rangle$. We have $|L_2(\mathbf{x}_1)|$ "good" states out of $|L_2|$ possible states in $|\Psi_{L_2}\rangle$, so after $\mathcal{O}\left(\sqrt{\frac{|L_2|}{|L_2(\mathbf{x}_1)|}}\right)$ applications of $Q_{1,2}$ we obtain the state

$$|\Psi_{L_2(\mathbf{x}_1)}\rangle = \frac{1}{\sqrt{|L_2(\mathbf{x}_1)|}} \sum_{\mathbf{x}_2 \in L_2(\mathbf{x}_1)} |\mathbf{x}_2\rangle \,. \tag{5}$$

In fact, what we create is a state close to Eq. (5) as we do not perform any measurement. For now, we drop the expression "close to" for all the states in this description,

Now consider the state $|\Psi_{L_2(\mathbf{x}_1)}\rangle \otimes |\Psi_{L_3}\rangle$ and the function $f_{[2],3}$ that uses the first and second registers and a fixed $\mathbf{x}_1$ as inputs. We apply the unitary $Q_{2,3}$ to $|\Psi_{L_3}\rangle$, where $Q_{2,3} = -H^{\otimes \bar{d}} R H^{\otimes \bar{d}} O_{f_{[2],3}}$. In other words, for all vectors from L_3, we check if they satisfy the inner product constraints with respect to $\mathbf{x}_1$ and $\mathbf{x}_2$. In this setting there are $|L_3(\mathbf{x}_1, \mathbf{x}_2)|$ "good" states in $|\Psi_{L_3}\rangle$ whose amplitudes we aim to amplify. Applying Grover's iteration unitary $Q_{2,3}$ the order of $\mathcal{O}\left(\sqrt{\frac{|L_3|}{|L_3(\mathbf{x}_1,\mathbf{x}_2)|}}\right)$ times, we obtain the state

$$|\Psi_{L_2(\mathbf{x}_1)}\rangle\, |\Psi_{L_3(\mathbf{x}_1,\mathbf{x}_2)}\rangle = \frac{1}{\sqrt{|L_2(\mathbf{x}_1)|}} \sum_{\mathbf{x}_2 \in L_2(\mathbf{x}_1)} |\mathbf{x}_2\rangle \left(\frac{1}{\sqrt{|L_3(\mathbf{x}_1, \mathbf{x}_2)|}} \sum_{\mathbf{x}_3 \in L_3(\mathbf{x}_1,\mathbf{x}_2)} |\mathbf{x}_3\rangle \right).$$

We continue creating the lists $L_{i+1}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_i)$ by filtering the *initial* list L_{i+1} with respect to $\mathbf{x}_1$ (fixed by the outer classical loop), and with respect to $\mathbf{x}_2, \ldots, \mathbf{x}_i$ (given in a superposition) using the function $f_{[i],i+1}$. At level $k-1$ we obtain the state $|\Psi_{L_2(\mathbf{x}_1)}\rangle \otimes |\Psi_{L_3(\mathbf{x}_1,\mathbf{x}_2)}\rangle \otimes \ldots \otimes |\Psi_{L_{k-1}(\mathbf{x}_1,\ldots,\mathbf{x}_{k-2})}\rangle$. For the last list L_k we filter with respect to $\mathbf{x}_1, \ldots, \mathbf{x}_{k-2}$ as for the list L_{k-1}. Finally, for a fixed $\mathbf{x}_1$, the "filtered" state we obtained is of the form

$$|\Psi_F\rangle = |\Psi_{L_2(\mathbf{x}_1)}\rangle \otimes |\Psi_{L_3(\mathbf{x}_1,\mathbf{x}_2)}\rangle \otimes \ldots \otimes |\Psi_{L_{k-1}(\mathbf{x}_1,\ldots,\mathbf{x}_{k-2})}\rangle \otimes |\Psi_{L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{k-2})}\rangle \,. \tag{6}$$

The state is expected to contain $\mathcal{O}(1)$ many $(k-1)$-tuples $(\mathbf{x}_2, \ldots, \mathbf{x}_k)$ which together with $\mathbf{x}_1$ give a solution to the configuration problem. To prepare the state $|\Psi_F\rangle$ for a fixed $\mathbf{x}_1$, we need

$$T_{\text{InGrover}} = \mathcal{O}\left(\sqrt{\left(\frac{|L_2|}{|L_2(\mathbf{x}_1)|}\right)} + \ldots + \sqrt{\left(\frac{|L_k|}{|L_k(\mathbf{x}_1, \ldots, \mathbf{x}_{k-2})|}\right)} \right) \tag{7}$$

unitary operations of the form $(-H^{\otimes \bar{d}}) R H^{\otimes \bar{d}} O_{f_{[i],j}}$. This is what we call the "inner" Grover procedure.

Let us denote by $\mathcal{A}$ an algorithm that creates $|\Psi_F\rangle$ from $|\mathbf{0}\rangle \otimes \ldots \otimes |\mathbf{0}\rangle$ in time T_{InGrover}. In order to obtain a solution tuple $(\mathbf{x}_2, \ldots, \mathbf{x}_k)$ we apply amplitude

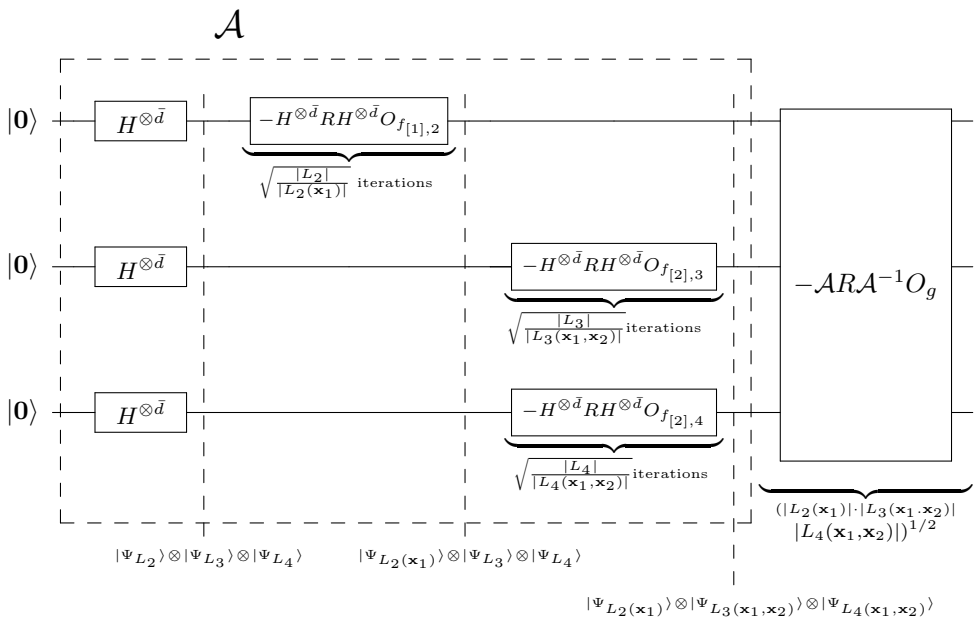


Figure 3: Quantum circuit representing the quantum part of Algorithm 2 with $k = 4$, i.e. this circuit is executed inside the loop over $\mathbf{x}_1 \in L_1$. The Hadamard gates create the superposition $|\Psi_{L_2}\rangle \otimes |\Psi_{L_3}\rangle \otimes |\Psi_{L_4}\rangle$. We apply $\sqrt{\frac{|L_2|}{|L_2(\mathbf{x}_1)|}}$ Grover iterations to $|\Psi_{L_2}\rangle$ to obtain the state $|\Psi_{L_2(\mathbf{x}_2)}(\mathbf{x}_1)\rangle \otimes |\Psi_{L_3}\rangle \otimes |\Psi_{L_4}\rangle$. We then apply (sequentially) $\mathcal{O}\left(\sqrt{\frac{|L_3|}{|L_3(\mathbf{x}_1,\mathbf{x}_2)|}}\right)$ *resp.* $\mathcal{O}\left(\sqrt{\frac{|L_4|}{|L_4(\mathbf{x}_1,\mathbf{x}_2)|}}\right)$ Grover iterations to the second *resp.* third registers, where the checking function takes as input the first and second *resp.* the first and third registers. This whole process is $\mathcal{A}$ and is repeated $\mathcal{O}(\sqrt{|L_2(\mathbf{x}_1)| \cdot |L_3(\mathbf{x}_1,\mathbf{x}_2)|\,|L_4(\mathbf{x}_1,\mathbf{x}_2)|})$ times inside the amplitude amplification. Final measurement gives a triple $(\mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)$ which, together with a fixed $\mathbf{x}_1$, forms a solution to the configuration problem.

amplification using the unitary $Q_{\text{Outer}} = -\mathcal{A}R\mathcal{A}^{-1}O_g$, where g is the function that operates on the last two registers and is defined as

$$g(\mathbf{x}, \mathbf{x}') = \begin{cases} 1, & |\langle \mathbf{x}\,, \mathbf{x}' \rangle - C_{k-1,k}| \leq \varepsilon \\ 0, & \text{else.} \end{cases} \tag{8}$$

Notice that in the state $|\Psi_F\rangle$ it is only the last two registers storing $\mathbf{x}_{k-1}$ and $\mathbf{x}_k$ that are left to be checked against the target configuration. This is precisely what we use O_g to check. Let $|\mathbf{z}\rangle = |\mathbf{x}_2, \ldots, \mathbf{x}_k\rangle$ be a solution tuple. The state $|\mathbf{z}\rangle$ appears in $|\Psi_F\rangle$ with amplitude

$$\langle \mathbf{z}|\Psi_F\rangle = \mathcal{O}\left((\sqrt{|L_2(\mathbf{x}_1)| \cdot \ldots \cdot |L_{k-1}(\mathbf{x}_1, \ldots, \mathbf{x}_{k-2})| \cdot |L_k(\mathbf{x}_1, \ldots, \mathbf{x}_{k-2})|})^{-1} \right).$$

This value is the inverse of the number of iteration steps Q_{Outer} which we repeat in order to obtain $\mathbf{z}$ when measuring $|\Psi_F\rangle$. The overall complexity of the algorithm for the configuration problem becomes

$$\begin{aligned} T^{\text{Q}}_{\text{BLS}} = \mathcal{O}\Bigg(|L_1| \Bigg(\sqrt{\left(\frac{|L_2|}{|L_2(\mathbf{x}_1)|}\right)} + \ldots + \sqrt{\left(\frac{|L_k|}{|L_k(\mathbf{x}_1, \ldots, \mathbf{x}_{k-2})|}\right)} \Bigg) \\ \cdot \sqrt{|L_2(\mathbf{x}_1)| \cdot |L_3(\mathbf{x}_1, \mathbf{x}_2)| \cdot \ldots \cdot |L_k(\mathbf{x}_1, \ldots, \mathbf{x}_{k-2})|} \Bigg), \end{aligned} \tag{9}$$

where all the filtered lists in the above expression are assumed to be of expected size greater than or equal to 1. For certain target configurations intermediate lists are of sizes less than 1 in expectation (see Eq. (1)), which should be understood as the expected number of times we need to construct these lists to obtain 1 element in them. So there will exist elements in the superposition for which a solution does not exist. Still, for the elements, for which a solution does exist (we expect $\mathcal{O}(1)$ of these), we perform $\mathcal{O}(\sqrt{|L|})$ Grover iterations during the "inner" Grover procedure, and during the "outer" procedure these "good" elements contribute a $\mathcal{O}(1)$ factor to the running time. Therefore formally, each $|L_i(\mathbf{x}_1, \ldots, \mathbf{x}_{i-1})|$ in Eq. (9) should be changed to $\max\{1, |L_i(\mathbf{x}_1, \ldots, \mathbf{x}_{i-1})|\}$. Alternatively, one can enforce that intermediate lists are of size greater than 1 by choosing the target configuration appropriately.

The procedure we have just described is summarised in Algorithm 2. If we want to use this algorithm to solve the Approximate k-List problem (Definition 3.1), we additionally require that the number of output solutions is equal to the size of the input lists. Using the results of Theorem 3.1, we can express the complexity of Algorithm 2 for the Approximate k-List problem via the determinant of the target configuration C and its minors.

Theorem 4.1. *Given input $L_1, \ldots, L_k \subset \mathsf{S}^{d-1}$ and a configuration $C \in \mathscr{C}$, such that Eq. (2) holds, Algorithm 2 solves the Approximate k-List problem in time*

$$T_{\text{k-List}} = \widetilde{\mathcal{O}}\left(\left(\left(\frac{1}{\det(C)} \right)^{\frac{k+1}{2(k-1)}} \cdot \sqrt{\det(C[1 \ldots k-1])} \right)^{d/2} \right) \tag{10}$$

using $M_{\text{k-List}} = \widetilde{\mathcal{O}}\left(\left(\frac{1}{\det(C)} \right)^{\frac{d}{2(k-1)}} \right)$ classical memory and $\mathrm{poly}(d)$ quantum memory with success probability at least $1 - 2^{-\Omega(d)}$.

Algorithm 2 Quantum algorithm for the Configuration Problem

Input: $L_1, \ldots, L_k$– lists of vectors from S^{d-1}, target configuration $C_{i,j} = \langle \mathbf{x}_i \, , \mathbf{x}_j \rangle \in \mathbb{R}^{k \times k}$– a Gram matrix, $\varepsilon > 0$.

Output: L_{out}– list of k-tuples $(\mathbf{x}_1, \ldots, \mathbf{x}_k) \in L_1 \times \cdots \times L_k$, s.t. $|\langle \mathbf{x}_i \, , \mathbf{x}_j \rangle - C_{ij}| \leq \varepsilon$ for all i, j.

1: $L_{\text{out}} \leftarrow \emptyset$
2: **for all** $\mathbf{x}_1 \in L_1$ **do**
3: Prepare the state $|\Psi_{L_2}\rangle \otimes \ldots \otimes |\Psi_{L_k}\rangle$
4: **for all** $i = 2 \ldots k-1$ **do**
5: Run Grover's on the i^{th} register with the checking function $f_{[i-1],i}$ to transform the state $|\Psi_{L_i}\rangle$ to the state $|\Psi_{L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1})}\rangle$.
6: Run Grover's on the k^{th} register with the checking function $f_{[k-2],k}$ to transform the state $|\Psi_{L_k}\rangle$ to the state $|\Psi_{L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{k-2})}\rangle$.
7: Let $\mathcal{A}$ be unitary that implements steps 3–6, i.e.

$$\mathcal{A}\,|\mathbf{0}^{\otimes k}\rangle \rightarrow |\Psi_F\rangle\,.$$

8: Run amplitude amplification using the unitary $-\mathcal{A}R\mathcal{A}^{-1}O_g$, where g is defined in Eq. (8).
9: Measure all the registers, obtain a tuple $(\mathbf{x}_2, \ldots, \mathbf{x}_k)$.
10: **if** $(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ satisfies C **then**
11: $L_{\text{out}} \leftarrow L_{\text{out}} \cup \{(\mathbf{x}_1, \ldots, \mathbf{x}_k)\}$.

Proof 4.1. *From Theorem 3.1, the input lists $L_1, \ldots, L_k$ should be of sizes $|L| = \widetilde{\mathcal{O}}\left(\left(\frac{1}{\det(C)}\right)^{\frac{d}{2(k-1)}}\right)$ to guarantee a sufficient number of solutions. This determines the requirement for classical memory. Furthermore, since all intermediate lists are stored in the superposition, we require quantum registers of size* $\text{poly}(d)$.

Next, we can simplify the expression for $T^{\text{Q}}_{\text{BLS}}$ given in Eq. (9) *by noting that $|L_2(\mathbf{x}_1)| \geq |L_3(\mathbf{x}_1, \mathbf{x}_2)| \geq \ldots \geq |L_{k-1}(\mathbf{x}_1, \ldots, \mathbf{x}_{k-2})| = |L_k(\mathbf{x}_1, \ldots, \mathbf{x}_{k-2})|$. The dominant term in the sum appearing in Eq.* (9) *is $\sqrt{\left(\frac{|L_k|}{|L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{k-2})|}\right)}$.*

From Theorem 3.2, the product $\sqrt{|L_2(\mathbf{x}_1)| \cdot \ldots \cdot |L_{k-1}(\mathbf{x}_1, \ldots, \mathbf{x}_{k-2})|}$ in Eq. (9) *can be simplified to $|L|^{\frac{k-2}{2}} \left(\sqrt{\det(C[1 \ldots k-1])}\right)^{d/2}$, from where we arrive at the expression for $T_{\text{k-List}}$ as in the statement.*

The success probability of Algorithm 2 is determined by the success probability of the amplitude amplification run in Step 8. For this we consider the precise form of the state $|\Psi_F\rangle$ given in Eq. (6). *This state is obtained by running $k-1$ (sequential) Grover algorithms. Each tensor $|\Psi_{L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1})}\rangle$ in this state is a superposition*

$$|\Psi_{L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1})}\rangle = \sqrt{\frac{1-\epsilon_i}{|L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1})|}} \sum_{\mathbf{x} \in L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1})} |\mathbf{x}\rangle + \sqrt{\frac{\epsilon_i}{|L_i \setminus L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1})|}} \sum_{\mathbf{x} \in L_i \setminus L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1})} |\mathbf{x}\rangle\,,$$

where $\epsilon_i < \frac{|L_i(\mathbf{x}_1,\ldots,\mathbf{x}_i)|}{|L_i|} \leq 2^{-\Omega(d)}$. The first inequality comes from the success

probability of Grover's algorithm, Theorem 2.1, the second inequality is due to the fact that all lists on a "lower" level are exponentially smaller than lists on a "higher" level, see Theorem 3.2. Therefore, the success probability of the amplitude amplification is given by $\prod_{i=2}^{k-1} \frac{1-\epsilon_i}{|L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1})|} \cdot \frac{1-\epsilon_k}{|L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{k-2})|} \geq (1 - 2^{-\Omega(d)}) \prod_{i=2}^{k-1} |L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1})|^{-1}$. *According to Theorem 2.2, after performing* $\mathcal{O}\left(\prod_{i=2}^{k} |L_i(\mathbf{x}_1,\ldots,\mathbf{x}_i)|\,|L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{k-2})|\right)$ *amplitude amplification iterations, in Step 9 we measure a "good"* $(\mathbf{x}_2,\ldots,\mathbf{x}_k)$ *with probability at least* $1 - 2^{-\Omega(d)}$.

4.1 Quantum Version of the Configuration Search Algorithm from [HKL18]

The main difference between the two algorithms for the configuration problem – the algorithm due to Bai–Laarhoven–Stehlé [BLS16] and due to Herold–Kirshanova–Laarhoven [HKL18] – is that the latter constructs intermediate filtered lists, Figure 2. We use quantum enumeration to construct and classically store these lists.

For a fixed $\mathbf{x}$, quantum enumeration repeatedly applies Grover's algorithm to an input list L_i, where each application returns a random vector from the filtered list $L_i(\mathbf{x})$ with probability greater than $1 - 2^{-\Omega(d)}$. The quantum complexity of obtaining one vector from $L_i(\mathbf{x})$ is $\mathcal{O}\left(\sqrt{\frac{|L_i|}{|L_i(\mathbf{x})|}}\right)$. We can also check that the returned vector belongs to $L_i(\mathbf{x})$ by checking its inner product with $\mathbf{x}$. Repeating this process $\widetilde{\mathcal{O}}(|L_i(\mathbf{x})|)$ times, we obtain the list $L_i(\mathbf{x})$ stored classically in time $\widetilde{\mathcal{O}}(\sqrt{|L_i| \cdot |L_i(\mathbf{x})|})$. The advantage of constructing the lists $L_i(\mathbf{x})$ is that we can now efficiently prepare the state $|\Psi_{L_2(\mathbf{x})}\rangle \otimes \ldots \otimes |\Psi_{L_k(\mathbf{x})}\rangle$ (cf. Line 3 in Algorithm 2) and run amplitude amplification on the states $|\Psi_{L_i(\mathbf{x})}\rangle$ rather than on $|\Psi_{L_i}\rangle$. This may give a speed up if the complexity of the Steps 3–11 of Algorithm 2, which is of order $\widetilde{\mathcal{O}}(T^{\mathrm{Q}}_{\mathrm{BLS}}/\,|L_1|)$, dominates the cost of quantum enumeration, which is of order $\widetilde{\mathcal{O}}(\sqrt{|L_i| \cdot |L_i(\mathbf{x})|})$. In general, we can continue creating the "levels" as in [HKL18] (see Figure 2b) using quantum enumeration and at some level switch to the quantum BLS style algorithm. For example, for some level $1 < j \leq k-1$, we apply quantum enumeration to obtain $L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{j-1})$ for all $i > j$. Then for all $(j-1)$-tuples $(\mathbf{x}_1,\ldots,\mathbf{x}_{j-1}) \in L_1 \times \ldots \times L_{j-1}(\mathbf{x}_1,\ldots,\mathbf{x}_{j-2})$, apply Grover's algorithm as in steps 3–11 of Algorithm 2 but now to the states $|\Psi_{L_j(\mathbf{x}_1,\ldots,\mathbf{x}_{j-1})}\rangle \otimes \ldots \otimes |\Psi_{L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{j-1})}\rangle$. Note that since we have these lists stored in memory, we can efficiently create this superposition. In this way we obtain a quantum "hybrid" between the HKL and the BLS algorithms: until some level j, we construct the intermediate lists using quantum enumeration, create superpositions over all the filtered lists at level j for some fixed values $\mathbf{x}_1,\ldots,\mathbf{x}_{j-1}$, and apply Grover's algorothm to find (if it exists) the $(k-j+1)$ tuple $(\mathbf{x}_j,\ldots,\mathbf{x}_k)$. Pseudocode for this approach is given in Algorithm 3.

Let us now analyse Algorithm 3. To simplify notation, we denote $L_i^{(j)} = L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{j-1})$ for all $i \geq j$, letting $L_i^{(1)}$ be the input lists L_i (so the upper index denotes the level of the list). All $\mathcal{O}$ notations are omitted. Each quantum enumeration of $L_i^{(j)}$ from $L_i^{(j-1)}$ costs $\sqrt{\left|L_i^{(j-1)}\right| \left|L_i^{(j)}\right|}$. On level $1 \leq \ell \leq j-1$, we repeat such an enumeration $\prod_{r=1}^{\ell-1} \left|L_r^{(r)}\right|$ times to create the

Algorithm 3 Hybrid quantum algorithm for the Configuration Problem

Input: $L_1, \ldots, L_k$, lists of vectors from S^{d-1}, target configuration $C_{i,j} = \langle \mathbf{x}_i \, , \mathbf{x}_j \rangle \in \mathbb{R}^{k \times k}$, $\varepsilon > 0$, $2 \leq j \leq k-1$, level we construct the intermediate filtered lists until.

Output: L_{out}– list of k-tuples $(\mathbf{x}_1, \ldots, \mathbf{x}_k) \in L_1 \times \cdots \times L_k$, s.t. $|\langle \mathbf{x}_i \, , \mathbf{x}_j \rangle - C_{ij}| \leq \varepsilon$ for all i, j.

1: $L_{\text{out}} \leftarrow \emptyset$
2: **for all** $\mathbf{x}_1 \in L_1$ **do**
3: Use quantum enumeration to construct $L_i(\mathbf{x}_1)$ for $\forall i \geq 2$
4: **for all** $\mathbf{x}_2 \in L_2(\mathbf{x}_1)$ **do**
5: Use quantum enumeration to construct $L_i(\mathbf{x}_1, \mathbf{x}_2)$, $\forall i \geq 3$
6: $\ddots$
7: **for all** $\mathbf{x}_{j-1} \in L_{j-1}(\mathbf{x}_1, \ldots, \mathbf{x}_{j-2})$ **do**
8: Use quantum enumeration to construct $L_i(\mathbf{x}_1, \ldots, \mathbf{x}_{j-1})$, $\forall i \geq j$
9: Prepare the state $|\Psi_{L_j(\mathbf{x}_1,\ldots,\mathbf{x}_{j-1})}\rangle \otimes \ldots \otimes |\Psi_{L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{j-1})}\rangle$
10: **for all** $i = j+1 \ldots k-1$ **do**
11: Run Grover's on the i^{th} register with the checking function $f_{[i-1],i}$ to transform the state $|\Psi_{L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{j-1})}\rangle$ to the state $|\Psi_{L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1})}\rangle$.
12: Run Grover's on the k^{th} register with the checking function $f_{[k-2],k}$ to transform the state $|\Psi_{L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{j-1})}\rangle$ to the state $|\Psi_{L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{k-2})}\rangle$.
13: Let $\mathcal{A}$ be unitary that implements Steps 9–12, i.e.

$$\mathcal{A}\,|\mathbf{0}^{\otimes(k-j+1)}\rangle \to |\Psi_{L_j(\mathbf{x}_1,\ldots,\mathbf{x}_{j-1})}\rangle \otimes |\Psi_{L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{k-2})}\rangle$$

14: Run amplitude amplification using the unitary $-\mathcal{A}R\mathcal{A}^{-1}O_g$, where g is defined in Eq. (8).
15: Measure all the registers, obtain a tuple $(\mathbf{x}_j, \ldots, \mathbf{x}_k)$.
16: **if** $(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ satisfies C **then**
17: $L_{\text{out}} \leftarrow L_{\text{out}} \cup \{(\mathbf{x}_1, \ldots, \mathbf{x}_k)\}$.

intermediate lists, once for each $(\mathbf{x}_1, \ldots, \mathbf{x}_{\ell-1})$. Once the lists $L_i^{(j)}$, $i \geq j$, are constructed, Grover's algorithm gives the state $|\Psi_{L_j^{(j)}}\rangle \ldots |\Psi_{L_{k-1}^{(k-1)}}\rangle \, |\Psi_{L_k^{(k-1)}}\rangle$ in time $\left(\sqrt{\frac{\left|L_{j+1}^{(j)}\right|}{\left|L_{j+1}^{(j+1)}\right|}} + \ldots + \sqrt{\frac{\left|L_{k-1}^{(j)}\right|}{\left|L_{k-1}^{(k-1)}\right|}} + \sqrt{\frac{\left|L_k^{(j)}\right|}{\left|L_k^{(k-1)}\right|}}\right)$ (Steps 11–12 in Algorithm 3). On Step 14 the unitary $\mathcal{A}$ must be executed $\sqrt{\left|L_j^{(j)}\right| \cdot \ldots \cdot \left|L_{k-1}^{(k-1)}\right| \cdot \left|L_k^{(k-1)}\right|}$ times to ensure that the measurement of the system gives the "good" tuple $(\mathbf{x}_j, \ldots, \mathbf{x}_k)$. Such tuples may not exist: for $j \geq 3$, i.e. for *fixed* $\mathbf{x}_1, \mathbf{x}_2$, we expect to have less than 1 such tuples. So most of the time, the measurement will return a random $(k-j+1)$-tuple, which we classically check against the target configuration C. Overall, given on input a level j, the runtime of Algorithm 3

is

$$T^{\mathrm{Q}}_{\mathrm{Hybrid}}(j) = \max_{1\le\ell\le j-1}\left\{\prod_{r=1}^{\ell-1}\left|L_r^{(r)}\right|\cdot\max_{\ell\le i\le k}\left\{\sqrt{\left|L_i^{(\ell)}\right|\left|L_i^{(\ell+1)}\right|}\right\},\right.$$
$$\prod_{r=1}^{j-1}\left|L_r^{(r)}\right|\left(\sqrt{\frac{\left|L_{j+1}^{(j)}\right|}{\left|L_{j+1}^{(j+1)}\right|}}+\ldots+\sqrt{\frac{\left|L_{k-1}^{(j)}\right|}{\left|L_{k-1}^{(k-1)}\right|}}+\sqrt{\frac{\left|L_k^{(j)}\right|}{\left|L_k^{(k-1)}\right|}}\right) \tag{11}$$
$$\left.\cdot\sqrt{\left|L_j^{(j)}\right|\cdot\ldots\left|L_{k-1}^{(k-1)}\right|\cdot\left|L_k^{(k-1)}\right|}\right\}.$$

Similar to Eq. (9), all the list sizes in the above formula are assumed to be greater than or equal to 1. If, for a certain configuration it happens that the expected size of a list is less than 1, it should be replaced with 1 in this expression. The above complexity can be expressed via the determinant and subdeterminants of the target configuration C using Theorem 3.2. An optimal value of j for a given C can be found using numerical optimisations by looking for j that minimises Eq. (11).

Speed-ups with nearest neighbour techniques. We can further speed up the creation of filtered lists in both Algorithms 2 and 3 with a quantum version of nearest neighbour search. In particular, in Appendix B we describe a locality sensitive filtering (LSF) technique (first introduced in [BDGL16]) in the quantum setting, extending the idea of Laarhoven [Laa15] to $k > 2$.

Numerical optimisations. We performed numerical optimisations for the target configuration C which minimises the runtime of the two algorithms for the configuration problem given in this section. The upper part of Table 2 gives time optimal c for Eq. (10) and the c' of the corresponding memory requirements for various k. These constants decrease with k and, eventually, those for time become close to the value 0.2989. The explanation for this behaviour is the following: looking at Eq. (9) the expression decreases when the lists $L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1})$ under the square root become smaller. When k is large enough, in particular, once $k \ge 6$, there is a target configuration that ensures that $|L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1})|$ are of expected size 1 for levels $i \ge 4$. So for $k \ge 6$, under the observation that the maximal value in the sum appearing in Eq. (9) is attained by the last summand, the runtime of Algorithm 2 becomes $T^{\mathrm{Q}}_{\mathrm{BLS}} = |L_1|^{3/2}\cdot\sqrt{|L_2(\mathbf{x}_1)|\,|L_3(\mathbf{x}_1,\mathbf{x}_2)|}$. The list sizes can be made explicit using Eq. (3) when a configuration C is such that $|L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1})|$ are of expected size 1. Namely, for $k \ge 6$ and for configuration C that minimises the runtime exponent, Eq. (9) with the help of Eq. (3) simplifies to $\left(\left(\frac{1}{\det C}\right)^{\frac{5}{2(k-1)}}\sqrt{\det C[1,2,3]}\right)^{d/2}$.

The optimal runtime exponents for the hybrid, Algorithm 3, with $j = 2$ are given in the middle part of Table 2. Experimentally, we establish that $j = 2$ is optimal for small values of k and that this algorithm has the same behaviour for large values of k as Algorithm 2. The reason is the following: for the runtime optimal configuration C the intermediate lists on the same level increase in size "from left to right", i.e. $|L_2(\mathbf{x}_1)| \le |L_3(\mathbf{x}_1)| \le \ldots, \le |L_k(\mathbf{x}_1)|$. It turns out that $|L_k(\mathbf{x}_1)|$ becomes almost $|L_k|$ (i.e. the target inner product is very close to 0),

k	2	3	4	5	6	...	28	29	30
Quantum version of [BLS16] Algorithm 2									
Time	0.3112	0.3306	0.3289	0.3219	0.3147	...	0.29893	0.29893	0.29893
Space	0.2075	0.1907	0.1796	0.1685	0.1596	...	0.1395	0.1395	0.1395
Quantum Hybrid version of [BLS16, HKL18] Algorithm 3									
Time	0.3112	0.3306	0.3197	0.3088	0.3059	...	0.29893	0.29893	0.29893
Space	0.2075	0.1907	0.1731	0.1638	0.1595	...	0.1395	0.1395	0.1395
Low memory Quantum Hybrid version of [BLS16, HKL18] Algorithm 3									
Time	0.3112	0.3349	0.3215	0.3305	0.3655	...	0.6352	0.6423	0.6490
Space	0.2075	0.1887	0.1724	0.1587	0.1473	...	0.0637	0.0623	0.0609

Table 2: Asymptotic complexity exponents for the approximate k-List problem, base 2. The top part gives optimised runtime exponents and the corresponding memory exponents for Algorithm 2. These are the results of the optimisation (minimisation) of the runtime expression given in Eq. (10). The middle part gives the runtime and memory exponents for Algorithm 3, again optimising for time, with $j = 2$, i.e. when we use quantum enumeration to create the second level lists $L_i(\mathbf{x}_1)$, $i \geq 2$. The bottom part gives the exponents for Algorithm 3 with $j = 2$ in the memory optimal setting.

so quantumly enumerating this list brings no advantage over Algorithm 2 where we use the initial list L_k, of essentially the same size, in Grover's algorithm.

5 Quantum Configuration Search via k-Clique Listing

In this section we introduce a distinct approach to finding solutions of the configuration problem, Definition 3.3, via k-clique listing in graphs. We achieve this by repeatedly applying k-clique finding algorithms to the graphs. Throughout this section we assume that $L_1 = \cdots = L_k = L$. We first solve the configuration problem with $k = 3$, C the balanced configuration with all off diagonals equal to $-1/3$ and the size of L determined by Eq. (2). We then adapt the idea to the case for general k. In Appendix C.1 we give the $k = 4$ balanced case to elucidate the jump to the general k case, and in Appendix C.2 the case for general k with unbalanced configurations.

Let $G = (V, E)$ be an undirected graph with known vertices and an oracle $O_G \colon V^2 \to \{\texttt{True}, \texttt{False}\}$. On input $(\mathbf{x}_1, \mathbf{x}_2) \in V^2$, O_G returns `True` if $(\mathbf{x}_1, \mathbf{x}_2) \in E$ and `False` otherwise. A k-clique is $\{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ such that $O_G(\mathbf{x}_i, \mathbf{x}_j) = \texttt{True}$ for $i \neq j$. Given k in the balanced case, $(\mathbf{x}_i, \mathbf{x}_j) \in E \iff |\langle \mathbf{x}_i \, , \mathbf{x}_j \rangle + 1/k| \leq \varepsilon$ for some $\varepsilon > 0$. In the unbalanced case $(\mathbf{x}_i, \mathbf{x}_j) \in E \iff |\langle \mathbf{x}_i, \mathbf{x}_j \rangle - C_{i,j}| \leq \varepsilon$ (considered in Appendix C.2). In both cases, the oracle computes a d dimensional inner product and compares the result against the target configuration. Throughout we let $|V| = n$ and $|E| = m$.

5.1 The Triangle Case

We start with the simple triangle finding algorithm of [BdWD+01]. A triangle is a 3-clique. Given the balanced configuration and $k = 3$ on S^{d-1}, we have

$$n = |L| = \widetilde{\mathcal{O}}\left((3\sqrt{3}/4)^{d/2}\right), \; m = |L|\,|L(\mathbf{x}_1)| = \widetilde{\mathcal{O}}\left(n^2(8/9)^{d/2}\right) \tag{12}$$

by Eq. (2) and Theorem 3.2 respectively,[6] We expect $\Theta(n)$ triangles to be found [HKL18]. The algorithm of [BdWD+01] consists of three steps:

1. Use Grover's algorithm to find any edge $(\mathbf{x}_1, \mathbf{x}_2) \in E$ among all potential $\mathcal{O}(n^2)$ edges.
2. Given an edge $(\mathbf{x}_1, \mathbf{x}_2)$ from Step 1, use Grover's algorithm to find a vertex $\mathbf{x}_3 \in V$, such that $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ is a triangle.
3. Apply amplitude amplification on Steps 1–2.

Note that the algorithm searches for any triangle in the graph, not a fixed one. To be more explicit about the use of the oracle O_G, below we describe a circuit that returns a triangle. Step 1 takes the state $\frac{1}{n}\sum_{(\mathbf{x}_1,\mathbf{x}_2)\in V^2} |\mathbf{x}_1\rangle \otimes |\mathbf{x}_2\rangle$ and applies $\mathcal{O}(\sqrt{n^2/m})$ times the Grover iteration given by $-H^{\otimes 2\bar{d}} R H^{\otimes 2\bar{d}} O_G$. The output is the state $\sqrt{\frac{\epsilon}{n^2-m}} \sum_{(\mathbf{x}_1,\mathbf{x}_2)\notin E} |\mathbf{x}_1\rangle \otimes |\mathbf{x}_2\rangle + \sqrt{\frac{1-\epsilon}{m}} \sum_{(\mathbf{x}_1,\mathbf{x}_2)\in E} |\mathbf{x}_1\rangle \otimes |\mathbf{x}_2\rangle$, where ϵ represents the probability of failure. We disregard this as in the proof of Theorem 4.1. We then join with a uniform superposition over the vertices to create the state $\frac{1}{\sqrt{m}} \sum_{(\mathbf{x}_1,\mathbf{x}_2)\in E} |\mathbf{x}_1\rangle \otimes |\mathbf{x}_2\rangle \otimes \frac{1}{\sqrt{n}} \sum_{\mathbf{x}_3\in V} |\mathbf{x}_3\rangle$ and apply $-H^{\otimes 3\bar{d}} R H^{\otimes 3\bar{d}} O_G^{\Delta}$ $\mathcal{O}(\sqrt{n})$ times. This oracle O_G^{Δ} outputs `True` on a triple from V^3 if each pair of vertices has an edge. We call the final state $|\Psi_F\rangle$. Let $\mathcal{A}\,|\vec{0}^{\otimes 3}\rangle \to |\Psi_F\rangle$, then we apply amplitude amplification with $\mathcal{A}$ repeated some number of times determined by the success probability of $\mathcal{A}$ calculated below.

Given that oracle queries O_G or O_G^{Δ} have some $\mathrm{poly}(d)$ cost, we may calculate the time complexity of this method directly from the query complexity. The cost of the first step is $\mathcal{O}(\sqrt{n^2/m})$ and the second step $\mathcal{O}(\sqrt{n})$. From Eq. (12), and that the costs of Step 1 and Step 2 are additive, we see that $\mathcal{O}(\sqrt{n})$ dominates, therefore Steps 1–2 cost $\mathcal{O}(\sqrt{n})$. The probability that Step 2 finds a triangle is the probability that Step 1 finds an edge of a triangle. Given that there are $\Theta(n)$ triangles, this probability is $\Theta(n/m)$, therefore by applying the amplitude amplification in Step 3, the cost of finding a triangle is $\mathcal{O}(\sqrt{m})$.[7]

The algorithm finds one of the n triangles uniformly at random. By the coupon collector's problem we must repeat the algorithm $\widetilde{\mathcal{O}}(n)$ times to find all the triangles. Therefore the total cost of finding all triangles is $\widetilde{\mathcal{O}}(n\sqrt{m}) = \widetilde{\mathcal{O}}(|L|^{3/2}|L(\mathbf{x}_1)|^{1/2}) \approx 2^{0.3349d+o(d)}$ using $2^{0.1887d+o(d)}$ memory. This matches the complexity of Algorithm 2 for $k = 3$ in the balanced setting (see Table 2).

[6] As we are in the balanced configuration case, and our input lists are identical, Theorem 3.2 has no dependence on j.

[7] Note that this differs from [BdWD+01] as in general either of Step 1 or 2 may dominate and we also make use of the existence of $\Theta(n)$ triangles.

5.2 The General k-Clique Case

The algorithm generalises to arbitrary constant k. We have a graph with $|L|$ vertices, $|L||L(\mathbf{x}_1)|$ edges, $\ldots$, $|L||L(\mathbf{x}_1)|\ldots|L(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1})|$ i-cliques for $i \in \{3,\ldots,k-1\}$, and $\Theta(|L|)$ k-cliques. The following algorithm finds a k-clique, with $2 \leq i \leq k-1$

1. Use Grover's algorithm to find an edge $(\mathbf{x}_1,\mathbf{x}_2) \in E$ among all potential $\mathcal{O}(|L|^2)$ edges.

$\vdots$

i. Given an i-clique $(\mathbf{x}_1,\ldots,\mathbf{x}_i)$ from step $i-1$, use Grover's algorithm to find a vertex $\mathbf{x}_{i+1} \in V$, such that $(\mathbf{x}_1,\ldots,\mathbf{x}_{i+1})$ is an $(i+1)$-clique.

$\vdots$

k. Apply amplitude amplification on Steps 1–$(k-1)$.

The costs of Steps 1–$(k-1)$ are additive. The dominant term is from Step $k-1$, a Grover search over $|L|$, equal to $\mathcal{O}(\sqrt{|L|})$. To determine the cost of finding one k-clique, we need the probability that Steps 1–$(k-1)$ find a k-clique. We calculate the following probabilities, with $2 \leq i \leq k-2$

1. The probability that Step 1 finds a good edge, that is, an edge belonging to a k-clique.

i. The probability that Step i finds a good $(i+1)$-clique given that Step $i-1$ finds a good i-clique.

In Step 1 there are $\mathcal{O}(|L||L(\mathbf{x}_1)|)$ edges to choose from, $\Theta(|L|)$ of which belong to a k-clique. Thus the success probability of this Step is $\Theta(1/|L(\mathbf{x}_1)|)$. Thereafter, in Step i, given an i-clique $(\mathbf{x}_1,\ldots,\mathbf{x}_i)$ there are $\mathcal{O}(\max\{|L(\mathbf{x}_1,\ldots,\mathbf{x}_i)|,1\})$ $(i+1)$-cliques on the form $(\mathbf{x}_1,\ldots,\mathbf{x}_i,\mathbf{x}_{i+1})$, $\Theta(1)$ of which are good. The success probability of Steps 1–$(k-1)$ is equal to $\Theta\left(\prod_{i=1}^{k-2}\max\{|L(\mathbf{x}_1,\ldots,\mathbf{x}_i)|,1\}^{-1}\right)$. By applying amplitude amplification at Step k, we get the cost

$$\mathcal{O}\left(\sqrt{|L|}\sqrt{\prod_{i=1}^{k-2}\max\{|L(\mathbf{x}_1,\ldots,\mathbf{x}_i)|,1\}}\right),$$

for finding one k-clique. Multiplying the above expression by $\widetilde{\mathcal{O}}(|L|)$ gives the total complexity for finding $\Theta(|L|)$ k-cliques. This matches the complexity of Algorithm 2, Eq. (9), for balanced configurations for all k.

In Appendix C.2 we show how to solve the configuration problem with unbalanced configurations using a graph approach, again achieving the same complexity as Algorithm 2.

6 Quantum Configuration Search via Triangle Listing

Given the phrasing of the configuration problem as a clique listing problem in graphs, we restrict our attention to the balanced $k = 3$ case and appeal to the

wide body of recent work on triangle finding in graphs. Let the notation be as in Section 5, and in particular recall Eq. (12) then a triangle represents a solution to the configuration problem.

We note that the operations counted in the works discussed here are queries to an oracle that returns whether an edge exists between two vertices in our graph. While, in the case of [BdWD$^+$01], it is simple to translate this cost into a time complexity, for the algorithms which use more complex quantum data structures [LGN17] it is not. In particular, the costs of computing various auxiliary databases from certain sets is not captured in the total query cost.

The quantum triangle finding works we consider are [BdWD$^+$01, Gal14, LGN17]. In [BdWD$^+$01] a simple algorithm based on nested Grover search and quantum amplitude amplification is given which finds a triangle in $\mathcal{O}(n+\sqrt{nm})$ queries to O_G. For sufficiently sparse graphs G, with sparsity measured as $m = \mathcal{O}(n^c)$ and G becoming more sparse as c decreases, this complexity attains the optimal $\Omega(n)$. This is the algorithm extended in Section 5 for the k-configuration problem. In [Gal14] an algorithm is given that finds a triangle in $\widetilde{\mathcal{O}}(n^{5/4})$ queries to O_G. This complexity has no dependence on sparsity and is the currently best known result for generic graphs. Finally in [LGN17] an interpolation between the two previous results is given as the sparsity of the graph increases.

Theorem 6.1 ([LGN17, Theorem 1])**.** *There exists a quantum algorithm that solves, with high probability, the triangle finding problem over graphs of n vertices and m edges with query complexity*

$$\begin{cases} \mathcal{O}(n+\sqrt{nm}) & \text{if } 0 \le m \le n^{7/6} \\ \widetilde{\mathcal{O}}(nm^{1/14}) & \text{if } n^{7/6} \le m \le n^{7/5} \\ \widetilde{\mathcal{O}}(n^{1/6}m^{1/3}) & \text{if } n^{7/5} \le m \le n^{3/2} \\ \widetilde{\mathcal{O}}(n^{23/30}m^{4/15}) & \text{if } n^{3/2} \le m \le n^{13/8} \\ \widetilde{\mathcal{O}}(n^{59/60}m^{2/15}) & \text{if } n^{13/8} \le m \le n^2. \end{cases}$$

More specifically it is shown that for $c \in (7/6, 2)$ a better complexity can be achieved than shown in [BdWD$^+$01, Gal14]. Moreover at the end points the two previous algorithms are recovered; [BdWD$^+$01] for $c \le 7/6$ and [Gal14] for $c = 2$. We recall that these costs are in the query model, and that for $c > 7/6$, where we do not recover [BdWD$^+$01], we do not convert them into time complexity.

We explore two directions that follow from the above embedding of the configuration problem into a graph. The first is the most naïve, we simply calculate the sparsity regime (as per [LGN17]) that the graph, constructed as above, lies in and calculate a lower bound on the cost of listing all triangles.

The second splits our list into triples of distinct sublists and considers graphs formed from the union of said triples of sublists. The sublists are parameterised such that the sparsity and the expected number of triangles in these new graphs can be altered.

6.1 Naïve Triangle Finding

With $G = (V, E)$ and n, m as in (12), we expect to have

$$m = \mathcal{O}\left(n^{2+\delta}\right) = \mathcal{O}\left(n^{1.5500}\right), \ \delta = \log(8/9)/\log(3\sqrt{3}/4).$$

Therefore finding a single triangle takes $\widetilde{\mathcal{O}}(n^{23/30}m^{4/15}) = \widetilde{\mathcal{O}}\left(n^{1.1799}\right)$ queries to O_G [LGN17]. If, to list the expected $\Theta(n)$ triangles, we have to repeat this algorithm $\widetilde{\mathcal{O}}(n)$ times this leads to a total O_G query complexity of $\widetilde{\mathcal{O}}(n^{2.1799}) = 2^{0.4114d+o(d)}$ which is not competitive with classical algorithms [HK17] or the approach of Section 5.

6.2 Altering the Sparsity

Let n remain as in Eq.(12) and $\gamma \in (0,1)$ be such that we consider $\Gamma = n^{1-\gamma}$ disjoint sublists of L, $\ell_1, \ldots, \ell_\Gamma$, each with $n' = n^\gamma$ elements. There are $\mathcal{O}(n^{3(1-\gamma)})$ triples of such sublists, (ℓ_i, ℓ_j, ℓ_k), with i, j, k pairwise not equal and the union of the sublists within one triple, $\ell_{ijk} = \ell_i \cup \ell_j \cup \ell_k$, has size $\mathcal{O}(n')$. Let $G_{ijk} = (\ell_{ijk}, E_{ijk})$ with $(\mathbf{x}_1, \mathbf{x}_2)$ in $\ell_{ijk} \times \ell_{ijk}$, $(\mathbf{x}_1, \mathbf{x}_2) \in E_{ijk} \iff |\langle \mathbf{x}_1, \mathbf{x}_2 \rangle + 1/3| \leq \varepsilon$ as before. Using Theorem 3.2, each G_{ijk} is expected to have

$$m' = \mathcal{O}\left(|\ell_{ijk}|\,|\ell_{ijk}(x_1)|\right) = \mathcal{O}\left((n')^2(8/9)^{d/2}\right) = \mathcal{O}\left(n^{2\gamma}(8/9)^{d/2}\right)$$

edges. By listing all triangles in all G_{ijk} we list all triangles in G, and as n is chosen to expect $\Theta(n)$ triangles in G, we have sufficiently many solutions for the underlying k-List problem. We expect, by Theorem 3.2

$$\begin{aligned}|\ell_{ijk}||\ell_{ijk}(\mathbf{x}_1)||\ell_{ijk}(\mathbf{x}_1, \mathbf{x}_2)| &= |\ell_{ijk}|\left(|\ell_{ijk}|(8/9)^{d/2}\right)\left(|\ell_{ijk}|(2/3)^{d/2}\right) \\ &= \mathcal{O}(n^{3\gamma})(16/27)^{d/2} = \mathcal{O}(n^{3\gamma-2})\end{aligned}$$

triangles per ℓ_{ijk}. We must at least test each ℓ_{ijk} once, even if $\mathcal{O}(n^{3\gamma-2})$ is subconstant. The sparsity of ℓ_{ijk} given γ is calculated as

$$m' = \mathcal{O}\left((n')^{2+\beta(\gamma)}\right),\ \beta(\gamma) = \frac{\log(8/9)}{\gamma\log(3\sqrt{3}/4)}.$$

For given γ the number of ℓ_{ijk} to test is $\mathcal{O}(n^{3(1-\gamma)})$, the number of triangles to list per ℓ_{ijk} is $\mathcal{O}(n^{3\gamma-2})$ – we always perform at least one triangle finding attempt and assume listing them all takes $\widetilde{\mathcal{O}}(n^{3\gamma-2})$ repeats – and we are in the sparsity regime $c(\gamma) = 2 + \beta(\gamma)$ [LGN17]. Let a, b represent the exponents of n', m' respectively[8] in Theorem 6.1 given by $m' = (n')^{c(\gamma)}$. We therefore minimise, for $\gamma \in (0,1)$, the exponent of n in $\mathcal{O}(n^{3(1-\gamma)}) \cdot \widetilde{\mathcal{O}}(n^{3\gamma-2}) \cdot \widetilde{\mathcal{O}}((n')^a(m')^b)$,

$$3(1-\gamma) + \max\{0, 3\gamma - 2\} + a\gamma + \left(2\gamma + \frac{\log(8/9)}{\log(3\sqrt{3}/4)}\right) b.$$

The minimal query complexity of $n^{1.7298+o(d)} = 2^{0.326d+o(d)}$ is achieved at $\gamma = \frac{2}{3}$.

The above method leaves open the possibility of finding the same triangle multiple times. In particular if a triangle exists in $G_{ij} = (\ell_{ij}, E_{ij})$, with ℓ_{ij} and E_{ij} defined analogously to ℓ_{ijk} and E_{ijk}, then it will be found in G_{ijk} for all k, that is $\mathcal{O}(n^{1-\gamma})$ many times. Worse yet is the case where a triangle exists in $G_i = (\ell_i, E_i)$ where it will be found $\mathcal{O}(n^{2(1-\gamma)})$ times. However, in both cases the total number of rediscoveries of the same triangle does not affect the asymptotic complexity of this approach. Indeed in the ℓ_{ij} case this number is the product $\mathcal{O}(n^{2(1-\gamma)}) \cdot \mathcal{O}(n^{3\gamma} \cdot (8/9)^{d/2}) \cdot \mathcal{O}(n^{1-\gamma}) = \mathcal{O}(n)$, the product of the

[8] Note that we are considering G_{ijk} rather than G here, hence the $n \leftrightarrow n', m \leftrightarrow m'$ notation change.

number of ℓ_{ij}, the number of triangles[9] per ℓ_{ij} and the number of rediscoveries per triangle in ℓ_{ij} respectively. Similarly, this value remains $\mathcal{O}(n)$ in the ℓ_i case and as we are required to list $\mathcal{O}(n)$ triangles the asymptotic complexity remains $\mathcal{O}(n)$.

7 Parallelising Quantum Configuration Search

In this section we deviate slightly from the k-List problem and the configuration framework and target SVP directly. On input we receive $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\} \subset \mathbb{R}^d$, a basis of $\mathcal{L}(B)$. Our algorithm finds and outputs a short vector from $\mathcal{L}(B)$. As in all the algorithms described above, we will be satisfied with an approximation to the shortest vector and with heuristic analysis.

We describe an algorithm that can be implemented using a quantum circuit of width $\widetilde{\mathcal{O}}(N)$ and depth $\widetilde{\mathcal{O}}(\sqrt{N})$, where $N = 2^{0.2075d+o(d)}$. We therefore require our input and output to be less than $\widetilde{\mathcal{O}}(\sqrt{N})$, and if we were to phrase the 2-Sieve algorithm as a 2-List problem we would not be able to read in and write out the data. Our algorithm uses $\mathrm{poly}(d)$ classical memory. For the analysis, we make the same heuristic assumptions as in the original 2-Sieve work of Nguyen–Vidick [NV08].

All the vectors encountered by the algorithm (except for the final measurement) are kept in quantum memory. Recall that for a pair of normalised vectors $\mathbf{x}_1, \mathbf{x}_2$ to form a "good" pair, i.e. to satisfy $\|\mathbf{x}_1 \pm \mathbf{x}_2\| \leq 1$, it must hold that $|\langle \mathbf{x}_1 \, , \mathbf{x}_2 \rangle| \geq \frac{1}{2}$. The algorithm described below is the quantum parallel version of 2-Sieve. Each step is analysed in the subsequent lemmas.

Algorithm 4 A parallel quantum algorithm for 2-Sieve

Input: $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\} \subset \mathbb{R}^d$ a lattice basis
Output: $\mathbf{v} \in \mathcal{L}(B)$, a short vector from $\mathcal{L}(B)$

1: Set $N \leftarrow 2^{0.2075d+o(d)}$ and set $\lambda = \Theta(\sqrt{d} \cdot \det(B)^{1/d})$ the target length.
2: Generate a list $L_1 \leftarrow \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ of normalised lattice vectors using an efficient lattice sampling procedure, e.g. [Kle00].
3: Construct a list $L_2 \leftarrow \{\mathbf{x}'_1, \ldots, \mathbf{x}'_N\}$ such that $|\langle \mathbf{x}_i \, , \mathbf{x}'_i \rangle| \geq 1/2$ for $\mathbf{x}'_i \in L_1$. If no such $\mathbf{x}'_i \in L_1$ exists, set $\mathbf{x}'_i \leftarrow \mathbf{0}$.
4: Construct a list $L_3 \leftarrow \{\mathbf{y}_i : \mathbf{y}_i \leftarrow \min\{\|\mathbf{x}_i \pm \mathbf{x}'_i\|\}$ for all $i \leq N\}$ and normalise its elements except for the last iteration.
5: Swap the labels L_1, L_3. Reinitialise L_2 and L_3 to the zero state by transferring their contents to auxiliary memory.
6: Repeat Steps 3–5 $\mathrm{poly}(d)$ times.
7: Output a vector from L_1 of Euclidean norm less than λ.

Several remarks about Algorithm 4.

1. The bound on the repetition factor on Step 6 is, as in classical 2-Sieve algorithms, appropriately set to achieve the desired norm of the returned vectors. In particular, it suffices to repeat Steps 2–5 $\mathrm{poly}(d)$ times [NV08].

[9] Given that $|\ell_i| = n^\gamma, |\ell_{ij}| = 2n^\gamma, |\ell_{ijk}| = 3n^\gamma$ the expected numbers of triangles differ only by a constant.

2. In classical 2-Sieve algorithms, if $\mathbf{x}_i$ does not have a match $\mathbf{x}'_i$, it is simply discarded. Quantumly we cannot just discard an element from the system, so we keep it as the zero vector. This is why, as opposed to the classical setting, we keep our lists of exactly the same size throughout all the iterations.

3. The target norm λ is appropriately set to the desired length. The algorithm can be easily adapted to output several, say T, short vectors of $\mathcal{L}(B)$ by repeating Step 7 T times.

Theorem 7.1. *Given on input a lattice basis $\mathcal{L}(B) = \{\mathbf{b}_1, \ldots, \mathbf{b}_d\} \subset \mathbb{R}^d$, Algorithm 4 heuristically solves the shortest vector problem on $\mathcal{L}(B)$ with constant success probability. The algorithm can be implemented using a uniform family of quantum circuits of width $\widetilde{\mathcal{O}}(N)$ and depth $\widetilde{\mathcal{O}}(\sqrt{N})$, where $N = 2^{0.2075d+o(d)}$.*

We prove the above theorem in several lemmas. Here we only give proof sketches for these lemmas, and defer more detailed proofs to Appendix D. In the first lemma we explain the process of generating a database of vectors of size N having N processors. The main routines, Steps 3–5, are analysed in Lemma 7.2. Finally, in Step 7 we use Grover's algorithm to amplify the amplitudes of small norm vectors.

Lemma 7.1. *Step (2) of Algorithm 4 can be implemented using a uniform family of quantum circuits of width $\widetilde{\mathcal{O}}(N)$ and depth* $\operatorname{poly}\log(N)$.

Lemma 7.2. *Steps (3–5) of Algorithm 4 can be implemented using a uniform family of quantum circuits of width $\widetilde{\mathcal{O}}(N)$ and depth $\widetilde{\mathcal{O}}(\sqrt{N})$.*

Lemma 7.3. *Step (7) of the Algorithm 4 can be implemented using a uniform family of quantum circuits of width $\widetilde{\mathcal{O}}(N)$ and depth $\widetilde{\mathcal{O}}(\sqrt{N})$.*

Before we present our proofs for the above lemmas, we briefly explain our computational model. We assume that each input vector $\mathbf{b}_i$ is encoded in $\bar{d} = poly(d)$ qubits and we say that it is stored in a single register. We also consider the circuit model and assume we have at our disposal a set of elementary gates – Toffoli, and all 1-qubit unitary gates (including the Hadamard and Pauli X), i.e. a universal gate set that can be implemented efficiently. We further assume that any parallel composition of unitaries can be implemented simultaneously. For brevity, we will often want to interpret (computations consisting of) parallel processes to be running on parallel processors. We emphasise that this is inconsequential to the computation and our analysis. However, thinking this way greatly helps to understand the physical motivation and convey the intuition behind the computation.

Proof 7.1 (Proof sketch of Lemma 7.1)**.** *The idea is to copy the* cell *of* registers*, $|B\rangle$, encoding the basis $B = \{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ to N processors, where each processor is equipped with* $\operatorname{poly}\log(N)$ *qubits. The state $|B\rangle$ itself is a classical (diagonal) state made of $\bar{d}^2 = \mathcal{O}(\log^2(N))$ qubits. To copy B to all N processors, it takes $\lceil \log(N) \rceil$ steps each consisting of a cascade of CNOT operations.*

Each of the processors samples a single $\mathbf{x}_i$ using a randomised sampling algorithm, e.g. [Kle00]. This is an efficient classical procedure that can be implemented by a reversible circuit of $\operatorname{poly}(d)$ *depth and width. The exact same circuit can be used to realise the sampling on a quantum processor.*

Each processor i, having computed the $\mathbf{x}_i$, now keeps $\mathbf{x}_i$ locally and also copies it to a distinguished cell L1*. The state of the system now can be described as*

$$|\mathbf{x}_1\rangle_{\mathtt{P}_1} |\mathbf{x}_2\rangle_{\mathtt{P}_2} \dots |\mathbf{x}_N\rangle_{\mathtt{P}_N} |\mathbf{x}_1, \mathbf{x}_2 \dots \mathbf{x}_N\rangle_{\mathtt{L1}} |\text{ancilla}\rangle$$

where $\mathtt{P}_i$ is the register in possession of processor i. The total depth of the circuit is $\mathcal{O}(\log(N))$ to copy plus $\operatorname{poly}\log(N)$ to sample plus $\mathcal{O}(1)$ to copy to the list L_1. Each operation is carried out by N processors and uses $\operatorname{poly}\log(N)$ qubits. Thus the total depth of a quantum circuit implementing Step (2) is $\operatorname{poly}\log(N)$ and its width is $\widetilde{\mathcal{O}}(N)$.

Proof 7.2 (Proof sketch of Lemma 7.2)**.** *The key idea to construct the list L_2 is to let each processor $\mathtt{P}_i$, which already has a copy of $|\mathbf{x}_i\rangle, \mathbf{x}_i \in L_1$, search through L_1 (now stored in the distinguished cell* L1*) to find a vector $\mathbf{x}'_i$ such that $|\langle \mathbf{x}_i, \mathbf{x}'_i\rangle| \geq 1/2$ (if no such $\mathbf{x}'_i \in L_1$, set $\mathbf{x}'_i = 0$). The key ingredient is to parallelise this search, i.e. let all processors do the search at the same time. The notion of parallelisation is however only a (correct) interpretation of the operational meaning of the unitary transformations. It is important to stress that we make no assumptions about how data structures are stored, accessed and processed, beyond what is allowed by the axioms of quantum theory and the framework of the circuit model.*

For each processor i, we define a function $f_i(\mathbf{y}) = 1$ if $|\langle \mathbf{x}_i, \mathbf{y}\rangle| \geq 1/2$ and 0 otherwise; and let W_f and D_f be the maximal width and depth of a unitary implementing any f_i. It is possible to implement a quantum circuit of $\widetilde{\mathcal{O}}(N \cdot W_f)$ width and $\widetilde{\mathcal{O}}(\sqrt{N} D_f)$ depth that can in parallel find solutions to all $f_i, 1 \leq i \leq N$ [BBG+13]. This quantum circuit searches through the list in parallel, i.e. each processor can simultaneously access the memory and search. Note, f_i is really a reduced transformation. The "purification" of f_i is a two parameter function $f\colon X \times X \to \{0,1\}$. However, in each processor i, one of the inputs is "fixed and hardcoded" to be $\mathbf{x}_i$. The function f itself admits an efficient implementation in the size of the inputs, since this is the inner product function and also has a classical reversible circuit consisting of Toffoli and NOT gates. Once the search is done, it is expected with probability greater than $1-2^{-\Omega(d)}$ that each processor i will have found an index j_i, s.t. $|\langle \mathbf{x}_i, \mathbf{x}_{j_i}\rangle| \geq 1/2$, $\mathbf{x}_i, \mathbf{x}_{j_i} \in L_1$. One can always check if the processor found a solution, otherwise the search can be repeated a constant number of times. If none of the searches found a "good" j_i, we set $\mathbf{x}_{j_i} = \mathbf{0}$. Else, if any of the searches succeed, we keep that index j_i.

At this point we have a virtual list L_2, which consists of all indices j_i. We create a list L_3 in another distinguished cell, by asking each processor to compute $\mathbf{y}_i^+ = \mathbf{x}_i + \mathbf{x}_{j_i}$ and $\mathbf{y}_i^- = \mathbf{x}_i - \mathbf{x}_{j_i}$ and copy into the i^{th} register the shorter of $\mathbf{y}_i^+$ and $\mathbf{y}_i^-$, in the Euclidean length. The state of the system now is,

$$|\mathbf{x}_1\rangle_{P1} \dots |\mathbf{x}_N\rangle_{PN} |\mathbf{y}_1\rangle_{P1} \dots |\mathbf{y}_L\rangle_{PN} |\mathbf{x}_1 \dots \mathbf{x}_N\rangle_{L1} |\mathbf{y}_1 \dots \mathbf{y}_N\rangle_{L3} |\text{ancilla}\rangle\,.$$

*A swap between qubits say, S and R, is just $CNOT_{SR} \circ CNOT_{RS} \circ CNOT_{SR}$, and thus the Swap in Step 5 between L_1 and L_2 can be done with a depth 3 circuit. Finally reinitialise the lists L_2 and L_3 by swapping them with two registers of equal size that are all initialised to zero. This unloads the data from the main memories (*L2, L3*) and enables processors to reuse them for the next iteration.*

The total depth of the circuit is $\widetilde{\mathcal{O}}(\sqrt{N})$ (to perform the parallel search for "good" indices j_i), $\operatorname{poly}\log N$ *(to compute the elements of the new list L_3 and*

copy them), and $\mathcal{O}(1)$ (to swap the content in memory registers). Thus, in total we have constructed a circuit of $\widetilde{\mathcal{O}}(\sqrt{N})$ depth and $\widetilde{\mathcal{O}}(N)$ width.

Proof 7.3 (Proof sketch of Lemma 7.3)**.** *Given a database of vectors of size N and a norm threshold λ, finding a vector from the database of Euclidean norm less than λ amounts to Grover's search over the database. It can be done with a quantum circuit of depth $\widetilde{\mathcal{O}}(\sqrt{N})$. It could happen that the threshold λ is set to be too small, in which case Grover's search returns a random element form the database. In that case, we repeat the whole algorithm with an increased value for λ. After $\Theta(1)$ repetitions, we heuristically obtain a short vector from $\mathcal{L}(B)$.*

Proof 7.4 (Proof sketch of Theorem 7.1)**.** *As established from the lemmas above, each of Step 2, Steps 3–5 and Step 7 can be realised using a family of quantum circuits of depth and width (at most) $\widetilde{\mathcal{O}}(\sqrt{N})$ and $\widetilde{\mathcal{O}}(N)$ respectively. However, Steps 3–5 run $\mathcal{O}(\mathrm{poly}(d))$ times, thus the total depth of the circuit now goes up by at most a multiplicative factor of $\mathcal{O}(\mathrm{poly}(d)) = \mathcal{O}(\mathrm{poly}\log(N))$. The total depth and width of a circuit implementing Algorithm 4 remains as $\widetilde{\mathcal{O}}(\sqrt{N})$ and $\widetilde{\mathcal{O}}(N)$ respectively as $\widetilde{\mathcal{O}}$ notation suppresses subexponential factors. This concludes the proof.*

7.1 Distributed Configuration Search: Classical Analogue

Algorithm 4 should be compared with a classical model where there are $N = 2^{0.2075d+o(d)}$ computing nodes, each equipped with $\mathrm{poly}(d)$ memory. It suffices for these nodes to have a nearest neighbour architecture, where node i is connected to nodes $i-1$ and $i+1$, and arranged like beads in a necklace. We cost one time unit for $\mathrm{poly}(d)$ bits sent from any node to an adjacent node. A comparable distributed classical algorithm would be where each node, i, receives the basis B and samples a vector $\mathbf{v}_i$. In any given round, node i sends $\tilde{\mathbf{v}}_i$ to node $i+1$ and receives $\tilde{\mathbf{v}}_{i-1}$ from node $i-1$ (in the first round $\tilde{\mathbf{v}}_i := \mathbf{v}_i$). Then each node checks if the vector pair $(\mathbf{v}_i, \tilde{\mathbf{v}}_{i-1})$ gives a shorter sum or difference. If yes, it computes $\mathbf{v}_i^{(2)} = \min\{\mathbf{v}_i \pm \tilde{\mathbf{v}}_{i-1}\}$ and sets $\tilde{\mathbf{v}}_i := \mathbf{v}_{i-1}$. After N rounds every node i has compared their vector $\mathbf{v}_i$ with all N vectors sampled. The vectors $\mathbf{v}_i$ can be discarded and the new round begins with $\mathbf{v}_i^{(2)}$ being the new vector. The process is repeated $\mathrm{poly}(d)$ many times leading to $\mathcal{O}(N) \cdot \mathrm{poly}(d)$ time steps. Thus this distributed algorithm needs $\widetilde{\mathcal{O}}(N) = 2^{0.2075d+o(d)}$ time.

Acknowledgements. Most of this work was done while EK was at ENS de Lyon, supported by ERC Starting Grant ERC-2013-StG-335086-LATTAC and by the European Union PROMETHEUS project (Horizon 2020 Research and Innovation Program, grant 780701). EM is supported by the Swedish Research Counsel (grant 2015-04528) and the Swedish Foundation for Strategic Research (grant RIT17-0005). EWP is supported by the EPSRC and the UK government (grant EP/P009301/1). SRM is supported by the Clarendon Scholarship, Google-DeepMind Scholarship and Keble Sloane–Robinson Award.

We are grateful to the organisers of the Oxford Post-Quantum Cryptography Workshop held at the Mathematical Institute, University of Oxford, March 18–22, 2019, for arranging the session on Quantum Cryptanalysis, where this work began. We would like to acknowledge the fruitful discussions we had with Gottfried Herold during this session.

Finally, we would like to thank the AsiaCrypt'19 reviewers, whose constructive comments helped to improve the quality of this paper.

References

[AD97] Miklós Ajtai and Cynthia Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 284–293, 1997.

[ADH+19] Martin Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn Postlethwaite, and Marc Stevens. The general sieve kernel and new records in lattice reduction. In *Advances in Cryptology – EUROCRYPT 2019*, pages 717–746, 2019.

[ADRS15] Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. Solving the shortest vector problem in 2^n time using discrete gaussian sampling: Extended abstract. In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 733–742, New York, NY, USA, 2015. ACM.

[AGJO+15] Srinivasan Arunachalam, Vlad Gheorghiu, Tomas Jochym-O'Connor, Michele Mosca, and Priyaa Varshinee Srinivasan. On the robustness of bucket brigade quantum RAM. *New Journal of Physics*, 17(12):123010, dec 2015.

[AKS01] Miklós Ajtai, Ravi 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*, STOC '01, pages 601–610, 2001.

[ANS18] Yoshinori Aono, Phong Q. Nguyen, and Yixin Shen. Quantum lattice enumeration and tweaking discrete pruning. In *Advances in Cryptology – ASIACRYPT 2018*, pages 405–434, 2018.

[BBG+13] Robert Beals, Stephen Brierley, Oliver Gray, Aram W Harrow, Samuel Kutin, Noah Linden, Dan Shepherd, and Mark Stather. Efficient distributed quantum computing. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 469(2153):20120686, 2013.

[BBHT98] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(4–5):493–505, 1998. *Fortschritte der Physik*, 46(4–5):493–505, 1998.

[BDGL16] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 10–24, 2016.

[BdWD+01] Harry Buhrman, Ronald de Wolf, Christoph Dürr, Mark Heiligman, Peter Høyer, Frédéric Magniez, and Miklos Santha. Quantum algorithms for element distinctness. In *Proceedings of the 16th Annual Conference on Computational Complexity*, CCC '01, pages 131–137, Washington, DC, USA, 2001. IEEE Computer Society.

[BGJ14] Anja Becker, Nicolas Gama, and Antoine Joux. A sieve algorithm based on overlattices. *LMS Journal of Computation and Mathematics*, 17(A):49–70, 2014.

[BHMT02] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Quantum Computation and Quantum Information: A Millennium Volume*, 305:53–74, 2002. Earlier version in arxiv:quant-ph/0005055.

[BHT97] Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum algorithm for the collision problem. *ACM SIGACT News (Cryptology Column)*, 28, 1997.

[BLS16] Shi Bai, Thijs Laarhoven, and Damien Stehlé. Tuple lattice sieving. *LMS Journal of Computation and Mathematics*, 19:146—162, 2016.

[CCL17] Yanlin Chen, Kai-Min Chung, and Ching-Yi Lai. Space-efficient classical and quantum algorithms for the shortest vector problem. *arXiv e-prints*, Aug 2017.

[CDW17] Ronald Cramer, Léo Ducas, and Benjamin Wesolowski. Short stickelberger class relations and application to ideal-SVP. In *Advances in Cryptology - EUROCRYPT 2017*, pages 324–348, 2017.

[DRS14] D. Dadush, O. Regev, and N. Stephens-Davidowitz. On the closest vector problem with a distance guarantee. In *2014 IEEE 29th Conference on Computational Complexity (CCC)*, pages 98–109, June 2014.

[Duc18] Léo Ducas. Shortest vector from lattice sieving: A few dimensions for free. In *Advances in Cryptology – EUROCRYPT 2018*, pages 125–145, 2018.

[Gal14] F. L. Gall. Improved quantum algorithm for triangle finding via combinatorial arguments. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 216–225, Oct 2014.

[GLM08] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Phys. Rev. Lett.*, 100:160501, Apr 2008.

[GNR10] Nicolas Gama, Phong Q. Nguyen, and Oded Regev. Lattice enumeration using extreme pruning. In *Advances in Cryptology – EUROCRYPT 2010*, pages 257–278, 2010.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 212–219, 1996.

[HK17] Gottfried Herold and Elena Kirshanova. Improved algorithms for the approximate k-list problem in Euclidean norm. In *Public-Key Cryptography – PKC 2017*, pages 16–40, 2017.

[HKL18] Gottfried Herold, Elena Kirshanova, and Thijs Laarhoven. Speed-ups and time–memory trade-offs for tuple lattice sieving. In *Public-Key Cryptography – PKC 2018*, pages 407–436, 2018.

[Kan83] Ravi Kannan. Improved algorithms for integer programming and related lattice problems. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, pages 193–206, 1983.

[Kle00] Philip N. Klein. Finding the closest lattice vector when it's unusually close. In *SODA*, pages 937–941, 2000.

[KLM07] Phillip Kaye, Raymond Laflamme, and Michele Mosca. *An introduction to quantum computing*. Oxford University Press, 2007.

[Kup13] Greg Kuperberg. Another subexponential-time quantum algorithm for the dihedral hidden subgroup problem. In *8th Conference on the Theory of Quantum Computation, Communication and Cryptography, TQC*, pages 20–34, 2013.

[Laa15] Thijs Laarhoven. *Search problems in cryptography*. PhD thesis, Eindhoven University of Technology, 2015.

[LGN17] François Le Gall and Shogo Nakajima. Quantum algorithm for triangle finding in sparse graphs. *Algorithmica*, 79(3):941–959, Nov 2017.

[LMvdP15] Thijs Laarhoven, Michele Mosca, and Joop van de Pol. Finding shortest lattice vectors faster using quantum search. *Designs, Codes and Cryptography*, 77(2):375–400, Dec 2015.

[Map] Maplesoft, a division of Waterloo Maple Inc., Waterloo, Ontario. Standard Worksheet Interface, Maple 2016.0, February 17 2016.

[Mon18] Ashley Montanaro. Quantum-walk speedup of backtracking algorithms. *Theory of Computing*, 14(15):1–24, 2018.

[MV10] Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 1468–1480, 2010.

[NV08] Phong Q. Nguyen and Thomas Vidick. Sieve algorithms for the shortest vector problem are practical. *Journal of Mathematical Cryptology*, pages 181–207, 2008.

[PMHS19] Alice Pellet-Mary, Guillaume Hanrot, and Damien Stehlé. Approx-svp in ideal lattices with pre-processing. In *Advances in Cryptology – EUROCRYPT 2019*, pages 685–716, 2019.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 84–93, 2005.

[Reg09] Oded Regev. Lecture notes: Lattices in computer science. http://www.cims.nyu.edu/~regev/teaching/lattices_fall_2009/index.html, 2009. Accessed: 30-04-2019.

[TKH18] Tadanori Teruya, Kenji Kashiwabara, and Goichiro Hanaoka. Fast lattice basis reduction suitable forÂ massive parallelization and its application to the shortest vector problem. In *Public-Key Cryptography – PKC 2018*, pages 437–460, 2018.

A Configuration Search Algorithm

The pseudocode of the algorithm for the configuration problem from [HK17] is given in Algorithm 5.

Algorithm 5 Algorithm for the Configuration Problem

Input: $L_1, \ldots, L_k$ – lists of vectors from S^{d-1}. $C_{i,j} = \langle \mathbf{x}_i \, , \mathbf{x}_j \rangle \in \mathbb{R}^{k \times k}$ – Gram matrix. $\varepsilon > 0$.
Output: L_{out} – list of k-tuples $\mathbf{x}_1 \in L_1, \ldots, \mathbf{x}_k \in L_k$, s.t. $|\langle \mathbf{x}_i \, , \mathbf{x}_j \rangle - C_{ij}| \leq \varepsilon$, for all i, j.

$L_{\text{out}} \leftarrow \{\}$
for all $\mathbf{x}_1 \in L_1$ **do**
  for all $j = 2 \ldots k$ **do**
    $L_j^{(1)} \leftarrow \textsc{Filter}((\mathbf{x}_1, L_j, C_{1,j}, \varepsilon))$
  for all $\mathbf{x}_2 \in L_2^{(1)}$ **do**
    for all $j = 3 \ldots k$ **do**
      $L_j^{(2)} \leftarrow \textsc{Filter}((\mathbf{x}_2, L_j^{(1)}, C_{2,j}, \varepsilon))$
    $\ddots$
      for all $\mathbf{x}_k \in L_k^{(k-1)}$ **do**
        $L_{\text{out}} \leftarrow L_{\text{out}} \cup \{(\mathbf{x}_1, \ldots \mathbf{x}_k)\}$
return L_{out}

function $\textsc{Filter}((), \mathbf{x}, L, c, \varepsilon)$
  $L' \leftarrow \{\}$
  for all $\mathbf{x}' \in L$ **do**
    if $|\langle \mathbf{x} \, , \mathbf{x}' \rangle - c| \leq \varepsilon$ **then**
      $L' \leftarrow L' \cup \{\mathbf{x}'\}$
  return L'

B Quantum Algorithms for Locality Sensitive Filters

We employed quantum enumeration (Grover's algorithm) to quantumly speed up the creation of filtered lists. Recall how these filtered lists are created classically (see Section 3): given a point $\mathbf{x}$, a list L and a target inner product $C_{1,i}$, we iterate over all $\mathbf{x}_i \in L$ to find those that satisfy $\langle \mathbf{x} , \mathbf{x}_i \rangle \approx C_{1,i}$. This is precisely the problem addressed by neariest neighbour techniques. A nearest neighbour algorithm of particular interest to us is the the Locality Sensitive Filtering (LSF) method [BDGL16]. In this subsection we briefly explain the method, and then we show how it can be sped up with a quantum computer. For $k = 2$ this idea was proposed by Laarhoven in [Laa15].

Spherical Locality Sensitive Filters The idea of LSF is to create a data structure $\mathcal{D}$ of *buckets* $B_{\mathbf{v}}$ of the form $\mathcal{D} = \cup_{\mathbf{v}} B_{\mathbf{v}}$. Each bucket $B_{\mathbf{v}}$, indexed by some *filter vector* $\mathbf{v} \in \mathsf{S}^{d-1}$, contains all vectors from L that are α-close to $\mathbf{v}$, i.e, $\langle \mathbf{x}, \mathbf{v} \rangle \geq \alpha$. Let us denote by $\mathcal{V}$ the set of all filter vectors $\mathbf{v}$. In application, $\mathcal{V}$ is large set: later we set $|\mathcal{V}|$ to be exponential in d.

For a given set of buckets and a list L, building a data structure $\mathcal{D}$ for L means that for each $\mathbf{x} \in L$, we find all $\mathbf{v} \in \mathcal{V}$ that are α-close to $\mathbf{x}$, and, for all such $\mathbf{v}$, we put $\mathbf{x}$ into the bucket $B_{\mathbf{v}}$. Now if we want to find all points from $\mathbf{x} \in L$ that are c-close to a query point $\mathbf{q}$, we first find all β-close $\mathbf{v} \in \mathcal{V}$ to $\mathbf{q}$, and then for all these $\mathbf{v}$ search inside the relevant buckets $B_{\mathbf{v}}$ for $\mathbf{x}$'s, which are c-close to $\mathbf{q}$. The main observation is that quantum search over the relevant buckets can be done using Grover's algorithm. First we create a superposition over the content of the relevant buckets (the bucket labels are known) and then apply Grover's search over the vectors stored in these buckets to find a c-close to $\mathbf{q}$. As we want to find all c-close vectors, we repeat this process until all these vectors are found.

To solve the task of finding relevant filter vectors $\mathbf{v}$ for a given $\mathbf{x}$ (or $\mathbf{q}$) efficiently, vectors $\mathbf{v}$ are chosen with some structure. On the one hand, it enables us to find filter vectors relevant for a given $\mathbf{x}$ in time (up to lower order terms) proportional to the number of such filter vectors. On the other hand, even with such structure, we can argue that the joint distribution of $\mathbf{v}$'s is sufficiently close to $|\mathcal{V}|$ independent vectors from S^{d-1}. For further details on this technique, we refer the reader to [BDGL16]. Here we simply assume that $\mathbf{v}$'s are independent and that we can find all relevant $\mathbf{v}$'s for a given query point in time essentially equal to the size of the output. Note that because we can find all the relevant filters in time asymptotically equal to the output size, applying Grover's search will not speed up this part of the algorithm. The main routines Insert($\mathbf{x}$) – putting $\mathbf{x}$ into all α-close buckets, and Query($\mathbf{q}$) – finding all c-close points for $\mathbf{q}$, are listed in Algorithm 6. The only difference between the classical and quantum routines are in the algorithm Query($\mathbf{q}$), where the search inside a relevant bucket is done using quantum enumeration.

The three parameters α, β, c govern the complexity of LSF. First, they determine the size of $\mathcal{V}$ necessary to find all vectors from a list, which are c-close to a given query point. The insertion parameter α determines the number of buckets a vector is put into and thus, controls the size of buckets. The query parameter β determines the number of relevant buckets for a query $\mathbf{q}$. In the

next theorem, adapted from [HKL18, Theorem 3], we give closed formulas for these costs.

Algorithm 6 Algorithms for spherical locality-sensitive filtering

Parameters: $\alpha, \beta, c, \mathcal{V}$
$\mathcal{D} = \cup_{\mathbf{v}\in\mathcal{V}} B_{\mathbf{v}}$.

1: **function** INSERT($\mathbf{x}$)
2: Find all $\mathbf{v} \in \mathcal{V}$ s.t. $\langle \mathbf{v}\,, \mathbf{x}\rangle \approx \alpha$
3: **for all** $\mathbf{v} \in \mathcal{V}$ s.t. $\langle \mathbf{v}\,, \mathbf{x}\rangle \approx \alpha$ **do**
4: $B_{\mathbf{v}} \leftarrow B_{\mathbf{v}} \cup \{\mathbf{x}\}$

1: **function** QUERY($\mathbf{q}$) ▷ Find $\mathbf{x} \in L$ with $\langle \mathbf{x}\,, \mathbf{q}\rangle \geq c$
2: PointsFound $\leftarrow \emptyset$
3: Find all $\mathbf{v} \in \mathcal{V}$ s.t. $\langle \mathbf{v}\,, \mathbf{x}\rangle \approx \beta$
4: Create a superposition over all relevant buckets $\mathbf{v}$, query all vectors in these buckets

$$|\Psi\rangle = \sqrt{\frac{1}{|\mathcal{V}|\,(1-\beta^2)^{d/2} \cdot |L|\,(1-\alpha^2)^{d/2}}} \sum_{\mathbf{v}\text{ relevant}} |\mathbf{v}\rangle \sum_{\mathbf{x}\in B_{\mathbf{v}}} |\mathbf{x}\rangle$$

5: Run Grover's algorithm on $|\Psi\rangle$ to output an $\mathbf{x}$ using the checking function

$$f(\mathbf{x}, \mathbf{x}') = \begin{cases} 1, & \langle \mathbf{x}\,, \mathbf{q}\rangle \approx c \\ 0, & \text{else.} \end{cases}$$

6: PointsFound $\leftarrow$ PointsFound $\cup \{\mathbf{x}\}$
7: Repeat Steps 4–6 $\widetilde{\mathcal{O}}(|L|\,(1-c^2)^{d/2})$ times until all the c-close vectors are found
8: **return** PointsFound

Theorem B.1 (Complexity of spherical LSF, adapted from [HKL18, Theorem 3])**.** *Assume we are given a list L of iid. uniform points from S^{d-1} of size exponential in d, a target $0 \leq c \leq 1$, and fixed $0 \leq \alpha, \beta \leq 1$. Then the costs of LSF procedures given in Algorithm 6 determined by:*

- ***Number of buckets (filter vectors)*** *used in creating the data structure $\mathcal{D} = \cup_{\mathbf{v}\in\mathcal{V}} B_{\mathbf{v}}$:*

$$|\mathcal{V}| = \frac{\left(\det \left(\begin{smallmatrix} 1 & c \\ c & 1 \end{smallmatrix}\right)\right)^{d/2}}{\left(\det \left(\begin{smallmatrix} 1 & \alpha & \beta \\ \alpha & 1 & c \\ \beta & c & 1 \end{smallmatrix}\right)\right)^{d/2}};$$

- ***Update cost*** *per $\mathbf{x} \in L$ to find all α-close $\mathbf{v} \in \mathcal{V}$: $T_{\text{Update}} = |\mathcal{V}| \cdot (1-\alpha^2)^{d/2}$;*

- ***Preprocessing time*** *to build the data structure $\mathcal{D} = \cup_{\mathbf{v}\in\mathcal{V}} B_{\mathbf{v}}$:*

$$T_{\text{Prep}} = |L| \cdot |\mathcal{V}| \cdot (1-\alpha^2)^{d/2};$$

- ***Memory*** *to store the the data structure $\mathcal{D}$ is $|L| \cdot |\mathcal{V}| \cdot (1-\alpha^2)^{d/2}$;*

- ***Quantum query cost*** *per* $\mathbf{q}$ *to find all relevant* β*-close* $\mathbf{v} \in \mathcal{V}$ *and to find all* c*-close* $\mathbf{x}$*'s:*

$$T_{\text{Query}} = |\mathcal{V}| \cdot (1-\beta^2)^{d/2} + \sqrt{|\mathcal{V}|\,(1-\beta^2)^{d/2} \cdot |L|\,(1-\alpha^2)^{d/2} \cdot |L|\,(1-c^2)^{d/2}}.$$

A proof of this theorem can be found in [HKL18, Appendix D]. The intuition behind all these complexities is the following: for a point $\mathbf{x} \in L \subset \mathsf{S}^{d-1}$ and a fixed vector $\mathbf{v}$, the probability that $\langle \mathbf{v}\,, \mathbf{x} \rangle \approx \alpha$ is $(1-\alpha^2)^{d/2}$. Different to the classical case, we do not assume that the buckets are of expected size 1 (intuitively, this is because iterating over the buckets becomes cheaper in the quantum setting). The number of buckets is determined by the probability that the triple $(\mathbf{x}, \mathbf{v}, \mathbf{q})$, conditioned on the fact that $\langle \mathbf{x}, \mathbf{q} \rangle \approx c$, satisfies all the inner product constraints: $\langle \mathbf{x}, \mathbf{q} \rangle \approx c$, $\langle \mathbf{x}, \mathbf{v} \rangle \approx \alpha$, $\langle \mathbf{v}, \mathbf{q} \rangle \approx \beta$. The quantum query cost consists of first, finding all the relevant buckets in $\mathcal{V}$ and second, enumerating the expected number of $|L|\,(1-c^2)^{d/2}$ many c-close vectors. The expected size of each bucket is $|L|\,(1-\alpha^2)^{d/2}$.

Using LSF for the configuration problem. We make use of locality sensitive filters for the configuration problem as follows: given a target configuration C and the lists L_i, preprocess the lists L_i for $i \geq 2$ using the data structures D_i and the LSF parameters α_i, β_i. Once the preprocessing is done, we can construct the intermediate lists $L_i(\mathbf{x}_1)$ using quantum queries into D_i's for $\mathbf{x}_1 \in L_1$. When $L_i(\mathbf{x}_1)$ are constructed we can either run Grover's algorithm to find the $(k-1)$-tuple analogously to Algorithm 2, or we can construct new LSF data structures now for $L_i(\mathbf{x}_1)$'s and "delay" Grover's search for deeper levels. In general, to create all the filtered lists on level $j \geq 2$, we first prepare the LSF data structures $D_i^{(j-1)}$ for the lists $L_i^{(j-1)}$ on level $j-1$, for all $i \geq j$. This is done classically. Then for each $\mathbf{x}_{j-1} \in L_{j-1}^{(j-1)}$, we create the filtered lists $L_i^{(j)}$ by calling $\textsc{Query}(\mathbf{x}_{j-1})$ into the corresponding $D_i^{(j-1)}$. Contrary to Algorithm 3, here we run quantum enumeration over the relevant buckets rather than over $L_i^{(j-1)}$, which brings a speed-up provided the construction of $D_i^{(j-1)}$'s was not too costly.

Pseudocode of the algorithm for the configuration problem with LSF is given in Algorithm 7. Let us again denote $L_i^{(j)} := L_i(\mathbf{x}_1, \ldots, \mathbf{x}_{j-1})$ for all $i \geq j$, assuming $L_i^{(1)}$ are the input lists. Then the time needed to construct an intermediate list $L_i^{(j)}, i \geq j$ is

$$T(i,j) = \prod_{r=1}^{j-2} \left| L_r^{(r)} \right| \left(T_{\text{Prep}}(i, j-1) + \left| L_{j-1}^{(j-1)} \right| T_{\text{Query}}(j-1, j-1) \right),$$

where $T_{\text{Prep}}(i, j-1)$ is the preprocessing cost for $L_i^{(j-1)}$ and $T_{\text{Query}}(j-1, j-1)$ is the query costs for the elements from $L_{j-1}^{(j-1)}$. Therefore, to create all the intermediate lists on level j, we need time

$$T(j) = \max_{j \leq i \leq k} T(i,j).$$

These costs can be established from Theorem B.1. Once all the lists on level j are constructed, we run Grover search for a good $(k-j+1)$ tuple analogous to

Algorithm 3. Overall complexity of Algorithm 7 (cf. Eq. (11)) is:

$$T^{\mathrm{Q}}_{\mathrm{LSF}}(j) = \max\left\{ T(j), \prod_{r=1}^{j-1} \left|L_r^{(r)}\right| \left(\sqrt{\frac{\left|L_{j+1}^{(j)}\right|}{\left|L_{j+1}^{(j+1)}\right|}} + \ldots + \sqrt{\frac{\left|L_{k-1}^{(j)}\right|}{\left|L_{k-1}^{(k-1)}\right|}} + \sqrt{\frac{\left|L_{k}^{(j)}\right|}{\left|L_{k}^{(k-1)}\right|}} \right) \cdot \sqrt{\left|L_j^{(j)}\right| \cdot \ldots \left|L_{k-1}^{(k-1)}\right| \cdot \left|L_k^{(k-1)}\right|} \right\}. \tag{13}$$

The best runtime exponents are obtained when the locality sensitive techniques are used to construct the intermediate lists, Algorithm 7. It is again optimal to set $j = 2$ (as in the case of Algorithm 3), i.e., when the LSF data structure is used to quantumly construct the second level lists. For $k = 2$ the exponent was established in [Laa15] and is currently the best known when we seek for optimal time. Locality sensitive filters also speed up the configuration search algorithm for $k = 3, 4$. However, the advantage of near neighbour techniques decreases when the lists become smaller, which is what we observe when k increases. Furthermore, the LSF technique (and near neighbour methods, in general) become more expensive when the target inner product c becomes smaller: one requires a larger data structure to allocate c-close vectors, which is why we did not run numerical optimisations for large k's.

k	2	3	4	5
Quantum k-List with LSF				
Time	0.2653	0.2908	0.3013	0.3048
Space	0.2653	0.2908	0.3013	0.3048
Quantum k-List with LSF, low-memory regime				
Time	0.2925	0.3266	0.3178	0.3281
Space	0.2075	0.1887	0.1724	0.1587

Table 3: Asymptotic complexity exponents for the approximate k-List problem using the LSF techniques, Algorithm 7 with $j = 2$.

Algorithm 7 Quantum algorithm for the Configuration problem with LSF

Input: $L_1, \ldots, L_k-$ lists of vectors from S^{d-1}, target configuration $C_{i,j} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle \in \mathbb{R}^{k \times k}-$ a Gram matrix, $\varepsilon > 0$, $2 \leq j \leq k-1-$ level until we construct the intermediate filtered lists; LSF parameters $\{\alpha_{i,r}, \beta_{i,r}\}_{1 \leq i \leq k, 1 \leq r \leq j-1}$

Output: $L_{\text{out}}-$ list of k-tuples $(\mathbf{x}_1, \ldots, \mathbf{x}_k) \in L_1 \times \cdots \times L_k$, s.t. $|\langle \mathbf{x}_i, \mathbf{x}_j \rangle - C_{ij}| \leq \varepsilon$ for all i, j.

1: $L_{\text{out}} \leftarrow \emptyset$
2: Create LSF data structures $\mathcal{D}_i^{(1)}$ with parameters $\{\alpha_{i,1}, \beta_{i,1}\}$ $\forall$Ł$_i, i \geq 2$
3: **for all** $\mathbf{x}_1 \in L_1$ **do**
4: Call QUERY($\mathbf{x}_1$) using $\mathcal{D}_i^{(1)}$ to construct $L_i(\mathbf{x}_1)$ $\forall i \geq 2$
5: Create LSF data structures $\mathcal{D}_i^{(2)}$ with parameters $\{\alpha_{i,2}, \beta_{i,2}\}$ $\forall$Ł$_i, i \geq 3$
6: **for all** $\mathbf{x}_2 \in L_2(\mathbf{x}_1)$ **do**
7: Call QUERY($\mathbf{x}_1$) using $\mathcal{D}_i^{(2)}$ to construct $L_i(\mathbf{x}_1, \mathbf{x}_2)$, $\forall i \geq 3$
8: $\ddots$
9: **for all** $\mathbf{x}_{j-1} \in L_{j-1}(\mathbf{x}_1, \ldots, \mathbf{x}_{j-2})$ **do**
10: Call QUERY($\mathbf{x}_1$) using $\mathcal{D}_i^{(j-1)}$ to construct $L_i(\mathbf{x}_1, \ldots, \mathbf{x}_{j-1})$, $\forall i \geq j$
11: Prepare the state $|\Psi_{L_j(\mathbf{x}_1,\ldots,\mathbf{x}_{j-1})}\rangle \otimes \ldots \otimes |\Psi_{L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{j-1})}\rangle$
12: **for all** $i = j+1 \ldots k-1$ **do**
13: Run Grover's on the i^{th} register with the checking function $f_{[i-1],i}$ to transform the state $|\Psi_{L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{j-1})}\rangle$ to the state $|\Psi_{L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1})}\rangle$.
14: Run Grover's on the k^{th} register with the checking function $f_{[k-2],k}$ to transform the state $|\Psi_{L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{j-1})}\rangle$ to the state $|\Psi_{L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{k-2})}\rangle$.
15: Let $\mathcal{A}$ be unitary that implements Steps 9–12, i.e.

$$\mathcal{A}\,|\mathbf{0}\rangle \rightarrow |\Psi_{L_j(\mathbf{x}_1,\ldots,\mathbf{x}_{j-1})}\rangle \otimes |\Psi_{L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{k-1})}\rangle$$

16: Run amplitude amplification using the unitary $-\mathcal{A}R\mathcal{A}^{-1}O_g$, where g is defined in Eq. (8).
17: Measure all the registers, obtain a tuple $(\mathbf{x}_j, \ldots, \mathbf{x}_k)$.
18: **if** $(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ satisfies C **then**
19: $L_{\text{out}} \leftarrow L_{\text{out}} \cup \{(\mathbf{x}_1, \ldots, \mathbf{x}_k)\}$.

C Some More k-clique Cases

C.1 The $k = 4$ Case

Here we modify the idea of applying quantum triangle listing to the 3-List problem developed in Section 5, to the $k = 4$ case. We use the notations from Section 5. The general case is covered in Section 5.2. For $k = 4$, we obtain a graph with $n = |L| = \mathcal{O}\left(\left(4\sqrt[3]{4}/5\right)^{d/2}\right)$ edges, $m = |L||L(\mathbf{x}_1)|$ vertices, $t = |L||L(\mathbf{x}_1)||L(\mathbf{x}_1, \mathbf{x}_2)|$ triangles and $\Theta(n)$ 4-cliques (cf. Eq. (12)). The algorithm from Section 5.1 can be modified to give an algorithm for finding 4-cliques as follows.

1. Use Grover's algorithm to find an edge $(\mathbf{x}_1, \mathbf{x}_2) \in E$ among all potential $\mathcal{O}(n^2)$ edges.
2. Given an edge $(\mathbf{x}_1, \mathbf{x}_2)$, use Grover's algorithm to find a vertex $\mathbf{x}_3 \in V$, such that $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ is a triangle.
3. Given a triangle $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$, use Grover's algorithm to find a vertex $\mathbf{x}_4 \in V$, such that $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)$ is a 4-clique.
4. Apply amplitude amplification on steps 1–3.

As in the $k = 3$ case, the algorithm returns any 4-clique in the graph, rather than a fixed one. Step 1 searches for any edge, and given that edge Step 2 searches for any triangle containing it.

Step 1 costs $\mathcal{O}(\sqrt{n^2/m})$ and Step 3 costs $\mathcal{O}(\sqrt{n})$. As an edge is expected to be in many triangles, $t/m = \alpha^{d/2}, \alpha > 1$, the cost of Step 2 is below $\mathcal{O}(\sqrt{n})$. As steps 1–3 are additive, step 3 dominates and their cost is $\mathcal{O}(\sqrt{n})$, as before.

For steps 1–3 to find a 4-clique, two things must happen. First, Step 1 needs to pick a good edge, that is, an edge in a 4-clique. Given there are $\Theta(n)$ 4-cliques, this happens with probability $\Theta(n/m)$. Next, given that Step 1 picks a good edge, Step 2 must pick a triangle, including the good edge, which itself is in a 4-clique. This happens with probability $\Theta(m/t)$. The success probability is therefore $\Theta(n/t)$. Using amplitude amplification the total cost of finding a 4-clique is therefore $\mathcal{O}(\sqrt{t})$.

By the coupon collector's problem, the total cost of finding all 4-cliques is $\widetilde{\mathcal{O}}(n\sqrt{t}) = \widetilde{\mathcal{O}}\left(|L|^{3/2}(|L(\mathbf{x}_1)||L(\mathbf{x}_1, \mathbf{x}_2)|)^{1/2}\right) \approx 2^{0.3418d+o(d)}$ using $2^{0.1724d+o(d)}$ memory. This also matches the complexity of Algorithm 2 in the $k = 4$ balanced setting.

C.2 The General k-clique Case for Unbalanced Configurations

The graph approach of Section 5 can also be applied to unbalanced configurations. Given the matrix C of a good configuration we form the following coloured, undirected graph, $G = (V, E)$. We have lists $L = L_1 = \cdots = L_k$ and vertices are elements of L. Let there be an edge of colour $c_{i,j}$ between vertices $\mathbf{x}_1$ and $\mathbf{x}_2$ if and only if $|\langle \mathbf{x}_1 , \mathbf{x}_2 \rangle - C_{i,j}| \leq \varepsilon$, with ε as before. Define oracles $O_{G_{i,j}} : V^2 \to \{\texttt{True}, \texttt{False}\}$ such that $O_{G_{i,j}}(\mathbf{x}_1, \mathbf{x}_2) = \texttt{True}$ if and only if there is an edge of colour $c_{i,j}$ between $\mathbf{x}_1, \mathbf{x}_2$. That is, we have $\binom{k}{2}$ coloured edges and

for each colour we have an oracle that checks whether there is an edge of this colour between two vertices.

In this setting we search for a coloured k-clique. That is, a set of vertices $\{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ such that there is an edge of colour $c_{i,j}$ between the vertices $\mathbf{x}_i$ and $\mathbf{x}_j$, for all $i \neq j$.

With small changes, we can apply the same k-clique finding algorithm in this setting. The algorithm is the following, with $2 \leq i \leq k-1$

1. Use Grover's algorithm to find an edge $(\mathbf{x}_1, \mathbf{x}_2)$ of colour $c_{1,2}$ among all potential $\mathcal{O}(|L|^2)$ edges of colour $c_{1,2}$.

   ⋮

i. Given a coloured i-clique $(\mathbf{x}_1, \ldots, \mathbf{x}_i)$, use Grover's algorithm to find a vertex $\mathbf{x}_{i+1} \in V$, such that $(\mathbf{x}_1, \mathbf{x}_{i+1})$, …, $(\mathbf{x}_i, \mathbf{x}_{i+1})$ are edges of colours $c_{1,i+1}$, …, $c_{i,i+1}$ respectively, to form a coloured $(i+1)$-clique $(\mathbf{x}_1, \ldots, \mathbf{x}_{i+1})$.

   ⋮

k. Apply amplitude amplification on steps $1-(k-1)$.

The complexity analysis is similar to the one in Section 5.2. The dominant cost of Steps $1-(k-1)$ is a Grover search over the $|L_i| = |L|$ vertices. What differs slightly is the calculation of the success probability.

Let us first look at the triangle case. An unbalanced configuration implies that the number of edges of colours $c_{1,2}$, $c_{1,3}$ and $c_{2,3}$ may be different. However, by having picked a good configuration and the right initial list size $|L|$ we still have $n = \Theta(|L|)$ triangles with one edge of colour $c_{1,2}$, one edge of colour $c_{1,3}$ and one edge of colour $c_{2,3}$. The total number of edges of colour $c_{1,2}$ is $m = \mathcal{O}(|L||L_2(\mathbf{x}_1)|)$. The probability that Steps 1–2 succeed is equal to the probability that Step 1 picks an edge of colour $c_{1,2}$, belonging to a coloured triangle. This probability is equal to $\Theta(n/m) = \Theta(1/|L_2(\mathbf{x}_1)|)$.

In the $k = 4$ case, by having picked a good configuration and the right initial list size $|L|$, even though the number of edges of different colours varies, the expected number of coloured 4-cliques is still $n = \Theta(|L|)$. The probability in Step 1 of picking a coloured edge belonging to a 4-clique is $\Theta(n/m) = \Theta(1/|L_2(\mathbf{x}_1)|)$. Given such an edge $(\mathbf{x}_1, \mathbf{x}_2)$, comparing with Figure 2b, we see that the number of coloured triangles $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$, such that $(\mathbf{x}_1, \mathbf{x}_2)$ is an edge of colour $c_{1,2}$, is equal to $\mathcal{O}(|L_3(\mathbf{x}_1, \mathbf{x}_2)|)$. Given that we have picked an edge belonging to a 4-clique, the number of these triangles belonging to a 4-clique is $\Theta(1)$. Thus, the total success probability for the $k = 4$ case is equal to $\Theta(1/(|L_2(\mathbf{x}_1)||L_3(\mathbf{x}_1, \mathbf{x}_2)|))$.

In the general case, in the first step the probability of finding an edge of colour $c_{1,2}$ belonging to a coloured k-clique is $\Theta(1/|L_2(\mathbf{x}_1)|)$. Next, assume that step $i-1$, for $2 \leq i \leq k-2$, has found a coloured i-clique $(\mathbf{x}_1, \ldots, \mathbf{x}_i)$ belonging to a coloured k-clique. The number of coloured $(i+1)$-cliques on the form $(\mathbf{x}_1, \ldots, \mathbf{x}_i, \mathbf{x}_{i+1})$ is $\Theta(\max(|L_{i+1}(\mathbf{x}_1, \ldots, \mathbf{x}_i)|, 1))$. This corresponds to the diagonal elements of Figure 2b. Of these $(i+1)$-cliques, $\Theta(1)$ belong to a coloured k-clique. Thus the probability of picking a coloured $(i+1)$-clique belonging to a coloured k-clique is $\Theta(1/\max(|L_{i+1}(\mathbf{x}_1, \ldots, \mathbf{x}_i)|, 1))$. The total

success probability is thus

$$\Theta\left(\left(\prod_{i=1}^{k-2}\max\left(|L_{i+1}(\mathbf{x}_1,\ldots,\mathbf{x}_i)|,1\right)\right)^{-1}\right).$$

By applying the amplitude amplification on Step k, we get the cost

$$\mathcal{O}\left(\sqrt{|L|}\sqrt{\prod_{i=1}^{k-2}\max\left(|L_{i+1}(\mathbf{x}_1,\ldots,\mathbf{x}_i)|,1\right)}\right),$$

for finding one k-clique. The total complexity of this algorithm is thus equal to

$$\widetilde{\mathcal{O}}\left(|L|^{3/2}\sqrt{\prod_{i=1}^{k-2}\max\left(|L_{i+1}(\mathbf{x}_1,\ldots,\mathbf{x}_i)|,1\right)}\right).$$

This matches the complexity of Algorithm 2 in the unbalanced setting, i.e. Eq. (9).

D Proofs of Lemmas from Section 7

Here we present the proofs of Lemmas 7.1, 7.2, 7.3 essentially giving a blueprint of the construction of the circuit implementing Algorithm 4. We will often describe unitaries (and reduced transformations) in detail, but equally often simply describe the operation and just claim there is some unitary that implements it. Particularly, when it is clear or can be derived by observation, such as when it can be shown there is an efficient classical reversible circuit implementing a function, using only Toffoli and NOT gates, one can as well construct an efficient quantum circuit to implement that. This allows us to focus on conveying the non-trivial and interesting aspects of the construction, rather than tiring an interested reader with trivialities such as presenting a unitary comprised of elementary gate sets that carries out matrix multiplication efficiently in the size of the input bitstrings. Lastly, we assume the setup as described earlier and crucially the fact that the circuit is prepared and initialised before the input, which consists of basis vectors, is received.

Proof D.1 (Proof of Lemma 7.1)**.** *We denote the input as a state $|B\rangle$. We begin by copying $|B\rangle$ into N processors. To do this, first define unitary $RC_{A,B}$ that implements a CNOT to registers A and B, with A being the control and B the target. Assume both registers are of same size.*

$$RC_{A,B} := CNOT_{A1,B1} \otimes CNOT_{A2,B2} \otimes CNOT_{A\bar{d},B\bar{d}}, \tag{14}$$

where Ai and Bi are the $i-th$ qubit of register A and B respectively. Notice thst all control and target qubits are different and can be composed in parallel. Thus the operator $RC_{A,B}$ can be applied applied with a depth-1 circuit. Operationally, a register of arbitrary size can be correlated with another register of (at least) the same size in 1 time step.

To copy $|B\rangle$ to all N processors a cascade of RC operations suffice. To be more precise, we first copy $|B\rangle$ to register $\mathtt{R}_1$ in processor $\mathtt{P}_1$. In the next time step copy $|B\rangle$ and $\mathtt{R}_1$ to registers $\mathtt{R}_2$ and $\mathtt{R}_3$ in processors $\mathtt{P}_2$, $\mathtt{P}_3$respectively. Continue the process for $\log(N)$ steps, where $|B\rangle, \mathtt{R}_1 \ldots \mathtt{R}_{N/2-1}$ are copied to $\mathtt{R}_{N/2}, \mathtt{R}_{N/2+1} \ldots \mathtt{R}_{N-1}$ registers of the last $N/2$ processors respectively (wlog, we assume N is even). Effectively, a $\log(N)$ depth circuit copies $|B\rangle$ to all N processors. The width of the circuit is $\widetilde{\mathcal{O}}(N)$, since each processor only needs a $\mathrm{poly}(d) = \mathrm{poly}\log N$ to store all the basis vectors.

Each processor $\mathtt{P}_i$ samples a vector $\mathbf{x}_i$ from the lattice $\mathcal{L}(B)$ using an efficient randomised sampling procedure. There are various ways to do it, we employ the standard technique due to Klein [Kle00]. It consists of sampling a vector $\mathbf{t}_i$ from $\mathbb{R}^d$ and running Babai's decoding algorithm to find a lattice point close to $\mathbf{t}_i$. Babai's algorithm can be implemented using a reversible circuit of depth $\mathrm{poly}(d)$. To sample $\mathbf{t}_i \in \mathbb{R}^d$ we first put some (hard wired) large enough bound C and produce a vector $\mathbf{t}_i \in [-C, C]^d$ coordinate-wise using the following process. We construct a $C+1$ dimensional maximally entangled state, $\frac{1}{\sqrt{C+1}} \sum_{j=0}^{C+1} |jj\rangle$ and discard the second of the two subsystems. Thus $\mathbf{t}_i = \mathrm{Tr}_2(\frac{1}{C+1} \sum_{i,j} |ii\rangle \langle jj|_{12})$. Operationally this would be equivalent to picking a random value less than $C+1$. For purposes of the circuit the first value picks the sign, and rest of it a number between 0 and C. This corresponds to picking one element of the vector $\mathbf{t}_i$. Repeating the same construction d times in parallel gives the complete vector $\mathbf{t}_i$.

It is efficient to construct a C-dimensional maximally entangled state as $\mathcal{O}(C)$ depth circuit suffices. Note that the requirement on the maximally entangled state and C are independent of the input vectors themselves, hence this can be considered as a resource from the initialisation phase. Thus a circuit that samples $\mathbf{x}_1, \ldots, \mathbf{x}_N$ needs $\mathcal{O}(\mathrm{poly}\log(N))$ depth and $\widetilde{\mathcal{O}}(N)$ width.

Finally to write each of these $\mathbf{x}_i$ into a distinguished memory cell L1*, the unitary*

$$RC_{\mathtt{R}_1,\mathtt{L1}[1]} \otimes RC_{\mathtt{R}_2,\mathtt{L1}[2]} \ldots \otimes RC_{\mathtt{R}_N,\mathtt{L1}[N]}$$

is applied and this is again of depth 1. Note that L1[0] *is the zero vector.*

Thus the list L_1 can be sampled efficiently, given an input $|B\rangle$, of $\mathrm{poly}(d)$ qubits, and the circuit that implements it, is of depth $\mathrm{poly}\log(N)$ and width $\widetilde{\mathcal{O}}(N)$. This concludes proof of Lemma 7.1.

Proof D.2 (Proof sketch of Lemma 7.2)**.** *With a new list L_1 that stores N vectors, we want to construct another list L_2, such that for any $i \in [N]$ it holds that $\mathbf{x}'_i \in L_2$ is such that $|\langle \mathbf{x}'_i \, , \mathbf{x}_i \rangle| \geq 1/2$ and $\mathbf{x}_i \in L_1$ (If no such $\mathbf{x}_i \in L_1$ exist for a given $\mathbf{x}_i$, then set $\mathbf{x}_i = 0$).*

We begin by defining a checking function $f(\mathbf{x}, \mathbf{y}) = 1$ if $|\langle \mathbf{x} \, , \mathbf{y} \rangle| \geq 1/2$ and 0 otherwise. The corresponding unitary is of the form

$$U_f : |\mathbf{x}\rangle \, |\mathbf{y}\rangle \, |b\rangle \to |\mathbf{x}\rangle \, |\mathbf{y}\rangle \, |b \oplus f(\mathbf{x}, \mathbf{y})\rangle \, .$$

First notice this unitary can be implemented efficiently in the size of the inputs $\mathbf{x}, \mathbf{y}$ since it only computes the inner product and already has an classically efficient algorithm.

Now for each processor i, define $f_i(\mathbf{y}) = 1$ if $|\langle \mathbf{x}_i \, , \mathbf{y} \rangle| \geq 1/2$ and 0 otherwise with the corresponding unitary

$$U_{f_i} : |\mathbf{y}\rangle \, |b\rangle \to |\mathbf{y}\rangle \, |b \oplus f(\mathbf{x}_i, \mathbf{y})\rangle \, .$$

It is important to stress that $\mathbf{x}_i$ is fixed for the f_i, thus the domain of f_i is only the set of d-dimensional vectors, whereas for f it is the set of d^2 dimensional vectors and the unitaries U_{f_i} are really reduced unitaries.

Next, in order to load the data on which parallel Grover's search is performed, we define the unitary

$$V : |j_1 \ldots j_N\rangle \, |\mathbf{y}_1 \ldots \mathbf{y}_N\rangle \, |\mathbf{x}_1 \ldots \mathbf{x}_N\rangle \to |j_1 \ldots j_N\rangle \, |\mathbf{y}_1 \oplus \mathbf{x}_1 \ldots \mathbf{y}_N \oplus x_N\rangle \, |\mathbf{x}_1 \ldots \mathbf{x}_N\rangle \, .$$

One of the results of [BBG+13] states that there is a uniform family of circuits of width $\mathcal{O}(N(d \cdot \log N))$ and depth $\mathcal{O}(\log N \cdot \log(d \cdot \log N))$ that implements V. We use this unitary together with U_{f_i} to construct

$$\mathcal{W} = V \circ U_{f_1} \otimes \ldots \otimes U_{f_N} \circ V.$$

The purpose of $\mathcal{W}$ is to load the database in the memory using V, compute f_i on each of the processor in parallel using U_{f_i}, and then unload the database using $V^\dagger = V$. The action of $\mathcal{W}$ on the initial state is

$$\begin{aligned}
&|j_1 \ldots j_N\rangle \, |0 \ldots 0\rangle \, |\mathbf{y}_1 \ldots \mathbf{y}_N\rangle \, |\mathbf{x}_1 \ldots \mathbf{x}_N\rangle \\
\xrightarrow{V} \; &|j_1 \ldots j_N\rangle \, |\mathbf{x}_{j_1} \ldots \mathbf{x}_{j_N}\rangle \, |\mathbf{y}_1 \ldots \mathbf{y}_N\rangle \, |\mathbf{x}_1 \ldots \mathbf{x}_N\rangle \\
\xrightarrow{\otimes U_{f_i}} \; &|j_1 \ldots j_N\rangle \, |\mathbf{x}_{j_1} \ldots \mathbf{x}_{j_N}\rangle \, |\mathbf{y}_1 \oplus f_1(\mathbf{x}_{j_1}) \ldots \mathbf{y}_N \oplus f_N(\mathbf{x}_{j_N})\rangle \, |\mathbf{x}_1 \ldots \mathbf{x}_N\rangle \\
\xrightarrow{V} \; &|j_1 \ldots j_N\rangle \, |0 \ldots 0\rangle \, |\mathbf{y}_1 \oplus f_1(\mathbf{x}_{j_1}) \ldots \mathbf{y}_N \oplus f_N(\mathbf{x}_{j_N})\rangle \, |\mathbf{x}_1 \ldots \mathbf{x}_N\rangle
\end{aligned}$$

Setting $|\mathbf{y}_i\rangle = |-\rangle$*, for all* i*, the reduced unitary becomes*

$$\mathcal{W} : |j_1 \dots j_N\rangle \to (-1)^{f_1(\mathbf{x}_{j_1})} |j_1\rangle \dots (-1)^{f_N(\mathbf{x}_{j_N})} |j_N\rangle .$$

The unitary $\mathcal{W}$ *can be implemented using a uniform family of circuits of width* $N \cdot (\log N + d) = N \cdot \mathrm{poly} \log N$ *and depth* $\mathcal{O}(\log N \log(d \log N) + \mathrm{poly}(d)) = \mathrm{poly} \log(N)$.

Define $|\Psi\rangle = \frac{1}{\sqrt{N}} \sum_i |i\rangle$*, and let* $R = (2 |\Psi\rangle \langle\Psi| - I)^{\otimes N}$*. Notice that* R *can be implemented using a* $\mathcal{O}(\log N)$ *depth circuit. Executing* $R \circ \mathcal{W}$ *(Grover's iteration)* $\Theta(\sqrt{N})$ *times on the initial state*

$$\sum_{j_1, \dots j_N} |j_1 \dots j_N\rangle |0 \dots 0\rangle |- \dots -\rangle |\mathbf{x}_1 \dots \mathbf{x}_N\rangle$$

leads to the sate (close to)

$$|J\rangle_4 |\mathbf{0}\rangle_3 |-\rangle_2 |X\rangle_1 ,$$

where $J = (j'_1 \dots j'_N)$*,* $X = \{\mathbf{x}_1 \dots \mathbf{x}_N\}$*, with the property that if the first block* $|J\rangle$ *measured, it would give a sample of indices,* $J = (j'_1 \dots j'_N)$ *such that* $\Theta(N)$ *of them would be 'good indices'. Namely, for (almost) all* j'_i*, we have that* $|\langle \mathbf{x}_{j'_i}, \mathbf{x}_i \rangle| \geq 1/2$ *for the* $\mathbf{x}_i$ *from the first block* $|X\rangle$ *. This is an instantiation of parallel quantum search over a single database and is the key ingredient of the algorithm. The circuit implementing this procedure is width and depth* $\widetilde{\mathcal{O}}(N)$ *and* $\widetilde{\mathcal{O}}(\sqrt{N})$ *respectively [BBG+13].*

The success probability of each processor $\mathtt{P}_i$ *to find* j'_i *is at least* $1 - \Theta(1/N)$ *(see Theorem 2.1 and the fact that we expect to have* $\Theta(1)$ *"good"* j_i*'s for* $\mathbf{x}_i$ *). We might as well repeat this procedure a constant number of times, and collect the set of indices* $J^0, \dots J^q$ *and check if there exits a* $j^l_k \in \{j^0_k, \dots j^q_k\}$ *such that* $|\langle \mathbf{x}_{j^l_k}, \mathbf{x}_k \rangle| \geq 1/2$ *(where* j^l_k *denotes the* k*-th register of* J^l*). Each processor can do this check in parallel, i.e. the* k^{th} *processor checks among the collected* q*-elements to find if there is a correct solution. If there is one, it copies, using the RC operator defined in Eq.* (14)*, this index to a distinguished register, called* $\mathtt{L2}[k]$ *corresponding to the* k^{th} *processor* $\mathtt{P}_k$*. If no solution is found,* $\mathtt{L2}[k]$ *is set to the zero vector. This repetition contributes with a small* $\mathrm{poly}(d)$ *overhead in depth and width. Now we have virtual list* L_3 *stored at the memory block* $\mathtt{L2}$*, whose* k^{th} *register holds the index of a "good" vector.*

To create the list L_3*, the crucial idea is to imagine a classical reversible subroutine, for each processor* k*, that first computes two vectors* $\mathbf{y}^+_k = \mathbf{x}_k + \mathbf{x}_{\mathtt{L2}[k]}$ *and* $\mathbf{y}^-_k = \mathbf{x}_k - \mathbf{x}_{\mathtt{L2}[k]}$*. Then if* $||\mathbf{y}^+_k|| < ||\mathbf{y}^-_k||$*, copy* $\mathbf{y}^+_k$ *to a designated register* $\mathtt{L3}[k]$ *otherwise copy* $\mathbf{y}^-_k$ *to* $\mathtt{L3}[k]$*. It can be verified that a circuit implementing this subroutine using only Toffoli and NOT gates has* $\mathrm{poly}(d)$ *depth and width.*

Finally to swap the labels of $\mathtt{L1}$ *and* $\mathtt{L3}$ *register blocks, apply a single swap gate* $CNOT_{i,j} \circ CNOT_{j,i} \circ CNOT_{i,j}$ *pairwise for each qubit pair, i.e., swap the first two qubits of the register block* $\mathtt{L1}$ *and* $\mathtt{L3}$*, then in parallel the second two qubits of* $\mathtt{L1}$*,* $\mathtt{L3}$ *and so on. The swap procedure is a depth 3 circuit.*

Finally we would want to free the memory from $\mathtt{L2}$ *and* $\mathtt{L3}$ *for later computations. The easiest way is to simply transfer the content (by using a SWAP gate like before) to auxiliary memory cells of the same size that were already initialised to state* $|\mathbf{0}\rangle$*.*

Thus we have shown Steps (3)-(5) of Algorithm 4 can be implemented with a circuit of depth $\widetilde{\mathcal{O}}(\sqrt{N})$ and width $\widetilde{\mathcal{O}}(N)$. This concludes the proof of Lemma 7.2.

Proof D.3 (Proof sketch of Lemma 7.3)**.** *We now want to find an element in the list L_1 with Euclidean norm less than a given threshold λ. We set $\lambda = \sqrt{d} \cdot \det(B)^{1/d}$ to satisfy Minowski's bound for $\mathcal{L}(B)$. This bound is tight up to a constant, which we do not know a priori. In order to avoid too many vectors from L_1 to satisfy the bound, we run several trials of Grover's algorithm starting with a small value for λ and re-iterating Grover's search with this value increased. We expect to terminate after $\mathcal{O}(1)$ iterations. The checking function for each Grover's search is $g(\mathbf{x}) = 1$ if $||\mathbf{x}|| < \lambda$, and 0 otherwise, and the corresponding unitary implementing the function $T : |\mathbf{x}\rangle |\mathbf{y}\rangle \rightarrow |\mathbf{x}\rangle |\mathbf{y} \oplus f(\mathbf{x})\rangle$. This admits efficient classical and quantum circuits of depth and width* $\mathrm{poly}(d)$. *As each Grover's iteration requires $\mathcal{O}(\sqrt{N})$ depth circuit, we show that Step 7 of Algorithm 4 needs $\widetilde{\mathcal{O}}(N)$ width (input) and $\widetilde{\mathcal{O}}(T\sqrt{N})$. It concludes the proof of Lemma 7.3.*

Paper IV

Some Cryptanalytic and Coding-Theoretic Applications of a Soft Stern Algorithm

Using the class of information set decoding algorithms is the best known way of decoding general codes, i.e. codes that admit no special structure, in the Hamming metric. The Stern algorithm is the origin of the most efficient algorithms in this class. We consider the same decoding problem but for a channel with soft information. We give a version of the Stern algorithm for a channel with soft information that includes some novel steps of ordering vectors in lists, based on reliability values. We demonstrate how the algorithm constitutes an improvement in some cryptographic and coding theoretic applications. We also indicate how to extend the algorithm to include multiple iterations and soft output values..

Keywords: Soft decoding, Stern algorithm, Side-channel attacks, Information set decoding, Soft Stern algorithm.

1 Introduction

For a general code with no special structure used for communication on the binary symmetric channel (BSC), the maximum-likelihood decoding problem (with some assumptions) is NP-hard. Still, decoding random linear codes is a central problem for many applications in cryptography, for example code-based crypto. Information set decoding (ISD) algorithms are the most promising candidates for solving instances of this problem.

The performance of these algorithms determines the security and hence the necessary parameters for many cryptosystems. The development of ISD algorithms include the Prange algorithm [23], the Lee-Brickell algorithm [18], the Stern algorithm [25], Canteaut-Chabaud [8], Ball-Collision Decoding [6], Finiasz-Sendrier [13], BJMM [3] and the recent improvement from May and Ozerov [19]. The Stern algorithm is the starting point for the most efficient algorithms in this class as it introduced a collision step that significantly decreased the complexity.

In this paper, we consider the decoding problem for a general code with no special structure used for communication on the Additive White Gaussian Noise (AWGN) channel using the Euclidean metric. This is motivated by the fact that we have seen some recent applications for such decoding algorithms in coding theory and cryptography. One such application is the recently proposed version of the McEliece Public Key Cryptosystem (PKC) using soft information [2]. Another is the use of such algorithms in side-channel cryptanalysis, see, e.g., [22]. A third one is a new hybrid decoding of low-density parity-check (LDPC) codes in space telecommand links [1].

The soft decoding problem has been studied extensively in coding and communication, see, e.g., [10, 17], but mostly for special codes allowing efficient decoding. The study of general codes has been less intense. Early work by Chase [9] was followed by some work in the communication direction and a highly cited paper is [14]. More recently, fast soft-decision decoding of linear codes was considered in [1, 11, 26, 28] and by Wu and Hadjicostis in [30]. The same problem considered in the context of side-channel cryptanalysis in cryptology can be found in [4, 5, 12].

In this paper, we give a version of the Stern algorithm for the decoding problem with soft information, named the soft Stern algorithm. The algorithm reuses some ideas from previous work, such as ordered statistics [14]. It uses the idea of sorting of error vectors in lists, based on reliability values [27], and presents a novel way of combining this idea with the structure of the Stern algorithm. This leads to better performance compared to previously suggested algorithms like the one in [22]. Initially we consider a one-pass algorithm that succeeds with some probability q. We can then repeat this one-pass algorithm to achieve a higher success probability, where the way it is repeated depends on the application. Later, we briefly consider extending the algorithm to also allow for multiple iterations.

Next, we demonstrate how this new algorithm can be used in cryptographic and coding theory applications. First, we present a very efficient attack on an idea of using soft information in McEliece-type cryptosystems presented at ISIT 2016 [2]. Not only do we severely break the proposed schemes, but our algorithm shows that the whole idea of using soft information in this way is not fruitful. Secondly, we show how our algorithm can be applied to side-channel attacks.

The problem of soft decoding of general codes appears in side-channel attacks in both [21] and [22]. Using our algorithm, both of those attacks can be improved. Thirdly, we show how our algorithm can be used to improve the hybrid decoding of low-density parity-check (LDPC) codes [1]. Finally, we indicate that by using soft output, our algorithm can be applied to the problem of decoding product codes.

The remaining parts of the paper are organized as follows. In Section 2 we give some preliminaries on coding theory and the considered channel. Section 3 gives an overview of the new algorithm, and in Section 4 we give a complete example of the algorithm. In Section 5 we analyze its time complexity and give simulation results demonstrating the improvement compared to previously proposed algorithms. In Section 6 we indicate how to generalize our algorithm by allowing for multiple iterations and soft output. In Section 7 we cover different applications of the algorithm. Finally, Section 8 concludes the paper.

2 Preliminaries

We present some basic concepts in coding theory. Let $\mathbb{F}_2$ denote the binary finite field, $|x|$ the absolute value of x for $x \in \mathbb{R}$, and $\ln(\cdot)$ the logarithm with base e. Let π be a permutation of $\{1, \ldots, n\}$ and π^{-1} be its inverse. For a matrix $\mathbf{G}$, we let $\pi(\mathbf{G})$ denote the matrix obtained from $\mathbf{G}$ by permuting its column indices according to π.

2.1 Basics in Coding Theory

Definition 2.1. *An $[n, k]$ binary linear code $\mathcal{C}$ is a k-dimensional vector subspace of $\mathbb{F}_2^n$. Its co-dimension is $r = n - k$, characterizing the redundancy of the code.*

A generator matrix $\mathbf{G}$ of the linear code $\mathcal{C}$ is defined as a $k \times n$ matrix in $\mathbb{F}_2^{k \times n}$ whose rows form a basis of the code. Equivalently, the code can be defined by a matrix $\mathbf{H}$ in $\mathbb{F}_2^{r \times n}$ whose kernel is the code $\mathcal{C}$, called a parity-check matrix of $\mathcal{C}$. For a length n vector $\mathbf{v}$, the support $\mathrm{supp}(\mathbf{v})$ is defined as $\{i : v_i \neq 0, 1 \leq i \leq n\}$. The Hamming weight of $\mathbf{v}$ is $w_H(\mathbf{v}) = |\{i : v_i \neq 0, 1 \leq i \leq n\}|$ and the Hamming distance is $d_H(\mathbf{v}, \mathbf{v}') = w_H(\mathbf{v} + \mathbf{v}')$.

Suppose an $[n, k]$ binary linear code $\mathcal{C}$ with generator matrix $\mathbf{G}$ is used for transmission on the AWGN channel. Let $\mathbf{c} = (c_1, c_2, \cdots, c_n)$ be a codeword to be transmitted. In Binary Phase-Shift Keying (BPSK) transmission, the codeword $\mathbf{c}$ is mapped to a bipolar sequence $\hat{\mathbf{c}} = (\hat{c}_1, \hat{c}_2, \cdots, \hat{c}_n)$, where $\hat{c}_i \in \mathbb{R}$ through $\hat{c}_i = (-1)^{c_i}$, for $1 \leq i \leq n$. For any binary vector $\mathbf{x}$, we use the notation $\hat{\mathbf{x}}$ to denote the result after applying the above mapping.

After transmission, where AWGN noise is added, the received vector is denoted $\mathbf{r} = (r_1, r_2, \cdots, r_n)$, $r_i \in \mathbb{R}$ for $1 \leq i \leq n$, where $r_i = \hat{c}_i + w_i$ and w_i are iid Gaussian random variables with zero mean and standard deviation σ. Since the values are floating-point, we say that we have soft information. If the noise would be binary, we would have worked with hard information. If we would translate each value of the $\mathbf{r}$ vector to its most probably binary value in $\mathbf{c}$, we we would make a so called hard decision.

In the continuation, when discussing the reliability value of a position we refer to r_i. When discussing how reliable a position is we refer to the absolute value of r_i.

Our soft-decision decoding problem is now the following: Find the most likely codeword being transmitted when receiving $\mathbf{r}$. We consider maximum-likelihood decoding (MLD). It is well known that the MLD metric becomes the squared Euclidean distance and that the codeword $\mathbf{c}$ closest to a received vector $\mathbf{r}$ is the one that minimizes the distance $D(\hat{\mathbf{c}}, \mathbf{r}) = \sum_{i=1}^{n}(r_i - \hat{c}_i)^2$ (see, e.g., [29]).

For binary codes, it is common to use the log likelihood ratio (LLR), which is defined as

$$L_i = \ln\left[\frac{p(r_i|c_i = 0)}{p(r_i|c_i = 1)}\right],$$

where $p(r_i|c_i)$ is the pdf of r_i conditioned on c_i. After some calculations one can rewrite this for the AWGN channel as

$$L_i = \frac{2r_i}{\sigma^2}.$$

We point out that we actually only need soft information in LLR form and the algorithm to be proposed works for any noise distribution, not just AWGN.

Finally, we introduce a class of codes for later use.

Definition 2.2. *A low density parity-check (LDPC) code is a linear code admitting a sparse parity-check matrix, while a moderate density parity-check (MDPC) code is a linear code with a denser but still sparse parity-check matrix.*

In previous work, the Hamming weight of the row vector is usually employed to characterize its sparsity; LDPC codes have small constant row weights, MDPC codes have row weights $O(\sqrt{n \log n})$. These classes of codes are of interest since they are efficiently decodable using iterative decoding techniques exploiting the sparsity of the codes.

The class of quasi-cyclic MDPC (QC-MDPC) codes are of special interest as they are used in the QC-MDPC code-based McEliece cryptosystem [20]. The codes used in the cryptosystem are linear codes with sparse parity-check matrices of the form,

$$\mathbf{H} = (\mathbf{H}_0 \ \mathbf{H}_1 \ \ldots \ \mathbf{H}_{n_0-1}), \tag{1}$$

where n_0 is a small integer and each block $\mathbf{H}_i$, $0 \leq i \leq n_0 - 1$, is a circulant matrix with size $r \times r$, and $\mathbf{H}_{n_0-1}$ is invertible. For simplicity, we assume that $n_0 = 2$ throughout the paper, unless otherwise specified. Thus, we consider codes with rate $R = 1/2$, length $n = 2r$, and dimension $k = r$.

2.2 Soft McEliece

Soft McEliece [2] is a recent code-based McEliece PKC proposal using soft information. Instead of generating intentional errors from a Hamming ball, the authors generate noise according to a Gaussian distribution. In the key generation, as in the QC-MDPC scheme, they generate a sparse parity-check matrix $\mathbf{H}$ with the form of (1) and use it as the secret key. The public key can be derived as the (dense) generator matrix $\mathbf{G}$ in systematic form corresponding to $\mathbf{H}$.

Algorithm 1 The Stern algorithm

Input: Generator matrix $\mathbf{G}$, parameters p, l

1. Choose a column permutation π and form $\pi(\mathbf{G})$, meaning that the columns in $\mathbf{G}$ are permuted according to π.
2. Bring the generator matrix $\pi(\mathbf{G})$ to systematic form:
$$\mathbf{G}^* = (\mathbf{I}\ \mathbf{Q}\ \mathbf{J})\,.$$
3. Let $\mathbf{z}$ run through all weight p vectors of length $k/2$. Store all vectors $\mathbf{x} = (\mathbf{z}, \mathbf{0})\mathbf{G}^*$ in a sorted list $\mathcal{L}_1$, sorted according to $\phi(\mathbf{x})$. Then construct a list $\mathcal{L}_2$ sorted according to $\phi(\mathbf{x})$, containing all vectors $\mathbf{x} = (\mathbf{0}, \mathbf{z})\mathbf{G}^*$ where $\mathbf{z}$ runs through all weight p vectors. Add all pairs of vectors $\mathbf{x} \in \mathcal{L}_1$ and $\mathbf{x}' \in \mathcal{L}_2$ for which $\phi(\mathbf{x}) = \phi(\mathbf{x}')$ and put in a list $\mathcal{L}_3$.
4. For each $\mathbf{x} \in \mathcal{L}_3$, check if the weight of $\mathbf{x}$ is $w - 2p$. If no such codeword is found, return to 1.

Given a message $\mathbf{u} \in \mathbb{F}_2^k$, let $\mathbf{c} = \mathbf{uG}$ be the encoded codeword and $\hat{\mathbf{c}}$ the codeword in $\mathbb{R}^n$. The ciphertext is

$$\mathbf{r} = \hat{\mathbf{c}} + \mathbf{w},$$

where $\mathbf{w} = (w_1, w_2, \ldots, w_n)$ and w_i $(1 \leq i \leq n)$ is AWGN. The generation of $\mathbf{w}$ is repeated until the number of bit errors in $\mathbf{r}$ reaches a certain threshold. The decryption – decoding the received vector – can be performed using an iterative soft LDPC/MDPC decoder that uses the secret $\mathbf{H}$, see [2, 20].

Parameter settings

In [2], the authors suggested parameters
$(n, \sigma) = (7202, 0.44091)$ for 80-bit security, and $(15770, 0.41897)$ for 128-bit security. The complexity of decoding a received vector $\mathbf{r}$ knowing only the public generator matrix $\mathbf{G}$ must be larger than 2^{80} and 2^{128}, respectively.

2.3 The Stern Algorithm

The Stern algorithm finds a low weight codeword of Hamming weight w in the code described by $\mathbf{G}$. Transform the generator matrix $\mathbf{G}$ to systematic form with generator matrix

$$\mathbf{G}^* = (\mathbf{I}\ \mathbf{Q}\ \mathbf{J})\,,$$

where $\mathbf{I}$ is the $k \times k$ identity matrix, $\mathbf{Q}$ is a $k \times l$ matrix and $\mathbf{J}$ is a $k \times (n-k-l)$ matrix. Let $\phi(\mathbf{x})$ be the value of a vector $\mathbf{x}$ in positions $k+1$ to $k+l$, i.e. $\phi(\mathbf{x}) = (x_{k+1}, x_{k+2}, \cdots, x_{k+l})$. The algorithm description is given in Algorithm 1.

3 A Soft Version of the Stern Algorithm

We now present as the main contribution a version of the Stern algorithm that uses soft information.

3.1 A One-Pass Soft Stern Algorithm

Receiving the vector $\mathbf{r}$, one can obtain a binary vector by making bitwise hard decisions. We define

$$\operatorname{sgn}(r_i) = \begin{cases} 1, & \text{if } r_i \leq 0, \\ 0, & \text{otherwise.} \end{cases}$$

Assuming that c_i is uniformly distributed over $\mathbb{F}_2$, according to Bayes' law, the conditional probability $\mathbf{Pr}\left[c_i = \operatorname{sgn}(r_i)|r_i\right]$, denoted p_i, can be written as

$$p_i = \frac{1}{1 + e^{-|L_i|}}. \tag{2}$$

Also, define the bias as $\tau_i = |p_i - 1/2|$. The problem of recovering the message from a ciphertext is solved by finding a minimum-weight codeword from a linear code with a generator matrix

$$\begin{pmatrix} \mathbf{G} \\ \operatorname{sgn}(r_1), \operatorname{sgn}(r_2), \ldots, \operatorname{sgn}(r_n) \end{pmatrix}.$$

This would, however, give a poor performance compared to what can be achieved when we use the soft information. Instead, we suggest to use the Stern algorithm as a basis and to modify the different steps to make use of the soft information in the best possible way. Initially, we consider only a single round in this algorithm, which will give a (small) probability q of success. In many (cryptographic) applications this is sufficient as one might repeat the decoding attempt many times and thus achieve an expected complexity which is a factor $1/q$ larger than the complexity of a single round. Later on, in Section 6.2, we indicate how to extend the algorithm to allow for multiple iterations.

The new algorithm can be divided into three steps in the following way:

Transformation

This step performs a column permutation and some transformations. Instead of selecting a random column permutation as in the original Stern algorithm, we consider only a single round and we use a permutation that puts the most reliable positions as the $k + l$ first columns. These columns will correspond to the information set and l additional positions.

Firstly, all the n coordinates r_i are sorted according to the absolute value of their LLR and then we choose a set $\mathcal{S}$ containing the $k + l$ most significant coordinates. Denote the set containing the other positions by $\mathcal{O}$. We use π to denote a permutation such that $\pi(\mathcal{S}) = \{1, \ldots, k + l\}$.

The second condition on π is that the first k columns of $\pi(\mathbf{G})$ are independent, forming a basis. We then derive a systematic generator matrix $\mathbf{G}^*$ from $\mathbf{G}$ by permuting the columns using π and performing Gaussian elimination, giving

$$\mathbf{G}^* = (\mathbf{I}\ \mathbf{Q}\ \mathbf{J}),$$

where $\mathbf{Q}$ is a $k \times l$ matrix. The received vector $\mathbf{r}$ is permuted accordingly, giving vector $\pi(\mathbf{r})$. The k first positions are now an information set, denoted $\mathcal{I}$.

We next perform a transformation to ensure that the reliability value for each variable in the information set is positive. We first determine the most likely value for the variables in the information set, denoted by $\mathbf{m}$, where $m_i = \text{sgn}(r_{\pi^{-1}(i)})$, for $1 \leq i \leq k$. This $\mathbf{m}$ corresponds to the codeword $\mathbf{c}' = \mathbf{m}\mathbf{G}^*$. Then the vector $\pi(\mathbf{r})$ is transformed to the vector $\mathbf{r}' = (r'_1, \ldots, r'_n)$, where

$$r'_i = r_{\pi^{-1}(i)} \cdot (-1)^{c'_i}, \quad 1 \leq i \leq n. \tag{3}$$

We have the following proposition.

Proposition 3.1. *If $D(\hat{\mathbf{c}}, \mathbf{r}) = \delta$, then $D(\hat{\mathbf{c}''}, \mathbf{r}') = \delta$, where $\mathbf{c}'' = \mathbf{c} + \mathbf{c}'$.*

Therefore, the transformation has not changed the problem, but the first k positions now all have positive reliability, which may ease the description in the continuation.

For the next step, we will consider the shortened code from $(\mathbf{I}\ \mathbf{Q})$ and try to find a list of codeword candidates close to $\mathbf{r}'$ in the first $k+l$ positions. For columns with indices in $\{k+1, \ldots, k+l\}$ corresponding to the matrix $\mathbf{Q}$ in $\mathbf{G}^*$, we determine a syndrome $\mathbf{s}$ by $s_i = \text{sgn}(r'_{k+i})$, for $1 \leq i \leq l$.

Codewords for the shortened code are vectors $\mathbf{c}^{(\text{S})}$ such that $\mathbf{c}^{(\text{S})}\mathbf{H}' = \mathbf{0}$, where $\mathbf{H}' = \begin{pmatrix} \mathbf{Q} \\ \mathbf{I}_l \end{pmatrix}$. As we change the signs of position $k+1, \ldots, k+l$ to be all positive when we introduced the syndrome, our problem is finding the most probable low weight vectors $\mathbf{z}$ such that $\mathbf{z}\mathbf{H}' = \mathbf{s}$, assuming that the reliability values in position $1, \ldots, k+l$ are all positive, i.e., assuming $r'_i \geq 0$, for $1 \leq i \leq k+l$.

We next partition the set $\pi(\mathcal{S}) = \{1, \ldots, k+l\}$ into two disjoint equal-sized parts, $\mathcal{S}_1$ and $\mathcal{S}_2$, such that

$$\prod_{i \in \mathcal{S}_1} p_i \approx \prod_{j \in \mathcal{S}_2} p_j,$$

where $p_i = \mathbf{Pr}\left[c_i^{(\text{S})} = 0 | r'_i\right]$ as in (2). For simplicity, we assume that $\mathcal{S}_1 = \{1, \ldots, (k+l)/2\}$ and $\mathcal{S}_2 = \{(k+l)/2+1, \ldots, (k+l)\}$. In the algorithm, this is yet another condition to consider when selecting π. In the original Stern algorithm the choice of indices for the two sets does not influence the performance, but for the soft case it does, and this is the reason for the above condition.

Creating Bit Patterns and Partial Syndromes

We now create the most probable (low weight error words) $\mathbf{z}^{(1)}$ having nonzero values only in $\mathcal{S}_1$. We store the corresponding partial syndrome for the code with transposed parity check matrix $\mathbf{H}'$, created as $(\mathbf{z}^{(1)}, \mathbf{0})\mathbf{H}'$. As all reliability values are positive, the zero word is the most likely one, then different vectors of weight one, etc. Let $\mathbf{z}^{(1)}$ run through T length-$(k+l)/2$ binary vectors with the largest probability $\prod_{i \in \mathcal{S}_1} \mathbf{Pr}\left[c_i^{(\text{S})} = z_i^{(1)} | r'_i\right]$. We build a table $\mathcal{L}_1$ to store all selected $\mathbf{z}^{(1)}$ together with the vector $(\mathbf{z}^{(1)}, \mathbf{0})\mathbf{H}'$. The table $\mathcal{L}_1$ is sorted according to this partial syndrome.

We now repeat the same thing but for the subset $\mathcal{S}_2$, creating another table $\mathcal{L}_2$ in a similar manner. In this case we run through the most probable vectors $\mathbf{z}^{(2)}$ with nonzero positions only in $\mathcal{S}_2$. Each entry in the table consists of $\mathbf{z}^{(2)}$ together with the partial syndrome $\mathbf{s}+(\mathbf{0},\mathbf{z}^{(\mathbf{2})})\mathbf{H}'$ sorted according to the latter. Note that we add $\mathbf{s}$ in this case.

Colliding Partial Syndromes

Next, we search for partial syndrome collisions between the tables $\mathcal{L}_1$ and $\mathcal{L}_2$. On average we obtain $T^2/2^l$ colliding vectors. Later we assume that we choose $T \approx 2^l$ to minimize time-complexity.

For each collision, we add the corresponding vectors $(\mathbf{z}^{(1)},\mathbf{0})$ and $(\mathbf{0},\mathbf{z}^{(\mathbf{2})})$, and create a new vector $\mathbf{u}$ by choosing its first k entries. Then we get a candidate codeword $\mathbf{uG}^*$. As a final step, we check the Euclidean distance between each candidate codeword and the received vector $\mathbf{r}'$. If it is sufficiently small we return it as the desired closest codeword.

3.2 How to Create the Most Probable Vectors

In this part, we explain how to create the T most probable vectors required in Step 2 of the previous description.

Since the reliability values are all transformed to be positive, for the partitions $\mathcal{S}_m$, $m = 1, 2$, the most probable pattern is the all-zero vector $\mathbf{0}$, with probability $P_m = \prod_{i\in\mathcal{S}_m} p_i$. The probability for a pattern with ones in positions in $\mathcal{J}$ and the remaining positions all zero is $\exp(-\sum_{j\in\mathcal{J}} L_j)\cdot P_m$.

For $\mathcal{S}_1$, our problem can then be described as follows. Given $(k+l)/2$ positive real numbers $(L_1,\ldots,L_{(k+l)/2})$ which are sorted in increasing order, i.e., $L_1 \le L_2 \le \ldots \le L_{(k+l)/2}$, define a function $f(\mathcal{I}) = \sum_{j\in\mathcal{I}} L_j$, where $\mathcal{I} \subset \{1,\ldots,\frac{(k+l)}{2}\}$. Our goal is to find the T different index sets $\mathcal{I}_i, 1 \le i \le (k+l)/2$ with smallest corresponding values $f(\mathcal{I}_i)$. The method for solving this problem is based on [27].

Let $I_{i_1,i_2,\ldots,i_k}$, where $i_1 < i_2 < \cdots i_k$, denote the bit pattern with the value 1 in positions $i_1, i_2, \ldots, i_k$, and the value zero in the other positions. For such a bit pattern, its function value is again $f(\mathcal{I}) = \sum_{j\in\mathcal{I}} L_j$, where $\mathcal{I} = \{i_1, i_2, \ldots, i_k\}$.

Now, let $\mathcal{R}_i$ denote the set of bit patterns with a 1 in position i and zeros in all positions after i. Sort the elements in $\mathcal{R}_i$ by its function values in increasing order to form the list R_i. Given a pattern $I \in R_i$, by the successor of I we mean the next pattern in R_i.

To solve our original problem we use a binary tree, where each node represents one of the lists $R_2, R_3, \ldots, R_{(k+l)/2}$. Initially, let the nodes store the top element in each list R_i, being the patterns $I_2, I_3, \ldots, I_{(k+l)/2}$, respectively. Also, let each node store an index value 0. The root of the tree will have the pattern with the smallest function value, which initially is I_2. Each parent node in the tree has a smaller function value than its child nodes.

Let A denote a list of the bit patterns we have found sofar, and their corresponding function values. Initialize this list with the all zeros pattern at index 0 and the pattern with a 1 in the first position at the index 1.

In each step of our algorithm we add the pattern of the root node to A. Assume the root node has the pattern $I_{i_1,i_2,\ldots,i_m,i}$. Then we replace the node label by the next pattern in the list R_i. This is found by starting in A at the

Algorithm 2 The Soft Stern algorithm

Input: Generator matrix $\mathbf{G}$, received vector $\mathbf{r}$, parameters $T = 2^l, \delta$

Step 1

(1a) Choose a column permutation π such that 1) the first $k+l$ positions in $\pi(\mathbf{G})$ have the $k+l$ largest $|r_i|$'s and 2) the first k columns are independent and 3) $\prod_{i=1,2,\ldots(k+l)/2} p_i \approx \prod_{i=(k+l)/2+1,\ldots(k+l)} p_i$.

(1b) Make the permuted generator matrix $\pi(\mathbf{G})$ systematic:

$$\mathbf{G}^* = (\mathbf{I}\ \mathbf{Q}\ \mathbf{J})\,.$$

Permute and transform the received sequence $\mathbf{r}$ to make the reliability value for each coordinate in positions $1, 2, \ldots k$ positive, following (3), giving $\mathbf{r}'$.

(1c) Calculate the corresponding partial syndrome $\mathbf{s}$ and change the sign of any negative values of $r'_{k+1}, \ldots r'_{k+l}$.

Step 2 Construct a list $\mathcal{L}_1$ storing the most probable vectors $\mathbf{z^{(1)}}$ and the corresponding partial syndromes $(\mathbf{z^{(1)}}, \mathbf{0})\mathbf{H}'$. Then construct another list $\mathcal{L}_2$ storing the most probable vectors $\mathbf{z^{(2)}}$ and the corresponding partial syndromes $\mathbf{s} + (\mathbf{0}, \mathbf{z^{(2)}})\mathbf{H}'$.

Step 3 Sort the two lists according to their partial syndromes and search for collisions. For each colliding syndrome $(\mathbf{z^{(1)}}, \mathbf{0})\mathbf{H}'$ and $\mathbf{s}+(\mathbf{0}, \mathbf{z^{(2)}})\mathbf{H}'$, create a new vector $\mathbf{u}$ by choosing the first k entries of $(\mathbf{z^{(1)}}, \mathbf{z^{(2)}})$ and compute the corresponding $\hat{\mathbf{c}} = (\hat{c}_1, \ldots, \hat{c}_n)$, s.t. $\hat{c}_i = (-1)^{c_i}$, where $\mathbf{c} = \mathbf{u}\mathbf{G}^*$.

If $D(\hat{\mathbf{c}}, \mathbf{r}') \leq \delta$, invert the transformations to get the codeword close to the original $\mathbf{r}$ and return it. If no $\mathbf{c}$ with $D(\hat{\mathbf{c}}, \mathbf{r}') \leq \delta$ is found, return failure.

index of the pattern $I_{i_1,i_2,\ldots,i_m}$ and finding the next pattern $I_{j_1,j_2,\ldots,j_n,i}$ in A such that $j_n < i$. If no such pattern exists, we have used all patterns in R_i and we can delete the node from the tree. Otherwise, we replace the node label by the pattern $I_{j_1,j_2,\ldots,j_n,i}$ and we also store the index in A of the pattern $I_{j_1,j_2,\ldots,j_n}$. In either case, we end by rearranging the tree such that each parent node has a smaller function value than its child nodes.

The most expensive part of the algorithm is rearranging the tree. This requires at most $\lceil \log_2(k+l)/2 \rceil$ function comparisons. If we store the function value for each pattern in A, calculating the function value for a new pattern in the tree only requires a single addition.

Example of How to Find the Most Probable Vectors

An example of how to find the most probable bit patterns is illustrated in Figure 1. In this case we work with vectors of length 8 and the corresponding 8 real values are

$$[0.1622, 0.1656, 0.2630, 0.3112, 0.5285, 0.6020, 0.6541, 0.7943].$$

For the sake of clarity we work with the whole bit patterns, but storing only the indices of the positions with the value 1 is of course more efficient. At the beginning the list $A = [(00000000, 0), (10000000, 0.1622)]$. In each step we add the root node and its corresponding function value to A. We use the index of the root node to determine where in A we start looking for a new bit pattern. When we have found the next pattern we modify the root node and rearrange the tree. Notice that the bit pattern 11000000 does not have a successor node. Therefore, after adding the pattern to A the corresponding node in the tree is removed. By the time we have reached the sixth binary tree in Figure 1 we have

$$\begin{aligned} A = [&(00000000, 0.0000), (10000000, 0.1622), (01000000, 0.1656), (00100000, 0.2630), \\ &(00010000, 0.3112), (11000000, 0.3278), (10100000, 0.4252)]. \end{aligned}$$

The bit patterns in A gives us the 7 most probable bit patterns. By looking at the root node we see that pattern number 8 is 01100000.

4 A Decoding Example

This section contains a complete example of how a message is encoded, how Gaussian noise is added, how the errors are corrected using the proposed Soft Stern algorithm, and finally how the original message is recovered. The extended, binary Golay code is a linear, systematic, error-correcting code with parameters $(n, k, d_{min}) = (24, 12, 8)$ and generator matrix $\mathbf{G}$ equal to

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

Assume sending the message $\mathbf{u} = [0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 0]$, transforming each 1 to -1 and each 0 to 1, and adding Gaussian noise with $\sigma = 1$. In this example the received vector $\mathbf{r}$ is

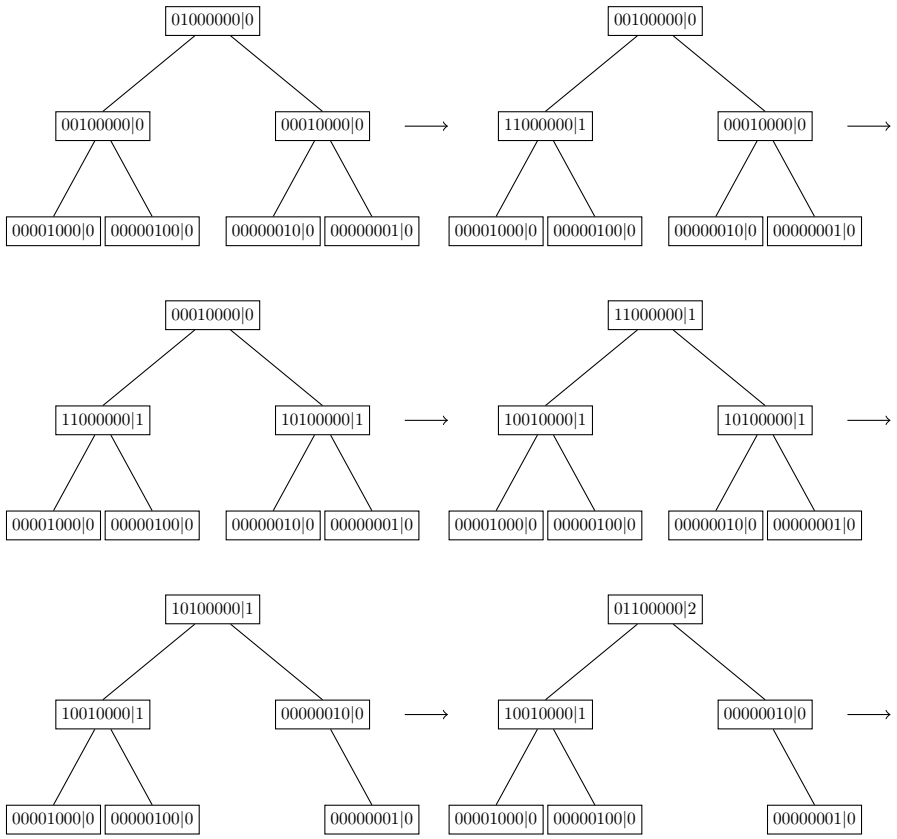


Figure 1: An ilustration of what the binary tree in the algorithm for finding the most probable bit patterns looks like in the first six steps.

$$
\begin{matrix}
[0.8437 & -0.8059 & 1.5800 & -0.1491 & 0.5741 & 0.6922 & 1.0734 & -1.9306 \\
-1.5678 & 1.1405 & 0.8998 & 2.6440 & 1.2390 & -0.6847 & 0.0724 & -0.2315 \\
0.1800 & -0.8032 & -1.3533 & -2.8248 & 0.1319 & 0.0888 & -1.2301 & -0.3469].
\end{matrix}
$$

Let us use $l = 4$ and $T = 2^l = 2^4 = 16$. After performing a permutation of the positions such that the first $k + l$ are the most reliable, such that the first k columns in the generator matrix are linearly independent, and such that the first $k + l$ positions are split into two parts with approximately equal products of p_i values, we obtain an $\mathbf{r}^*$ vector equal to

$$
\begin{matrix}
[-1.2301 & 0.6922 & 0.8437 & -1.3533 & 1.1405 & 1.0734 & 2.6440 & -0.6847 \\
-1.5678 & 1.5800 & 0.8998 & -0.8059 & -2.8248 & -1.9306 & 1.2390 & -0.8032 \\
0.5741 & -0.3469 & -0.2315 & 0.1800 & -0.1491 & 0.1319 & 0.0888 & 0.0724].
\end{matrix}
$$

The corresponding systematic generator matrix $\mathbf{G}^*$ is

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

Encoding the message $\mathbf{m}$, where each of the k positions of $\mathbf{m}$ are 1 if the corresponding position in $\mathbf{r}^*$ is positive and 0 otherwise, then changing the sign for each position in $\mathbf{r}^*$ where the corresponding position in the encoded vector $\mathbf{mG}^*$ is positive, results in an $\mathbf{r}'$ vector equal to

$$\begin{array}{[cccccccc}
1.2301 & 0.6922 & 0.8437 & 1.3533 & 1.1405 & 1.0734 & 2.6440 & 0.6847 \\
1.5678 & 1.5800 & 0.8998 & 0.8059 & 2.8248 & 1.9306 & -1.2390 & -0.8032 \\
0.5741 & 0.3469 & 0.2315 & 0.1800 & -0.1491 & -0.1319 & 0.0888 & -0.0724]
\end{array}.$$

We notice that the partial syndrome is $\mathbf{s} = [0\ 0\ 1\ 1]$ (by looking at the signs of positions $k+1$ to $k+l$). Picking the first $k+l$ values of $\mathbf{r}'$, switching signs to make all the values positive, calculating the corresponding LLR values, and sorting the LLR values such that each half of the values are in increasing order, gives a vector of LLR values equal to

$$\begin{array}{[cccccccc}
1.3694 & 1.3844 & 1.6875 & 2.1468 & 2.2811 & 2.4602 & 2.7066 & 5.2879 \\
1.6063 & 1.6119 & 1.7996 & 2.4779 & 3.1356 & 3.1600 & 3.8611 & 5.6495]
\end{array}.$$

Picking the parity-check matrix corresponding to the first $k+l$ columns of $\mathbf{G}^*$, then permuting the columns according to the permutation done to sort the LLR values, results in the permuted parity-check matrix $\mathbf{H}$ equal to

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

Using the first half of the LLR values to create the T most probable vectors on the form $(\mathbf{z}_1\ 0)$ and their corresponding syndromes $(\mathbf{z}_1\ \mathbf{0})\mathbf{H}^T$ we get the following list of vectors and syndromes

$$\mathbf{L}_1 = \left[\begin{array}{cccccccc|cccc} 0&0&0&0&0&0&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&1&0&0&0\\ 0&1&0&0&0&0&0&0&0&1&1&0\\ 0&0&1&0&0&0&0&0&1&1&1&1\\ 0&0&0&1&0&0&0&0&0&0&0&1\\ 0&0&0&0&1&0&0&0&1&0&1&1\\ 0&0&0&0&0&1&0&0&0&0&1&0\\ 0&0&0&0&0&0&1&0&1&1&0&0\\ 1&1&0&0&0&0&0&0&1&1&1&0\\ 1&0&1&0&0&0&0&0&0&1&1&1\\ 0&1&1&0&0&0&0&0&1&0&0&1\\ 1&0&0&1&0&0&0&0&1&0&0&1\\ 0&1&0&1&0&0&0&0&0&1&1&1\\ 1&0&0&0&1&0&0&0&0&0&1&1\\ 0&1&0&0&1&0&0&0&1&1&0&1\\ 1&0&0&0&0&1&0&0&1&0&1&0 \end{array}\right].$$

Using the last half of the LLR values to create the T most probable vectors on the form $(\mathbf{0}\ \mathbf{z}_2)$ and their corresponding syndromes $(\mathbf{0}\ \mathbf{z}_2)\mathbf{H}^T + \mathbf{s}$ we get the following list of vectors and syndromes

$$\mathbf{L}_2 = \left[\begin{array}{cccccccc|cccc} 0&0&0&0&0&0&0&0&0&0&1&1\\ 1&0&0&0&0&0&0&0&0&0&1&0\\ 0&1&0&0&0&0&0&0&0&1&1&0\\ 0&0&1&0&0&0&0&0&1&1&0&1\\ 0&0&0&1&0&0&0&0&0&0&0&1\\ 0&0&0&0&1&0&0&0&1&1&1&0\\ 0&0&0&0&0&1&0&0&1&0&0&0\\ 1&1&0&0&0&0&0&0&0&1&1&1\\ 1&0&1&0&0&0&0&0&1&1&0&0\\ 0&1&1&0&0&0&0&0&1&0&0&0\\ 0&0&0&0&0&0&1&0&0&1&1&1\\ 1&0&0&1&0&0&0&0&0&0&0&0\\ 0&1&0&1&0&0&0&0&0&1&0&0\\ 0&0&1&1&0&0&0&0&1&1&1&1\\ 1&0&0&0&1&0&0&0&1&1&1&1\\ 0&1&0&0&1&0&0&0&1&0&1&1 \end{array}\right].$$

Colliding these vectors we get the following list of possible candidates for a solution (where the first half of each row corresponds to $\mathbf{z}_1$ and the second half corresponds to $\mathbf{z}_2$)

$$\left[\begin{array}{cccccccc|cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0
\end{array}\right].$$

For each candidate we invert the permutation corresponding to the sorting of the LLR values. Then we pick the k first bits and create the message $\mathbf{u}_0$. We then encode the message using $\mathbf{G}^*$ to $\mathbf{c}_0 = \mathbf{u}_0\mathbf{G}^*$ and transform each 1 to -1 and each 0 to 1, creating $\hat{\mathbf{c}_0}$. Then we calculate the Euclidean distance between $\hat{\mathbf{c}_0}$ and $\mathbf{r}'$. The vector $\mathbf{c}_0$ that will lead us to the original message is probably the one with the smallest Euclidean distance. In this example the smallest Euclidean norm corresponds to candidate number 4.

Inverting the sorting step and picking the k first bits gives us

$$\mathbf{u}_0 = [0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1].$$

We then get

$$\mathbf{c}_0 = [0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1].$$

To get back to the solution to the original problem we flip the bits in $\mathbf{c}_0$ where the corresponding positions in $\mathbf{r}'$ and $\mathbf{r}^*$ differ in sign and get the vector

$$[1\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 0].$$

Then we invert the first transformation and get the vector

$$[0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1].$$

Now we notice that the first k positions in this vector are identical to the original message $\mathbf{u}$ and we have thus found the original message.

5 Complexity Analysis and Simulations

A suitable complexity measure is given by $C_{\text{one-pass}}/\mathbf{Pr}\,[\mathcal{A}]$, where $C_{\text{one-pass}}$ is the complexity of one pass of the algorithm and $\mathcal{A}$ represents the event that after the permutation and the transformation, the actual error pattern in the

first $(k+l)$ positions is a summation of two vectors in the two lists, respectively (i.e., that the Soft Stern algorithm will find the correct message).

When estimating complexity for matrix operations, we note that we can inductively implement the vector/matrix multiplication of $\mathbf{vM}$ by adding a new vector to an already computed and stored vector $\tilde{\mathbf{v}}\mathbf{M}$, where $\mathrm{supp}(\tilde{\mathbf{v}}) \subset \mathrm{supp}(\mathbf{v})$ and $d_H(\tilde{\mathbf{v}}, \mathbf{v}) = 1$. Thus, $C_{\text{one-pass}}$ measured in simple bit-oriented operations is roughly given by $C_{\text{Gauss}} + 4T \cdot (n-k) + C_{\text{create}}$, where C_{Gauss} is the complexity of Gaussian elimination that usually equals $0.5nk^2$ and C_{create} is the complexity for creating these most probable vectors. From Section 3.2 we have that $C_{\text{create}} = 2T\lceil \log_2((k+l)/2) \rceil$. Notice that the cost of creating the lists is low compared to calculating the partial syndromes and colliding these.

The probability $\mathbf{Pr}\left[\mathcal{A}\right]$ is given by

$$\mathbf{Pr}\left[\mathcal{A}\right] = Q_l \cdot P^{(1)} \cdot P^{(2)},$$

where

$$P^{(i)} = P_i \sum_{\mathcal{J} \in \mathfrak{P}^{(\mathrm{i})}} \exp\left(- \sum_{j \in \mathcal{J}} L_j \right),$$

for $i = 1, 2$. Here Q_l is the probability that $k+l$ columns in a uniformly random, binary $k \times (k+l)$ matrix have full rank. Also, $\mathfrak{P}^{(\mathrm{i})}$ is the set containing the T index sets corresponding to the T different vectors in $\mathcal{L}_i$, for $i = 1, 2$. For the sizes of k used in this paper, with very good precision we have

$$Q_l = \prod_{i=1+l}^{\infty} (1 - 2^{-i}).$$

Here we have $Q_0 \approx 0.2888$, and for each new column that is added the probability of not getting a matrix with full rank is roughly halved. Thus the probability of getting a full rank matrix increases fast with l. The next subsection will try to estimate (the remaining factors of) the probability $\mathbf{Pr}\left[\mathcal{A}\right]$.

5.1 Estimating and Simulating $\mathbf{Pr}\left[\mathcal{A}\right]$

As $\mathbf{Pr}\left[\mathcal{A}\right]$ depends on the received vector $\mathbf{r}$, it appears to be quite complicated to provide a useful explicit expression or approximation for $\mathbb{E}(\mathbf{Pr}\left[\mathcal{A}\right])$, where the expectation is over $\mathbf{r}$. We choose instead to provide thorough simulation results to illustrate how $\mathbf{Pr}\left[\mathcal{A}\right]$ compares to other previous algorithms. In our comparison, $\mathbf{Pr}\left[\mathcal{A}\right]$ directly translates to the success probability for the algorithm in question. We have simulated the following algorithms:

- The Soft Stern algorithm as proposed in the paper.
- Ordered Statistics Decoding (OSD). As explained in for example [15, 27, 30], we select the k most reliable and independent positions to form the most reliable basis (MRB). The error patterns in the list are chosen according to their reliability.
- Box-and-Match approach [28]. The essence of this algorithm is Stern-type, choosing the operated information set from the most reliable positions (i.e., an extension of MRB). However, they ignore the bit reliability when

building the two colliding lists. For ease of comparison, we estimate the performance of its variant similar to the newly proposed algorithm but without choosing the error patterns in the colliding lists according to their reliability.

- A hard-decision Stern algorithm. This is a simple approach where we first make a hard decision in each position and then apply the original Stern algorithm. Each position of the received vector is $X_i \sim \mathbb{N}(1, \sigma)$ if zero is sent and hard decision gives a bit error if $X_i < 0$. The bit error probability is $p = \phi(1/\sigma)$. The probability of t errors is $\sum \binom{n}{t} p^t (1-p)^{n-t}$. The simulation results show that for the simulated parameter setting, this algorithm performs much worse than its three counterparts. We thereby removed it from our comparisons in the plots for readability.

For simplicity of analysis we compare single iteration versions of the algorithms. Techniques for taking advantage the soft information in multiple iterations is discussed briefly in Section 6.2, and can be applied to any of the algorithms. For a fair comparison, we assume that the complexity in one-round is approximately $C_{Gauss} + C \cdot (n-k)T$, where C is a small constant. Thus, we assume that for every algorithm, the size of one list is limited to $T = 2^l$ (to $2T$ for OSD, since only one list is built in this algorithm). The comparison of $\mathbb{E}(\mathbf{Pr}[\mathcal{A}])$ for various k, σ, T is shown in figures below. In all figures we let $n = 2k$. Thus, we have a code rate of $1/2$. In all figures we ignore the Q_l factor.[1] We look at two different scenarios, one with parameters applicable to a cryptographic scenario and one with parameters applicable to a coding theoretic scenario.

We have implemented the algorithm in Sage [24] [2]. The implementation covers the algorithm as described in Section 3. It was used to create the example from Section 4 and for simulating the success probability in this section. The source code for the implementation can be obtained upon request.

Cryptographic Scenario

In cryptographic applications of general soft decoding algorithms it is not uncommon to see a very small, but still non-negligible success probability $\mathbf{Pr}[\mathcal{A}]$. A large value of T is typically used. To compare the performance of the algorithms in a crypto scenario we use large σ values. We let σ vary between 0.65 and 1 (in the latter case the capacity of the channel and the code rate are equal). We let $T = 2^l = 2^{20}$. In Figure 2, we plot the logarithm of the success probability as a function of σ in the cases where $k = 256$ and $k = 1024$. In both cases our soft Stern algorithm performs much better than the other algorithms. Notice that the scale on the y-axis is not the same in the two plots.

Coding Scenario

In a coding scenario it is crucial that the word error probability $1 - \mathbf{Pr}[\mathcal{A}]$ is small. The acceptable value of T is smaller than in the cryptographic setting.

[1] In a practical application of the algorithm one would have to swap in a few slightly less reliable positions if the first k positions are not linearly independent. Unless k is small this should not change the probabilities significantly.

[2] The implementation is available at https://github.com/ErikMaartensson/SoftStern

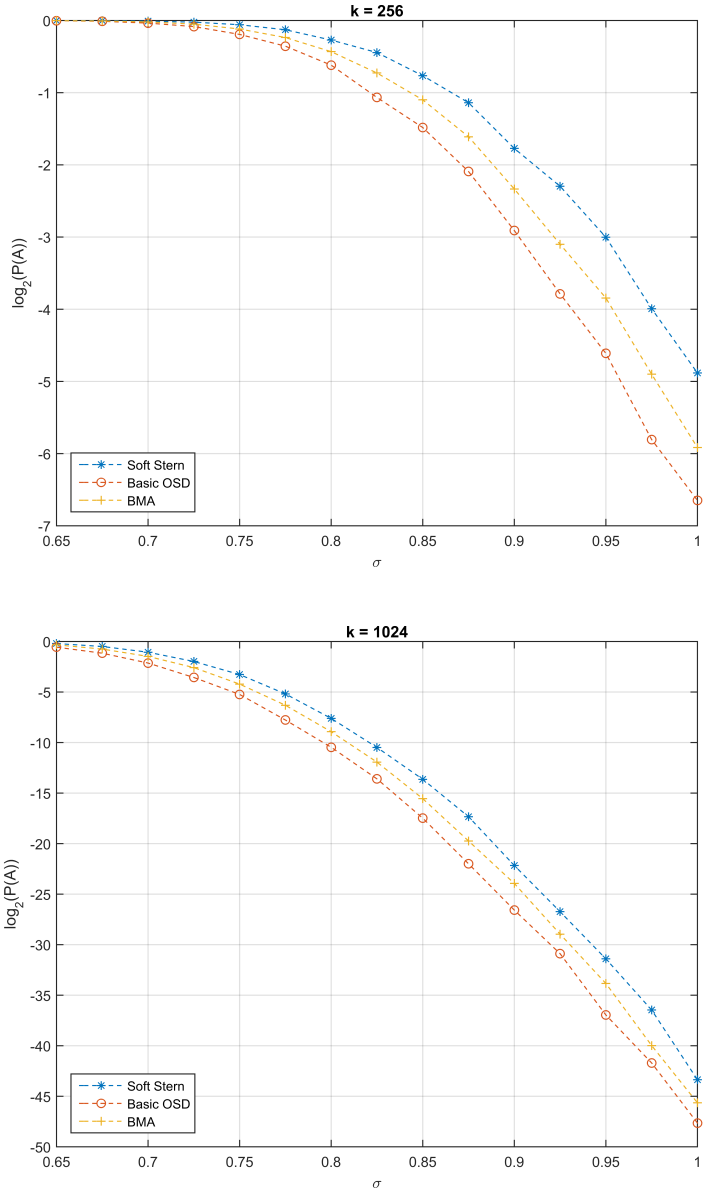


Figure 2: The logarithm of the success probability for the different algorithms as a function of σ.

To compare the algorithms we look at their probability of failing for small σ values. We vary σ between 0.4 and 0.65. We let $T = 2^l = 2^{10}$. In Figure 3, we plot the failure probability as a function of σ in the cases where $k = 128$ and $k = 512$. again, in both cases our soft Stern algorithm outperforms the other algorithms. Again, notice that the scale on the y-axis is not the same in the two plots.

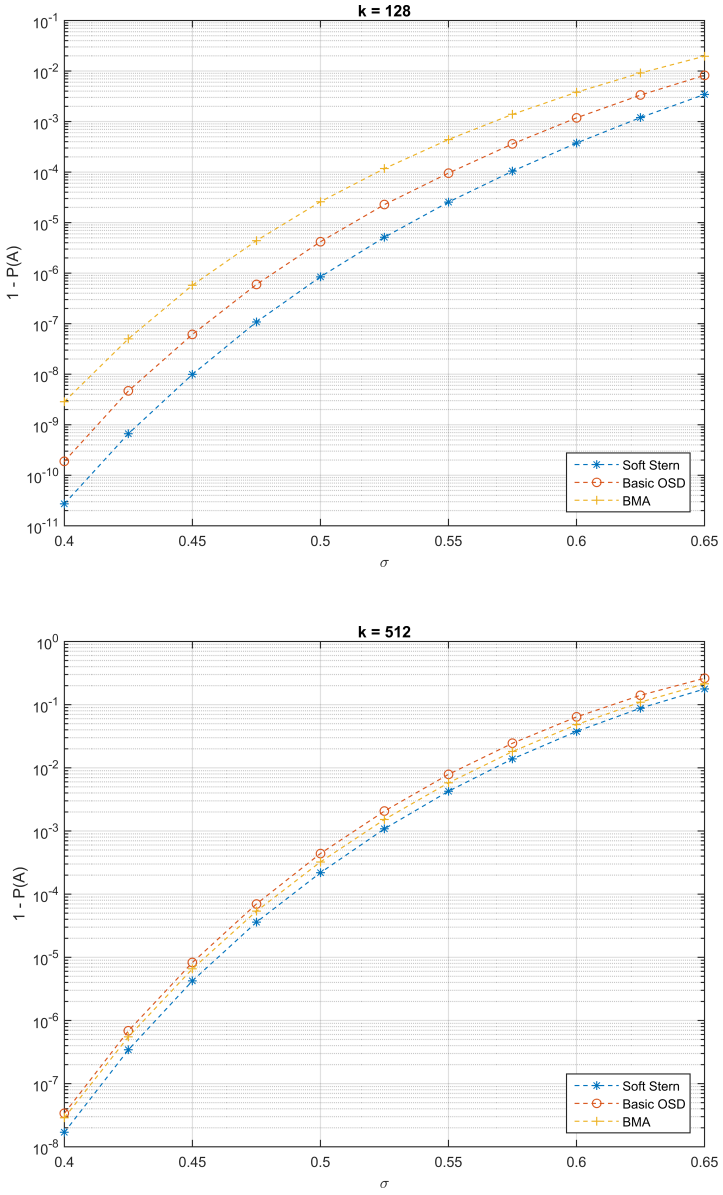


Figure 3: The failure probability for the different algorithms as a function of σ.

6 Generalizations

6.1 Soft Output

The algorithm can easily be modified to allow for soft output. The algorithm above outputs either the codeword $\hat{\mathbf{c}}$ closest to the received vector $\mathbf{r}$, or the first vector that is within some distance δ of $\mathbf{r}$. Instead, we can keep a number of the vectors $\mathbf{c}_i$ closest to $\mathbf{r}$. Based on the probabilities of each of the corresponding

bit patterns we can then calculate the weighted average for each position. Now the algorithm can output soft information.

6.2 Multiple Iterations

If we are unsuccessful in our one-pass algorithm, we might want to allow for a new iteration, or many, to increase the success probability. We then suggest to swap one column from $\mathcal{S}$ with one column from $\mathcal{O}$. We want to take advantage of the reliability values, while still having a degree of randomness in the swapping. The technique we suggest is the approach experimentally tested as the optimal in [22]. Here the probability of swapping out $i \in \mathcal{S}$ is proportional to the probability that the corresponding position is wrongfully classified, that is, $(1-p_i)/\sum_{p_j \in \mathcal{S}}(1-p_j)$, where p_i is the conditional probability of having a correct bitwise hard decision, as being defined in (2). The probability of swapping in $j \in \mathcal{O}$ is proportional to the squared bias of j, that is, $\tau_j^2/\sum_{\tau_k \in \mathcal{O}} \tau_k^2$, where τ_j is the respective bias, i.e., $\tau_j = |p_j - \frac{1}{2}|$. The complexity can be analysed by employing a Markov-chain model, as was done in [7, 8].

7 Applications

7.1 Breaking Soft McEliece

In [2], using soft information to enhance the performance of an attacking algorithm has been discussed, but no attacks below the security level have been presented. We show that soft McEliece can be broken by a trivial variant of Algorithm 2. The adversary will employ a simplistic version, i.e., keeping one element in each list. Therefore, she chooses l to be 0 and the considered error pattern is $\mathbf{0}$ in the k most reliable positions.

The attack can also be described as follows. The adversary chooses the k most reliable indices to form an information set $\mathcal{I}$, makes a bit-wise hard decision $\mathrm{sgn}(\cdot)$, and calculates the message $\mathbf{u}$ via a Gaussian elimination. She then tests whether this is a valid message. Otherwise, the adversary selects another ciphertext and tries again (if a single ciphertext can be broken the scheme is considered insecure). For one pass, the attack succeeds in case (i) that the sub-matrix corresponding to this information set is invertible and (ii) that there exist no errors among these positions. In implementation this latter probability is 0.98 if 80-bit security is targeted, and the expected complexity for recovering one message is about 3.5 Gaussian eliminations.

We give some intuition why this basic strategy can solve the decoding problem in soft McEliece for the proposed security parameters. In [2], one key security argument is that the total number of bit errors in one ciphertext follows a modified binomial distribution, which gives at least $\frac{n}{2}\mathrm{erfc}\left(\frac{1}{\sqrt{2}\sigma}\right)$ bit errors. However, for the most reliable coordinates, the number of bit errors are very few. We see that the expected number of bit errors among the $\frac{n}{2}$ most reliable bits is only 0.022 (or 0.015) using the parameters for 80 (or 128)-bit security. Most of the error positions are among the least reliable ones.

Moving to a higher noise level

Though this simplistic attack works well for soft McEliece, Algorithm 2 performs much better when the size of the targeted instance increases. Therefore, one should employ the full algorithm when aiming for cryptosystems with a reasonable security level.

A higher noise level increases the decryption (decoding) error probability. If $(n, \sigma) = (7202, 0.66)$, for instance, the 3601 most reliable bits are error-free with probability $2^{-13.0}$. Hence, on average about 29,000 Gaussian eliminations are required using this simplistic attack. By using soft Stern, setting $l = 23$ and choosing a suitable δ in Algorithm 2, we can reduce the expected complexity to around 23 Gaussian eliminations[3].

7.2 Applications in Side-channel Attacks

Transforming some problems in side-channel analysis to that of decoding random linear codes originates in [5]. In this context, although the noise distribution is not exactly a Gaussian, soft information can still be exploited, making Algorithm 2 more efficient than other ISD variants. For side-channel attacks in [21, 22], the following modified version of the LPN problem occurs. Here, $\mathrm{Ber}(p)$ denotes a random variable that takes the value 1 with probability p and 0 with probability $1 - p$, and $\langle \cdot, \cdot \rangle$ denotes the binary inner product of two vectors.

Definition 7.1 (Learning Parity with Variable Noise (LPVN) [22])**.** *Let $\mathbf{s} \in \mathbb{F}_2^k$ and ψ be a probability distribution over [0, 0.5]. Given n uniformly sampled vectors $\mathbf{a}_i \in \mathbb{F}_2^k$, n error probabilities ϵ_i sampled from ψ, and noisy observations $b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i = c_i + e_i$, where e_i is sampled from $Ber(\epsilon_i)$, find $\mathbf{s}$.*

They solve the problem by translating it into decoding a random linear code with soft information. They apply Stern's algorithm, but they do not sort the error patterns based on their probability of occurring. In this case the error is not Gaussian, but with some minor modifications our algorithm can still be applied. We sort the positions based on the ϵ_i values. The smaller ϵ_i is, the more reliable the position is. Next, we have

$$p_i = \mathbf{Pr}\left[b_i = c_i | \epsilon_i\right] = 1 - \epsilon_i.$$

After having done the transformations, such that the all-zero vector is the most probable vector in an index set $\mathcal{S}$, the probability of a bit pattern with ones in positions in $\mathcal{J} \subset \mathcal{S}$ and zeros in the other positions is

$$\prod_{i \in \mathcal{J}} \epsilon_i \cdot \prod_{i \notin \mathcal{J}, i \in \mathcal{S}} (1 - \epsilon_i) = \frac{\prod_{i \in \mathcal{J}} \epsilon_i}{\prod_{i \in \mathcal{J}} (1 - \epsilon_i)} \cdot \prod_{i \in \mathcal{S}} (1 - \epsilon_i) = \prod_{i \in \mathcal{J}} \frac{\epsilon_i}{(1 - \epsilon_i)} \cdot \prod_{i \in \mathcal{S}} p_i.$$

With some minor adjustments, the method for finding the most probable bit patterns, described in Section 3.2, can now be used.

[3] In the conference version [16], we presented an upper bound on the time complexity of solving this instance, i.e., 31 Gaussian eliminations. After careful simulation, we can now show a more accurate complexity estimation.

7.3 Hybrid Decoding

Another problem suited for our algorithm can be found in [1], where the problem of decoding linear codes with soft information appears. They analyze two codes proposed for space telecommanding. Both are LDPC codes, with $(n, k) = (128, 64)$ and $(n, k) = (512, 256)$ respectively. A hybrid approach for decoding is used. First one applies an efficient iterative decoder. In the few cases when the iterative decoder fails, one uses a decoder based on ordered statistics, thereby reducing the risk of decoding failure drastically.

However, the proposed ordered statistics algorithm does not make use of a Stern-type speed-up. It orders the positions after decreasing reliability. Then they try different error patterns in the k most reliable positions. Using our soft Stern algorithm, we instead divide the most reliable $k + l$ positions into two sets, and then look for collisions between the partial syndromes of the bit error patterns in the two sets. Adding such a Stern-type modification would greatly improve their ordered statistics decoder.

7.4 Product Codes

An application of the soft Stern algorithm with soft output is the decoding of product codes. Consider the serial concatenation of two codes, that do not have an efficient decoder with soft output. A small example would be the Golay code. An iterative decoder for this product code can be constructed by using the soft Stern algorithm with soft output together with a message-passing network (Tanner graph) between code symbols in the product code. Investigating this idea in more detail is an interesting research direction.

8 Conclusions

We have presented a new information set decoding algorithm using bit reliability, called the soft Stern algorithm. The algorithm outperforms what has been previously suggested for decoding general codes on the AWGN channel and similar tasks.

It can be utilized for a very efficient message-recovery attack on a recently proposed McEliece-type PKC named Soft McEliece [2], for an improved hybrid approach of decoding LDPC codes as in [1], and for side-channel attacks as in [21, 22]. We have also mentioned its use for decoding product codes.

Some modifications, such as multiple iterations of the algorithm, and producing soft output values, were discussed but not explicitly analyzed. Some ideas of future work may include further analyzing its use in iterative decoding and extending and deriving the exact algorithmic steps when considering multiple iterations.

Acknowledgments

The authors would like to thank the anonymous reviewers from ISIT 2017 and the reviewers for Advances in Mathematics of Communications for helping us

improve the quality of this paper.

References

[1] M. Baldi, N. Maturo, E. Paolini and F. Chiaraluce, On the use of ordered statistics decoders for low-density parity-check codes in space telecommand links, *EURASIP Journal on Wireless Communications and Networking*, **2016** (2016), 1–15.

[2] M. Baldi, P. Santini and F. Chiaraluce, Soft McEliece: MDPC code-based McEliece cryptosystems with very compact keys through real-valued intentional errors, in *IEEE International Symposium on Information Theory ISIT*, IEEE, (2016), 795–799.

[3] 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 *Advances in Cryptology – EUROCRYPT* (eds. D. Pointcheval and T. Johansson), Springer, (2012), 520–536.

[4] S. Belaïd, J.-S. Coron, P.-A. Fouque, B. Gèrard, J.-G. Kammerer and E. Prouff, Improved Side-Channel Analysis of Finite-Field Multiplication., in *Cryptographic Hardware and Embedded Systems – CHES* (eds. T. Güneysu and H. Handschuh), Springer, (2015), 395–415.

[5] S. Belaïd, P.-A. Fouque and B. Gèrard, Side-channel analysis of multiplications in GF(2^{128}), in *Advances in Cryptology – ASIACRYPT* (eds. P. Sarkar and T. Iwata), Springer, (2014), 306–325.

[6] D. J. Bernstein, T. Lange and C. Peters, Smaller decoding exponents: Ball-collision decoding, in *Advances in Cryptology – CRYPTO* (eds. P. Rogaway), Springer, (2011), 743–760.

[7] D. J. Bernstein, T. Lange and C. Peters, Attacking and Defending the McEliece Cryptosystem, in *International Workshop on Post-Quantum Cryptography – PQCrypto* (eds. J. Buchmann and J. Ding), Springer, (2008), 31–46.

[8] A. Canteaut and F. Chabaud, A new algorithm for finding minimum-weight wordsin a linear code: application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511, *IEEE Transactions on Information Theory*, **44** (1998), 367–378.

[9] D. Chase, A class of algorithms for decoding block codes with channel measurement information, *IEEE Transactions on Information theory*, **18** (1972), 170–182.

[10] J. Conway and N. Sloane, Soft decoding techniques for codes and lattices, including the Golay code and the Leech lattice, *IEEE Transactions on Information Theory*, **32** (1986), 41–50.

[11] I. Dumer, Sort-and-match algorithm for soft-decision decoding, *IEEE Transactions on Information Theory*, **45** (1999), 2333–2338.

[12] S. Dziembowski, S. Faust, G. Herold, A. Journault, D. Masny and F. Standaert, Towards sound fresh re-keying with hard (physical) learning problems, in *Advances in Cryptology – CRYPTO* (eds. M. Robshaw and J. Katz), Springer, (2016), 272–301.

[13] M. Finiasz and N. Sendrier, Security bounds for the design of code-based cryptosystems, in *Advances in Cryptology – ASIACRYPT*, (eds. M. Matsui), Springer, (2009), 88–105.

[14] M. P. Fossorier and S. Lin, Soft-decision decoding of linear block codes based on ordered statistics, *IEEE Transactions on Information Theory*, **41** (1995), 1379–1396.

[15] D. Gazelle and J. Snyders, Reliability-Based Code-Search Algorithms for Maximum-Likelihood Decoding of Block Codes, *IEEE Transactions on Information Theory*, **43** (1997), 239–249.

[16] Q. Guo, T. Johansson, E. Mårtensson and P. Stankovski, Information Set Decoding with Soft Information and some Cryptographic Applications, in *IEEE International Symposium on Information Theory – ISIT*, IEEE, (2017), 1793–1797.

[17] R. Koetter and A. Vardy, Algebraic soft-decision decoding of Reed-Solomon codes, *IEEE Transactions on Information Theory*, **49** (2003), 2809–2825.

[18] P. J. Lee and E. F. Brickell, An observation on the security of McEliece's public-key cryptosystem, in *Workshop on the Theory and Application of Cryptographic Techniques*, Springer, (1988), 275–280.

[19] A. May and I. Ozerov, On computing nearest neighbors with applications to decoding of binary linear codes, in *Advances in Cryptology - EUROCRYPT* (eds. E. Oswald and M. Fischlin), Springer, (2015), 203–228.

[20] R. Misoczki, J. P. Tillich, N. Sendrier and P. S. Barreto, MDPC-McEliece: New McEliece variants from moderate density parity-check codes, in *IEEE International Symposium on Information Theory – ISIT*, IEEE, (2013), 2069–2073.

[21] P. Pessl, L. Groot Bruinderink and Y. Yarom, To BLISS-B or not to be: Attacking strongSwan's Implementation of Post-Quantum Signatures, in *ACM SIGSAC Conference on Computer and Communications Security – CCS*, ACM, (2017), 1843–1855.

[22] P. Pessl and S. Mangard, Enhancing side-channel analysis of binary-field multiplication with bit reliability, in *RSA Conference Cryptographers' Track (CT-RSA)* (eds. K. Sako), (2016), 255–270.

[23] E. Prange, The use of information sets in decoding cyclic codes, *IRE Transactions on Information Theory*, **8** (1962), 5–9.

[24] The Sage Developers, SageMath, the Sage Mathematics Software System, http://www.sagemath.org, 2018.

[25] J. Stern, A method for finding codewords of small weight, in *Coding Theory and Applications* (eds. G. Cohen and J. Wolfmann), (1988), 106–113.

[26] A. Valembois, Fast soft-decision decoding of linear codes, stochastic resonance in algorithms, in *IEEE International Symposium on Information Theory – ISIT*, IEEE, (2000), 91.

[27] A. Valembois and M. Fossorier, Generation of binary vectors that optimize a given weight function with application to soft-decision decoding, in *IEEE Information Theory Workshop*, IEEE, (2001), 138–140.

[28] A. Valembois and M. Fossorier, Box and match techniques applied to soft-decision decoding, *IEEE Transactions on Information Theory*, **50** (2004), 796–810.

[29] A.J. Viterbi and J.K. Omura, *Principles of Digital Communication and Coding,* McGraw-Hill, New York, 1979.

[30] Y. Wu and C. N. Hadjicostis, Soft-decision decoding using ordered recodings on the most reliable basis, *IEEE transactions on information theory*, **53** (2007), 829–836.

Paper V

On the Asymptotics of Solving the LWE Problem Using Coded-BKW with Sieving

The Learning with Errors problem (LWE) has become a central topic in recent cryptographic research. In this paper, we present a new solving algorithm combining important ideas from previous work on improving the Blum-Kalai-Wasserman (BKW) algorithm and ideas from sieving in lattices. The new algorithm is analyzed and demonstrates an improved asymptotic performance. For the Regev parameters $q = n^2$ and noise level $\sigma = n^{1.5}/(\sqrt{2\pi}\log_2^2 n)$, the asymptotic complexity is $2^{0.893n}$ in the standard setting, improving on the previously best known complexity of roughly $2^{0.930n}$. The newly proposed algorithm also provides asymptotic improvements when a quantum computer is assumed or when the number of samples is limited. Also for a polynomial number of samples an asymptotic improvement is shown. For concrete parameter instances, improved performance is indicated..

Keywords: LWE, BKW, Coded-BKW, Lattice codes, Lattice sieving.

1 Introduction

Post-quantum crypto, the area of cryptography in the presence of quantum computers, is currently a major topic in the cryptographic community. Cryptosystems based on hard problems related to lattices are currently intensively investigated, due to their possible resistance against quantum computers. The major problem in this area, upon which cryptographic primitives can be built, is the *Learning with Errors* (LWE) problem.

LWE is an important, efficient and versatile problem. One famous application of LWE is the construction of Fully Homomorphic Encryption schemes [15–17, 22]. A major motivation for using LWE is its connections to lattice problems, linking the difficulty of solving LWE (on average) to the difficulty of solving instances of some (worst-case) famous lattice problems. Let us state the LWE problem.

Definition 1.1. *Let n be a positive integer, q a prime, and let $\mathcal{X}$ be an error distribution selected as the discrete Gaussian distribution on $\mathbb{Z}_q$. Fix $\mathbf{s}$ to be a secret vector in $\mathbb{Z}_q^n$, chosen according to a uniform distribution. Denote by $L_{\mathbf{s},\mathcal{X}}$ the probability distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ obtained by choosing $\mathbf{a} \in \mathbb{Z}_q^n$ uniformly at random, choosing an error $e \in \mathbb{Z}_q$ according to $\mathcal{X}$ and returning*

$$(\mathbf{a}, z) = (\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e)$$

in $\mathbb{Z}_q^n \times \mathbb{Z}_q$. The (search) LWE *problem is to find the secret vector $\mathbf{s}$ given a fixed number of samples from $L_{\mathbf{s},\mathcal{X}}$.*

The definition above gives the *search* LWE problem, as the problem description asks for the recovery of the secret vector $\mathbf{s}$. Another variant is the *decision* LWE problem. In this case the problem is to distinguish between samples drawn from $L_{\mathbf{s},\mathcal{X}}$ and a uniform distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$. Typically, we are then interested in distinguishers with non-negligible advantage.

For the analysis of algorithms solving the LWE problem in previous work, there are essentially two different approaches. One being the approach of calculating the specific number of operations needed to solve a certain instance for a particular algorithm, and comparing specific complexity numbers. The other approach is asymptotic analysis. Solvers for the LWE problem with suitable parameters are expected to have fully exponential complexity, say bounded by 2^{cn} as n tends to infinity. Comparisons between algorithms are made by deriving the coefficient c in the asymptotic complexity expression.

1.1 Related Work

We list the three main approaches for solving the LWE problem in what follows. A good survey with concrete complexity considerations is [6] and for asymptotic comparisons, see [27].

The first class is the algebraic approach, which was initialized by Arora-Ge [8]. This work was further improved by Albrecht et al., using Gröbner bases [2]. Here we point out that this type of attack is mainly, asymptotically, of interest when the noise is very small. For extremely small noise the complexity can be polynomial.

The second and most commonly used approach is to rewrite the LWE problem as a lattice problem, and therefore lattice reduction algorithms [18,46], such

as sieving and enumeration can be applied. There are several possibilities when it comes to reducing the LWE problem to some hard lattice problem. One is a direct approach, writing up a lattice from the samples and then to treat the search LWE problem as a Bounded Distance Decoding (BDD) problem [36,37]. One can also reduce the BDD problem to a UNIQUE-SVP problem [5]. Another variant is to consider the distinguishing problem in the dual lattice [40]. Lattice-based algorithms have the advantage of not using an exponential number of samples.

The third approach is the BKW-type algorithms.

BKW variants

The BKW algorithm was originally proposed by Blum, Kalai and Wasserman [13] for solving the Learning Parity with Noise (LPN) problem (LWE for $q = 2$). It resembles Wagner's generalized birthday approach [47].

For the LPN case, there has been a number of improvements to the basic BKW approach. In [35], transform techniques were introduced to speed up the search part. Further improvements came in work by Kirchner [30], Bernstein and Lange [11], Guo et al. [23], Zhang et al. [49], Bogos and Vaudenay [14].

Albrecht et al. were first to apply BKW to the LWE problem [3], which they followed up with Lazy Modulus Switching (LMS) [4], which was further improved by Duc et al. in [21]. The basic BKW approach for LWE was improved in [25] and [31], resulting in an asymptotic improvement. These works improved by reducing a variable number of positions in each step of the BKW procedure as well as introducing a coding approach. Although the two algorithms were slightly different, they perform asymptotically the same and we refer to the approach as coded-BKW. It was proved in [31] that the asymptotic complexity for Regev parameters (public-key cryptography parameter) $q = n^2$ and noise level $\sigma = n^{1.5}/(\sqrt{2\pi}\log_2^2 n)$ is $2^{0.930n+o(n)}$, the currently best known asymptotic performance for such parameters.

Sieving algorithms

A key part of the algorithm to be proposed is the use of sieving in lattices. The first sieving algorithm for solving the shortest vector problem was proposed by Ajtai, Kumar and Sivakumar in [1], showing that SVP can be solved in time and memory $2^{\Theta(n)}$. Subsequently, we have seen the NV-sieve [43], List-sieve [41], and provable improvement of the sieving complexity using the birthday paradox [26, 44].

With heuristic analysis, [43] started to derive a complexity of $2^{0.415n+o(n)}$, followed by GaussSieve [41], 2-level sieve [48], 3-level sieve [50] and overlattice-sieve [10]. Laarhoven started to improve the lattice sieving algorithms employing algorithmic breakthroughs in solving the nearest neighbor problem, angular LSH [32], and spherical LSH [34]. The asymptotically most efficient approach when it comes to time complexity is Locality Sensitive Filtering (LSF) [9] with both a space and time complexity of $2^{0.292n+o(n)}$. Using quantum computers, the complexity can be reduced to $2^{0.265n+o(n)}$ (see [33]) by applying Grover's quantum search algorithm.

1.2 Contributions

We propose a new algorithm for solving the LWE problem combining previous combinatorial methods with an important algorithmic idea – using a sieving approach. Whereas BKW combines vectors to reduce positions to zero, the previously best improvements of BKW, like coded-BKW, reduce more positions but at the price of leaving a small but in general nonzero value in reduced positions. These values are considered as additional noise. As these values increase in magnitude for each step, because we add them together, they have to be very small in the initial steps. This is the reason why in coded-BKW the number of positions reduced in a step is increasing with the step index. We have to start with a small number of reduced positions, in order to not obtain a noise that is too large.

The proposed algorithm tries to solve the problem of the growing noise from the coding part (or LMS) by using a sieving step to make sure that the noise from treated positions does not grow, but stays approximately of the same size. The basic form of the new algorithm then contains two parts in each iterative step. The first part reduces the magnitude of some particular positions by finding pairs of vectors that can be combined. The second part performs a sieving step covering all positions from all previous steps, making sure that the magnitude of the resulting vector components is roughly as in the already size-reduced part of the incoming vectors.

We analyze the new algorithm from an asymptotic perspective, proving a new improved asymptotic performance. For the asymptotic Regev parameters $q = n^2$ and noise level $\sigma = n^{1.5}$, the result is a time and space complexity of $2^{0.8951n+o(n)}$, which is a significant asymptotic improvement. We also get a first quantum acceleration (with complexity of $2^{0.8856n+o(n)}$) for the Regev parameters by using the performance of sieving in the quantum setting.

In addition, when the sample complexity is limited, e.g, to a polynomial number in n like $\Theta(n \log n)$, the new algorithm outperforms the previous best solving algorithms for a wide range of parameter choices.

Lastly, the new algorithm can be flexibly extended and generalized in various ways. We present three natural extensions further improving its asymptotic performance. For instance, by changing the reduction scale in each sieving step, we reduce the time and space complexity for the Regev parameters to $2^{0.8927n+o(n)}$ in the standard setting. With the help of quantum computers, the complexity can be even further reduced, down to $2^{0.8795n+o(n)}$.

1.3 Organization

The remaining parts of the paper are organized as followed. We start with some preliminaries in Section 2, including more basics on LWE, discrete Guassians and sieving in lattices. In Section 3 we review the details of the BKW algorithm and some recent improvements. Section 4 just contains a simple reformulation. In Section 5 we give the new algorithm in its basic form and in Section 6 we derive the optimal parameter selection and perform the asymptotic analysis. In Section 7 we derive asymptotic expressions for LWE with sparse secrets, and show an improved asymptotic performance when only having access to a polynomial number of samples. Section 8 contains new versions of coded-BKW with sieving, that further decrease the asymptotic complexity. Finally,

we conclude the paper in Section 9.

2 Background

2.1 Notations

Throughout the paper, the following notations are used.

- We write $\log(\cdot)$ for the base 2 logarithm and $\ln(\cdot)$ for the natural logarithm.
- In an n-dimensional Euclidean space $\mathbb{R}^n$, by the norm of a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ we refer to its L_2-norm, defined as

  $$\|\mathbf{x}\| = \sqrt{x_1^2 + \cdots + x_n^2}.$$

  We then define the Euclidean distance between two vectors $\mathbf{x}$ and $\mathbf{y}$ in $\mathbb{R}^n$ as $\|\mathbf{x} - \mathbf{y}\|$.
- An element in $\mathbb{Z}_q$ is represented as the corresponding value in $[-\frac{q-1}{2}, \frac{q-1}{2}]$.
- For an $[N, k_0]$ linear code, N denotes the code length and k_0 denotes the dimension.
- We use the following standard notations for asymptotic analysis.
  - $f(n) = \mathcal{O}(g(n))$ if there exists a positive constant C, s.t., $|f(n)| \leq C \cdot g(n)$ for n sufficiently large.
  - $f(n) = \Omega(g(n))$ if there exists a positive constant C, s.t., $|f(n)| \geq C \cdot g(n)$ for n sufficiently large.
  - $f(n) = \Theta(g(n))$ if there exist positive constants C_1 and C_2, s.t., $C_1 \cdot g(n) \leq |f(n)| \leq C_2 \cdot g(n)$ for n sufficiently large.
  - $f(n) = o(g(n))$ if for every positive C, we have $|f(n)| < C \cdot g(n)$ for n sufficiently large.

2.2 LWE Problem Description

Rather than giving a more formal definition of the decision version of LWE, we instead reformulate the search LWE problem, because our main purpose is to investigate its solving complexity. Assume that m samples

$$(\mathbf{a}_1, z_1), (\mathbf{a}_2, z_2), \ldots, (\mathbf{a}_m, z_m),$$

are drawn from the LWE distribution $L_{\mathbf{s},\mathcal{X}}$, where $\mathbf{a}_i \in \mathbb{Z}_q^n, z_i \in \mathbb{Z}_q$. Let $\mathbf{z} = (z_1, z_2, \ldots, z_m)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_m) = \mathbf{sA}$. We can then write

$$\mathbf{z} = \mathbf{sA} + \mathbf{e},$$

where $\mathbf{A} = \begin{bmatrix} \mathbf{a}_1^{\mathrm{T}} & \mathbf{a}_2^{\mathrm{T}} & \cdots & \mathbf{a}_m^{\mathrm{T}} \end{bmatrix}$, $z_i = y_i + e_i = \langle \mathbf{s}, \mathbf{a}_i \rangle + e_i$ and $e_i \xleftarrow{\$} \mathcal{X}$. Therefore, we have reformulated the search LWE problem as a decoding problem, in which the matrix $\mathbf{A}$ serves as the generator matrix for a linear code over $\mathbb{Z}_q$ and $\mathbf{z}$ is the received word. We see that the problem of searching for the secret vector $\mathbf{s}$ is equivalent to that of finding the codeword $\mathbf{y} = \mathbf{sA}$ such that the Euclidean distance $||\mathbf{y} - \mathbf{z}||$ is minimal.

The Secret-Noise Transformation

An important transformation [7, 30] can be applied to ensure that the secret vector follows the same distribution $\mathcal{X}$ as the noise. The procedure works as follows. We first write $\mathbf{A}$ in systematic form via Gaussian elimination. Assume that the first n columns are linearly independent and form the matrix $\mathbf{A_0}$. We then define $\mathbf{D} = {\mathbf{A_0}}^{-1}$ and write $\hat{\mathbf{s}} = \mathbf{s}\mathbf{D}^{-1} - (z_1, z_2, \ldots, z_n)$. Hence, we can derive an equivalent problem described by $\hat{\mathbf{A}} = (\mathbf{I}, \hat{\mathbf{a}}_{n+1}^{\mathrm{T}}, \hat{\mathbf{a}}_{n+2}^{\mathrm{T}}, \cdots, \hat{\mathbf{a}}_{m}^{\mathrm{T}})$, where $\hat{\mathbf{A}} = \mathbf{D}\mathbf{A}$. We compute

$$\hat{\mathbf{z}} = \mathbf{z} - (z_1, z_2, \ldots, z_n)\hat{\mathbf{A}} = (\mathbf{0}, \hat{z}_{n+1}, \hat{z}_{n+2}, \ldots, \hat{z}_m).$$

Using this transformation, one can assume that each entry in the secret vector is now distributed according to $\mathcal{X}$.

The noise distribution $\mathcal{X}$ is usually chosen as the discrete Gaussian distribution, which will be briefly discussed in Section 2.3.

2.3 Discrete Gaussian Distribution

We start by defining the discrete Gaussian distribution over $\mathbb{Z}$ with mean 0 and variance σ^2, denoted $D_{\mathbb{Z},\sigma}$. That is, the probability distribution obtained by assigning a probability proportional to $\exp(-x^2/2\sigma^2)$ to each $x \in \mathbb{Z}$. The discrete Gaussian over $\mathbb{Z}^n$ with variance σ^2, denoted $D_{\mathbb{Z}^n,\sigma}$, is defined as the product distribution of n independent copies of $D_{\mathbb{Z},\sigma}$.

Then, the discrete Gaussian distribution $\mathcal{X}$ over $\mathbb{Z}_q$ with variance σ^2 (also denoted $\mathcal{X}_\sigma$) can be defined by folding $D_{\mathbb{Z},\sigma}$ and accumulating the value of the probability mass function over all integers in each residue class modulo q.

Following the path of previous work [3], we make the following heuristic assumption about our discussed instances.

Heuristic 1. The discrete Gaussian distribution can be approximated by the continuous counterpart. For instance, if X is drawn from $\mathcal{X}_{\sigma_1}$ and Y is drawn from $\mathcal{X}_{\sigma_2}$, then $X + Y$ is regarded as being drawn from $\mathcal{X}_{\sqrt{\sigma_1^2+\sigma_2^2}}$.

This approximation, which is widely-adopted in literature (e.g., [4, 25, 28]), is motivated by the fact that a sufficient wild Gaussian[1] blurs the discrete structures.[2]

[1] It is proven in [39] that the noise is sufficiently large for the integer lattice of $\mathbb{Z}^n$ if σ is of order $\Omega(n^c)$ for $c > 0.5$.

[2] The LWE noise distribution is formally treated in [21] and in [31]. However, in [21], only the original BKW steps are investigated, and in [31], a non-standard LWE noise distribution is assumed. Both are non-applicable.

On the other hand, the adopted assumption from the literature to approximate the discrete Gaussian is strong. We can simplify it to two weaker heuristic assumptions in our deviation. Firstly, many noise variables with small variance will be generated and be added or subtracted. We assume that all the small noise variables are independent so that we can add their variance as can be done in the continuous Gaussian case. Secondly, by intuition from the central limit theorem, we assume that the final noise variable is close to a continuous Gaussian mod q. Thus, we can use the formula from [36] to estimate the required number of samples.

The sample complexity for distinguishing.

To estimate the solving complexity, we need to determine the number of required samples to distinguish between the uniform distribution on $\mathbb{Z}_q$ and $\mathcal{X}_\sigma$. Relying on standard theory from statistics, when the noise variance σ is sufficiently large to approximate the discrete Gaussian by a continuous one mod q, using either previous work [36] or Bleichenbacher's definition of bias [42], we can conclude that the required number of samples is

$$C \cdot e^{2\pi\left(\frac{\sigma\sqrt{2\pi}}{q}\right)^2}, \tag{1}$$

where C is a small positive constant.

2.4 Sieving in Lattices

We here give a brief introduction to the sieving idea and its application in lattices for solving the shortest vector problem (SVP). For an introduction to lattices, the SVP problem, and sieving algorithms, see e.g. [9].

In sieving, we start with a list $\mathcal{L}$ of relatively short lattice vectors. If the list size is large enough, we will obtain many pairs of $\mathbf{v}, \mathbf{w} \in \mathcal{L}$, such that $\|\mathbf{v} \pm \mathbf{w}\| \leq \max\{\|\mathbf{v}\|, \|\mathbf{w}\|\}$. After reducing the size of these lattice vectors a polynomial number of times, one can expect to find the shortest vector.

The core of sieving is thus to find a close enough neighbor $\mathbf{v} \in \mathcal{L}$ efficiently, for a vector $\mathbf{w} \in \mathcal{L}$, thereby reducing the size by further operations like addition or subtraction. This is also true for our newly proposed algorithm in a later section, since by sieving we solely desire to control the size of the added/subtracted vectors. For this specific purpose, many famous probabilistic algorithms have been proposed, e.g., Locality Sensitive Hashing (LSH) [29], Bucketing coding [20], and May-Ozerov's algorithm [38] in the Hamming metric with important applications to decoding binary linear codes.

In the Euclidean metric, the state-of-the-art algorithm in the asymptotic sense is Locality Sensitive Filtering (LSF) [9], which requires $2^{0.2075n+o(n)}$ samples. In the classic setting, the time and memory requirements are both in the order of $2^{0.292n+o(n)}$. The constant hidden in the running time exponent can be reduced to 0.265 in the scenario of quantum computing. In the remaining part of the paper, we choose the LSF algorithm for the best asymptotic performance when we need to instantiate the sieving method.

3 The BKW Algorithm

The BKW algorithm is the first sub-exponential algorithm for solving the LPN problem, originally proposed by Blum, Kalai and Wasserman [12,13]. It can also be trivially adopted to the LWE problem, with single-exponential complexity.

3.1 Plain BKW

The algorithm consists of two phases: the reduction phase and the solving phase. The essential improvement comes from the first phase, whose underlying fundamental idea is the same as Wagner's generalized birthday algorithm [47]. That is, using an iterative collision procedure on the columns in the matrix $\mathbf{A}$,

one can reduce its row dimension step by step, and finally reach a new LWE instance with a much smaller dimension. The solving phase can then be applied to recover the secret vector. We describe the core procedure of the reduction phase, called a plain BKW step, as follows. Let us start with $\mathbf{A}_0 = \mathbf{A}$.

Dimension reduction: In the i-th iteration, we look for combinations of two columns in $\mathbf{A}_{i-1}$ that add (or subtract) to zero in the last b entries. Suppose that one finds two columns $\mathbf{a}^{\mathrm{T}}_{j_1,i-1}, \mathbf{a}^{\mathrm{T}}_{j_2,i-1}$ such that

$$\mathbf{a}_{j_1,i-1} \pm \mathbf{a}_{j_2,i-1} = [* \quad * \quad \cdots \quad * \quad \underbrace{0 \quad 0 \quad \cdots \quad 0}_{b \text{ symbols}}],$$

where $*$ means any value. We then generate a new vector $\mathbf{a}_{j,i} = \mathbf{a}_{j_1,i-1} \pm \mathbf{a}_{j_2,i-1}$. We obtain a new generator matrix $\mathbf{A}_i$ for the next iteration, with its dimension reduced by b, if we remove the last b all-zero positions with no impact on the output of the inner product operation. We also derive a new "observed symbol" as $z_{j,i} = z_{j_1,i-1} \pm z_{j_2,i-1}$.

A trade-off: After one step of this procedure, we can see that the new noise variable is $e_{j,i} = e_{j_1,i-1} \pm e_{j_2,i-1}$. If the noise variables $e_{j_1,i-1}$ and $e_{j_2,i-1}$ both follow the Gaussian distribution with variance σ^2_{i-1}, then the new noise variable $e_{j,i}$ is considered Gaussian distributed with variance $\sigma^2_i = 2\sigma^2_{i-1}$.

After t_0 iterations, we have reduced the dimension of the problem to $n - t_0 b$. The final noise variable is thus a summation of 2^{t_0} noise variables generated from the LWE oracle. We therefore know that the noise connected to each column is of the form

$$e = \sum_{j=1}^{2^{t_0}} e_{i_j},$$

and the total noise is approximately Gaussian with variance $2^{t_0} \cdot \sigma^2$.

The remaining solving phase is to solve this transformed LWE instance. This phase does not affect its asymptotic complexity but has significant impact on its actual running time for concrete instances.

Similar to the original proposal [13] for solving LPN, which recovers 1 bit in the secret vector via majority voting, Albrecht et al. [3] exhaust one secret entry using a distinguisher. The complexity is further reduced by Duc et al. [21] using Fast Fourier Transform (FFT) to recover several secret entries simultaneously.

3.2 Coded-BKW

As described above, in each BKW step, we try to collide a large number of vectors $\mathbf{a}_i$ in a set of positions denoted by an index set I. We denote this sub-vector of a vector $\mathbf{a}$ as $\mathbf{a}_I$. We set the size[3] of the collision set to $\frac{q^b-1}{2}$, a very important parameter indicating the final complexity of the algorithm.

[3]Naively the size is q^b. However, using the fact that vectors with subsets $\mathbf{a}_I$ and $-\mathbf{a}_I$ can be mapped into the same category, we can reduce the size to $\frac{q^b-1}{2}$. The zero vector gets its own category.

In this part we describe another idea that, instead of zeroing out the vector $\mathbf{a}_I$ by collisions, we try to collide vectors to make $\mathbf{a}_I$ small. The advantage of this idea is that one can handle more positions in one step for the same size of the collision set.

This idea was first formulated by Albrecht et al. in PKC 2014 [4], aiming for solving the LWE problem with a small secret. They proposed a new technique called Lazy Modulus Switching (LMS). Then, in CRYPTO 2015, two new algorithms with similar underlying algorithmic ideas were proposed independently in [25] and [31], highly enhancing the performance in the sense of both asymptotic and concrete complexity. Using the secret-noise transformation, these new algorithms can be used to solve the standard LWE problem.

In this part we use the notation from [25] to describe the BKW variant called coded-BKW, as it has the best concrete performance, i.e., it can reduce the magnitude of the noise by a constant factor compared with its counterpart technique LMS. The core step – the coded-BKW step – can be described as follows.

Considering step i in the reduction phase, we choose a q-ary $[n_i, b]$ linear code, denoted $\mathcal{C}_i$, that can be employed to construct a lattice code, e.g., using Construction A (see [19] for details). The sub-vector $\mathbf{a}_I$ can then be written in terms of its two constituents, the codeword part $\mathbf{c}_I \in \mathcal{C}_i$ and an error part $\mathbf{e}_I \in \mathbb{Z}_q^{N_i}$. That is,

$$\mathbf{a}_I = \mathbf{c}_I + \mathbf{e}_I. \tag{2}$$

We rewrite the inner product $\langle \mathbf{s}_I, \mathbf{a}_I \rangle$ as

$$\langle \mathbf{s}_I, \mathbf{a}_I \rangle = \langle \mathbf{s}_I, \mathbf{c}_I \rangle + \langle \mathbf{s}_I, \mathbf{e}_I \rangle .$$

We can cancel out the part $\langle \mathbf{s}_I, \mathbf{c}_I \rangle$ by subtracting two vectors mapped to the same codeword, and the remaining difference is the noise. Using the same kind of reasoning, the size of the collision set can be $\frac{q^b-1}{2}$, as in the plain BKW step.

If we remove n_i positions in the i-th step, then we have removed $\sum_{i=1}^{t} n_i$ positions ($n_i \geq b$) in total. Thus, after guessing the remaining secret symbols in the solving phase, we need to distinguish between the uniform distribution and the distribution representing a sum of noise variables, i.e.,

$$z = \sum_{j=1}^{2^t} e_{i_j} + \sum_{i=1}^{n} s_i (E_i^{(1)} + E_i^{(2)} + \cdots + E_i^{(t)}), \tag{3}$$

where $E_i^{(h)} = \sum_{j=1}^{2^{t-h+1}} \hat{e}_{i_j}^{(h)}$ and $\hat{e}_{i_j}^{(h)}$ is the noise introduced in the h-th coded-BKW step. Here at most one error term $E_i^{(h)}$ is non-zero for one position in the index set, and the overall noise can be estimated according to Equation (3).

The remaining problem is to analyze the noise level introduced by coding. In [25], it is assumed that every $E_i^{(h)}$ is close to a Gaussian distribution, which is tested in implementation. Based on known results (e.g., [19]) on lattice codes, in [25], the standard deviation σ introduced by employing a q-ary $[N, k]$ linear code is estimated by

$$\sigma \approx q^{1-k/N} \cdot \sqrt{G(\Lambda_{N,k})}, \tag{4}$$

where $G(\Lambda_{N,k})$ is a code-related parameter satisfying

$$\frac{1}{2\pi e} < G(\Lambda_{N,k}) \leq \frac{1}{12}.$$

In [25], the chosen codes are with varying rates to ensure that the noise contribution of each position is equal. This is principally similar to the operation of changing the modulus size in each reduction step in [31]. It is trivial to get a code with $G(L_{N,k})$ larger than $1/12$. While choosing a better code is important for concrete complexity, for asymptotic complexity this trivial code is as fast as any other code.

4 A Reformulation

Let us reformulate the LWE problem and the steps in the different algorithms in a matrix form. Recall that we have the LWE samples in the form $\mathbf{z} = \mathbf{sA} + \mathbf{e}$. We write this as

$$(\mathbf{s}, \mathbf{e}) \begin{pmatrix} \mathbf{A} \\ \mathbf{I} \end{pmatrix} = \mathbf{z}. \tag{5}$$

The entries in the unknown left-hand side vector $(\mathbf{s}, \mathbf{e})$ are all i.i.d. The matrix above is denoted as $\mathbf{H_0} = \begin{pmatrix} \mathbf{A} \\ \mathbf{I} \end{pmatrix}$ and it is a known quantity, as well as $\mathbf{z}$.

By multiplying Equation (5) from the right with special matrices $\mathbf{P_i}$ we are going to reduce the size of columns in the matrix. Starting with

$$(\mathbf{s}, \mathbf{e})\mathbf{H_0} = \mathbf{z},$$

we find a matrix $\mathbf{P_0}$ and form $\mathbf{H_1} = \mathbf{H_0P_0}$, $\mathbf{z}_1 = \mathbf{zP_0}$, resulting in

$$(\mathbf{s}, \mathbf{e})\mathbf{H_1} = \mathbf{z}_1.$$

Continuing this process for t steps, we have formed $\mathbf{H_t} = \mathbf{H_0P_0} \cdots \mathbf{P_{t-1}}$, $\mathbf{z}_t = \mathbf{zP_0} \cdots \mathbf{P_{t-1}}$.

Here the dimensionality of the $\mathbf{P_i}$ matrices depends on how the total number of samples changes in each step. If we keep the total number of samples constant, then all $\mathbf{P_i}$ matrices have size $m \times m$.

Plain BKW can be described as each $\mathbf{P_i}$ having columns with only two nonzero entries, both from the set $\{-1, 1\}$. The BKW procedure subsequently cancels rows in the $\mathbf{H_i}$ matrices in a way such that $\mathbf{H_t} = \begin{pmatrix} \mathbf{0} \\ \mathbf{H'_t} \end{pmatrix}$, where columns of $\mathbf{H'_t}$ have 2^t non-zero entries[4]. The goal is to minimize the magnitude of the column entries in $\mathbf{H_t}$. The smaller magnitude, the larger advantage in the corresponding samples.

The improved techniques like LMS and coded-BKW reduce the $\mathbf{H_t}$ similar to the BKW, but improve by using the fact that the top rows of $\mathbf{H_t}$ do not have to be canceled to $\mathbf{0}$. Instead, entries are allowed to be of the same norm as in the $\mathbf{H'_t}$ matrix.

[4]Sometimes we get a little fewer than 2^t entries since 1s can overlap. However, this probability is low and does not change the analysis.

Algorithm 1 Coded-BKW with Sieving (main steps)

Input: Matrix $\mathbf{A}$ with n rows and m columns, received vector $\mathbf{z}$ of length m and algorithm parameters $t, n_i, 1 \leq i \leq t$, B

change the distribution of the secret vector (Gaussian elimination)
for i from 1 to t **do**:
  for all columns $\mathbf{h} \in \mathbf{H_{i-1}}$ **do**:
    $\Delta = \text{CodeMap}(\mathbf{h}, i)$
    put $\mathbf{h}$ in list $\mathcal{L}_\Delta$
  for all lists $\mathcal{L}_\Delta$ **do**:
    $\mathcal{S}_\Delta = \text{Sieve}(\mathcal{L}_\Delta, i, \sqrt{N_i} \cdot B)$
    put all $\mathcal{S}_\Delta$ as columns in $\mathbf{H_i}$
guess the $\mathbf{s}_n$ entry using hypothesis testing

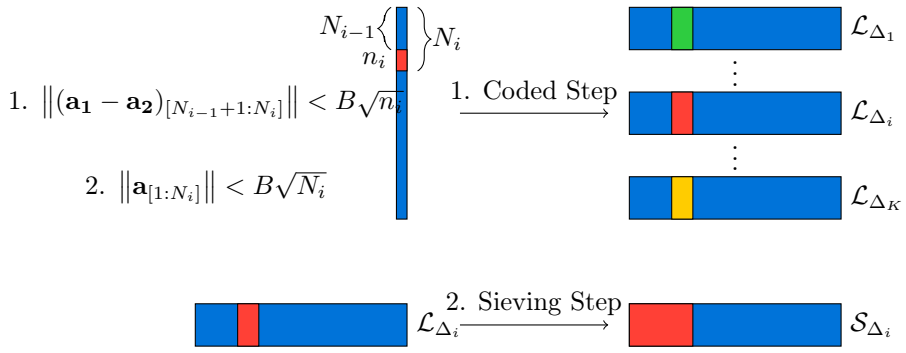


Figure 1: A micro picture of how one step of coded-BKW with sieving works.

5 A BKW-Sieving Algorithm for the LWE Problem

The algorithm we propose uses a similar structure as the coded-BKW algorithm. The new idea involves changing the BKW step to also include a sieving step. In this section we give the algorithm in a simple form, allowing for some asymptotic analysis. We exclude some steps that give non-asymptotic improvements. We assume that each entry in the secret vector $\mathbf{s}$ is distributed according to $\mathcal{X}$.

A summary of coded-BKW with sieving is detailed in Algorithm 1.

Note that one may also use some advanced distinguisher, e.g., the FFT distinguisher, which is important to the concrete complexity, but not for the asymptotic performance.

5.1 Initial Guessing Step

We select a few entries of $\mathbf{s}$ and guess these values (according to $\mathcal{X}$). We run through all likely values and for each of them we do the steps below. Based on a particular guess, the sample equations need to be rewritten accordingly.

For simplicity, the remaining unknown values are still denoted $\mathbf{s}$ after this guessing step and the length of $\mathbf{s}$ is still denoted n.

5.2 Transformation Steps

We start with some simplifying notation. The n positions in columns in $\mathbf{A}$ (first n positions in columns of $\mathbf{H}$) are considered as a concatenation of smaller vectors. We assume that these vectors have lengths which are $n_1, n_2, n_3, \ldots, n_t$, respectively, in such a way that $\sum_{i=1}^{t} n_i = n$. Also, let $N_j = \sum_{i=1}^{j} n_i$, for $j = 1, 2, \ldots, t$.

Before explaining the algorithmic steps, we introduce two notations that will be used later.

Notation CodeMap$(\mathbf{h}, i)$: We assume, following the idea of coded-BKW, that we have fixed a lattice code $\mathcal{C}_i$ of length n_i. The vector $\mathbf{h}$ fed as input to CodeMap is first considered only restricted to the positions $N_{i-1}+1$ to N_i, i.e., as a vector of length n_i. This vector, denoted $\mathbf{h}_{[N_{i-1}+1,N_i]}$, is then mapped to the closest codeword in $\mathcal{C}_i$. This closest codeword is denoted CodeMap$(\mathbf{h}, i)$.

The code $\mathcal{C}_i$ needs to have an associated procedure of quickly finding the closest codeword for any given vector. One could then use a simple code or a more advanced code. From an asymptotic viewpoint, it does not matter, but in a practical implementation there can be a difference. We are going to select the parameters in such a way that the distance to the closest codeword is expected to be no more than $\sqrt{n_i} \cdot B$, where B is a constant.

Notation Sieve$(\mathcal{L}_\Delta, i, \sqrt{N_i} \cdot B)$: The input $\mathcal{L}_\Delta$ contains a list of vectors. We are only considering them restricted to the first N_i positions. This procedure will find differences between any two vectors such that the norm of the difference restricted to the first N_i positions is less than $\sqrt{N_i} \cdot B$. All such differences are put in a list $\mathcal{S}_\Delta$ which is the output of the procedure. On average, the list $\mathcal{S}_\Delta$ should have roughly the same amount of vectors as $\mathcal{L}_\Delta$.

We assume that the vectors in the list $\mathcal{L}_\Delta$ restricted to the first N_i positions, all have a norm of about $\sqrt{N_i} \cdot B$. Then the problem is solved by algorithms for sieving in lattices, for example using Locality-Sensitive Hashing/Filtering.

For the description of the main algorithm, recall that

$$(\mathbf{s}, \mathbf{e})\mathbf{H_0} = \mathbf{z},$$

where $\mathbf{H_0} = \begin{pmatrix} \mathbf{A} \\ \mathbf{I} \end{pmatrix}$. We are going to perform t steps to transform $\mathbf{H_0}$ into $\mathbf{H_t}$ such that the columns in $\mathbf{H_t}$ are "small". Again, we look at the first n positions in a column corresponding to the $\mathbf{A}$ matrix. Since we are only adding or subtracting columns using coefficients in $\{-1, 1\}$, the remaining positions in the column are assumed to contain 2^i nonzero positions either containing a -1 or a 1, after i steps[5].

5.3 A BKW-Sieving Step

We are now going to fix an average level of "smallness" for a position, which is a constant denoted B, as above. The idea of the algorithm is to keep the norm of considered vectors of some length n' below $\sqrt{n'} \cdot B$.

[5]Sometimes we get a little fewer than 2^t entries since 1s can overlap. However, this probability is low and does not change the analysis.

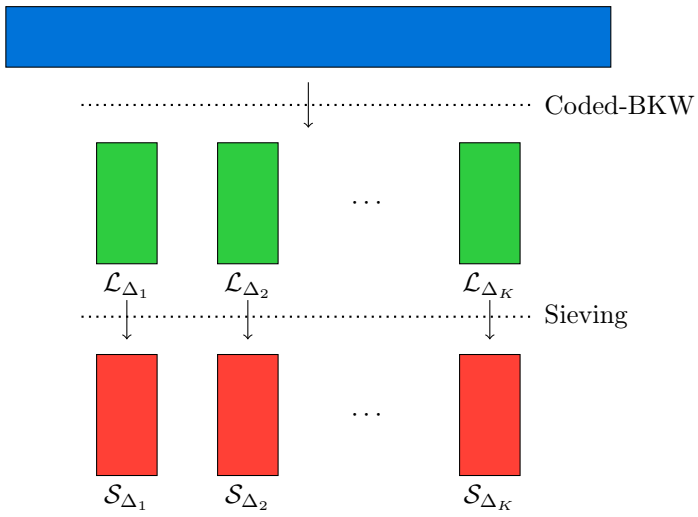


Figure 2: A macro picture of how one step of coded-BKW with sieving works. The set of lists $\mathcal{S}_{\Delta_1}, \ldots, \mathcal{S}_{\Delta_K}$ constitutes the samples for the next step of coded-BKW with sieving.

A column $\mathbf{h} \in \mathbf{H_0}$ will now be processed by first computing $\Delta = \text{CodeMap}(\mathbf{h}, 1)$. Then we place $\mathbf{h}$ in the list $\mathcal{L}_\Delta$. After running through all columns $\mathbf{h} \in \mathbf{H_0}$ they have been sorted into lists $\mathcal{L}_\Delta$. Use K to denote the total number of lists.

We then run through the lists, each containing roughly m/K columns. We perform a sieving step, according to $\mathcal{S}_\Delta = \text{Sieve}(\mathcal{L}_\Delta, \sqrt{N_1} \cdot B)$, for all $\Delta \in \mathcal{C}_i$. The result is a list of vectors, where the norm of each vector restricted to the first N_1 positions is less than $\sqrt{N_1} \cdot B$. The indices of any i_j, i_k are kept in such a way that we can compute a new received symbol $z = z_{i_j} - z_{i_k}$. All vectors in all lists $\mathcal{S}_\Delta$ are now put as columns in $\mathbf{H_1}$. We now have a matrix $\mathbf{H_1}$ where the norm of each column restricted to the first n_1 positions is less than $\sqrt{N_1} \cdot B$. This is the end of the first step.

Next, we repeat roughly the same procedure another $t-1$ times. A column $\mathbf{h} \in \mathbf{H_{i-1}}$ will now be processed by first computing $\Delta = \text{CodeMap}(\mathbf{h}, i)$. We place $\mathbf{h}$ in the list $\mathcal{L}_\Delta$. After running through all columns $\mathbf{h} \in \mathbf{H_{i-1}}$ they have been sorted in K lists $\mathcal{L}_\Delta$.

We run through all lists, where each list contains roughly m/K columns. We perform a sieving step, according to $\mathcal{S}_\Delta = \text{Sieve}(\mathcal{L}_\Delta, i, \sqrt{N_i} \cdot B)$. The result is a list of vectors where the norm of each vector restricted to the first N_i positions is less than $\sqrt{N_i} \cdot B$. A new received symbol is computed. All vectors in all lists $\mathcal{S}_\Delta$ are now put as columns in $\mathbf{H_i}$. We get a matrix $\mathbf{H_i}$ where the norm of each column restricted to the first N_i positions is less than $\sqrt{N_i} \cdot B$. This is repeated for $i = 2, \ldots, t$. We assume that the parameters have been chosen in such a way that each matrix $\mathbf{H_i}$ can have m columns.

We make the following heuristic assumption.

Heuristic 2. After each step i of coded-BKW with sieving, we make the assumption that the vectors of $\mathcal{L}_\Delta$ are uniformly distributed on a sphere with

radius $\sqrt{N_i} \cdot B$.

This is a standard assumption[6] in papers on lattice sieving.

After performing these t steps we end up with a matrix $\mathbf{H_t}$ such that the norm of columns restricted to the first n positions is bounded by $\sqrt{n} \cdot B$ and the norm of the last m positions is roughly $2^{t/2}$. Altogether, this should result in samples generated as

$$\mathbf{z} = (\mathbf{s}, \mathbf{e})\mathbf{H_t}.$$

The values in the $\mathbf{z}$ vector are then roughly Gaussian distributed, with variance $\sigma^2 \cdot (nB^2 + 2^t)$. By running a distinguisher on the created samples $\mathbf{z}$ we can verify whether our initial guess is correct or not. After restoring some secret value, the whole procedure can be repeated, but for a smaller dimension.

5.4 Illustrations of Coded-BKW with Sieving

A micro picture of how coded-BKW with sieving works can be found in Fig. 1. A sample gets mapped to the correct list $\mathcal{L}_{\Delta_i}$, based on the current n_i positions. Here $\mathcal{L}_{\Delta_1}, \ldots, \mathcal{L}_{\Delta_K}$ denote the set of all the K such lists. In this list $\mathcal{L}_{\Delta_i}$, when adding/subtracting two vectors the resulting vector gets elements that are on average smaller than B in magnitude in the current n_i positions. Then we only add/subtract vectors in the list $\mathcal{L}_{\Delta_i}$ such that the elements of the resulting vector on average is smaller than B in the first N_i positions. The list of such vectors is then denoted $\mathcal{S}_{\Delta_i}$.

A macro picture of how coded-BKW with sieving works can be found in Fig. 2. The set of all samples gets divided up into lists $\mathcal{L}_{\Delta_1}, \ldots, \mathcal{L}_{\Delta_K}$. Sieving is then applied to each individual list. The resulting sieved lists $\mathcal{S}_{\Delta_1}, \ldots, \mathcal{S}_{\Delta_K}$ then constitute the set of samples for the next step of coded-BKW with sieving.

5.5 High-Level Comparison with Previous BKW Versions

A high-level comparison between the behaviors of plain BKW, coded-BKW and coded-BKW with sieving is shown in Fig. 3.

Initially the average norm of all elements in a sample vector $\mathbf{a}$ is around $q/4$, represented by the first row in the figure. Plain BKW then gradually works towards a zero vector by adding/subtracting vectors in each step such that a fixed number of positions gets canceled out to 0.

The idea of coded-BKW is to not cancel out the positions completely, and thereby allow for longer steps. The positions that are not canceled out increase in magnitude by a factor of $\sqrt{2}$ in each step. To end up with an evenly distributed noise vector in the end we can let the noise in the new almost canceled positions increase by a factor of $\sqrt{2}$ in each step. Thus we can gradually increase the step size.

When reducing positions in coded-BKW, the previously reduced positions increase in magnitude by a factor of $\sqrt{2}$. However, the sieving step in coded-BKW with sieving makes sure that the previously reduced positions do not increase in magnitude. Thus, initially, we do not have to reduce the positions as much as in coded-BKW. However, the sieving process gets more expensive

[6] For finding the shortest vector in a lattice, the problem gets easier using this assumption. However, in our case, if the distribution of vectors is biased, it gets easier to find pairs of vectors close to each other.

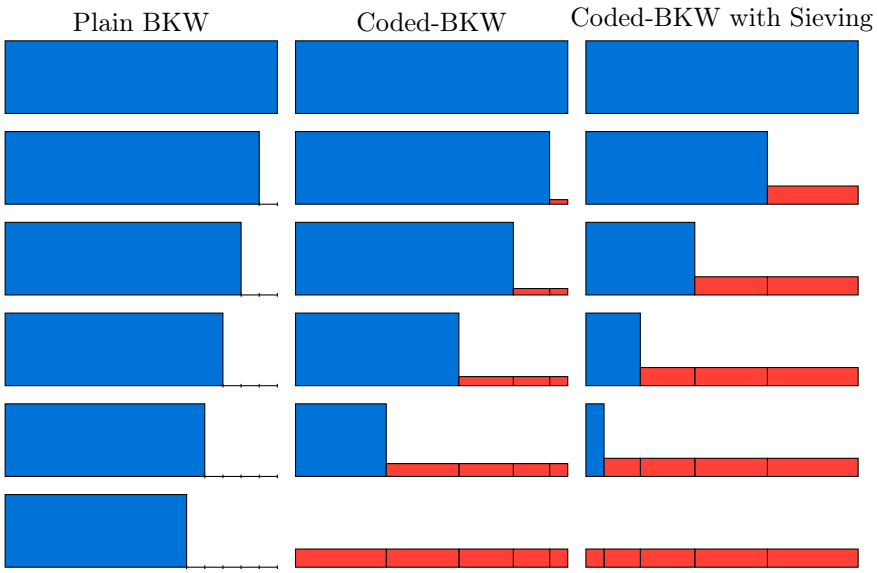


Figure 3: A high-level illustration of how the different versions of the BKW algorithm work. The x-axis represents positions in the $\mathbf{a}$ vector, and the y-axis depicts the average absolute value of the corresponding position. The blue color corresponds to positions that have not been reduced yet and the red color corresponds to reduced positions.

the more positions we work with, and we must therefore gradually divide our samples into fewer buckets to not increase the total cost of the later steps. Thus, we must gradually decrease the step size.

6 Parameter Selection and Asymptotic Analysis

After each step, positions that already have been treated should remain at some given magnitude B. That is, the average (absolute) value of a treated position should be very close to B. This property is maintained by the way in which we apply the sieving part at each reduction step. After t steps we have therefore produced vectors of average norm $\sqrt{n} \cdot B$.

Assigning the number of samples to be $m = 2^k$, where 2^k is a parameter that will decide the total complexity of the algorithm, we will end up with roughly $m = 2^k$ samples after t steps. As already stated, these received samples will be roughly Gaussian with variance $\sigma^2 \cdot (nB^2 + 2^t)$. We assume that the best strategy is to keep the magnitudes of the two different contributions of the same order, so we choose $nB^2 \approx 2^t$.

Furthermore, using Equation (3), in order to be able to recover a single secret position using m samples, we need

$$m = \mathcal{O}\left(e^{4\pi^2 \cdot \frac{\sigma^2 \cdot (nB^2 + 2^t)}{q^2}}\right).$$

Thus, we have

$$\ln 2 \cdot k = 4\pi^2 \cdot \frac{\sigma^2 \cdot (nB^2 + 2^t)}{q^2} + \mathcal{O}(1). \tag{6}$$

Each of the t steps should deliver $m = 2^k$ vectors of the form described before.

Since we have two parts in each reduction step, we need to analyze these parts separately. First, consider performing the first part of reduction step number i using coded-BKW with an $[n_i, d_i]$ linear code, where the parameters n_i and d_i at each step are chosen for optimal (global) performance. We sort the 2^k vectors into $K = \frac{q^{d_i}-1}{2}$ different lists. Here the coded-BKW step guarantees that all the vectors in a list, restricted to the n_i considered positions, have an average norm less than $\sqrt{n_i} \cdot B$ if the codeword is subtracted from the vector. So the number of lists $\frac{q^{d_i}-1}{2}$ has to be chosen so that this norm restriction is true. Then, after the coded-BKW step, the sieving step should leave the average norm over the N_i positions unchanged, i.e., less than $\sqrt{N_i} \cdot B$.

Since all vectors in a list can be considered to have norm $\sqrt{N_i} \cdot B$ in these N_i positions, the sieving step needs to find any pair that leaves a difference between two vectors of norm at most $\sqrt{N_i} \cdot B$. Using Heuristic 2, we know that a single list should contain at least $2^{0.208N_i}$ vectors to be able to produce the same number of vectors. The time and space complexity is $2^{0.292N_i}$ if LSF is employed.

Let us adopt some further notation. As we expect the number of vectors to be exponential we write $k = c_0 n$ for some c_0. Also, we adopt $q = n^{c_q}$ and $\sigma = n^{c_s}$. By choosing $nB^2 \approx 2^t$, from (6) we derive that

$$B = \Theta(n^{c_q - c_s}) \tag{7}$$

and

$$t = (2(c_q - c_s) + 1) \log_2 n + \mathcal{O}(1). \tag{8}$$

6.1 Asymptotics of Coded-BKW with Sieving

We assume exponential overall complexity and write it as 2^{cn} for some coefficient c to be determined. Each step is additive with respect to complexity, so we assume that we can use 2^{cn} operations in each step. In the t steps we are choosing $n_1, n_2, \ldots$ positions for each step.

The number of buckets needed for the first step of coded-BKW is $(C' \cdot n^{c_s})^{n_1}$, where C' is another constant. In each bucket the dominant part in the time complexity is the sieving cost $2^{\lambda n_1}$, for a constant λ. The overall complexity, the product of these expressions, should match the bound 2^{cn}, and thus we choose n_1 such that $(C' \cdot n^{c_s})^{n_1} \approx 2^{cn} \cdot 2^{-\lambda n_1}$.

Taking the log, $c_s \log n \cdot n_1 + \log C' n_1 = cn - \lambda n_1$. Therefore, we obtain

$$n_1 = \frac{cn}{c_s \log n + \lambda + \log C'}.$$

To simplify expressions, we use the notation $W = c_s \log n + \lambda + \log C'$.

For the next step, we get $W \cdot n_2 = cn - \lambda n_1$, which simplifies in asymptotic sense to

$$n_2 = \frac{cn}{W}\left(1 - \frac{\lambda}{W}\right).$$

Continuing in this way, we have $W \cdot n_i = cn - \lambda \sum_{j=1}^{i-1} n_j$ and we can obtain an asymptotic expression for n_i as

$$n_i = \frac{cn}{W}\left(1 - \frac{\lambda}{W}\right)^{i-1}.$$

After t steps we have $\sum_{i=1}^{t} n_i = n$, so we observe that

$$\sum_{i=1}^{t} n_i = \frac{cn}{W}\sum_{i=1}^{t}\left(1 - \frac{\lambda}{W}\right)^{i-1},$$

which simplifies to

$$n = \sum_{i=1}^{t} n_i = \frac{cn}{\lambda}\left(1 - \left(1 - \frac{\lambda}{W}\right)^{t}\right).$$

Now, we know that

$$c = \lambda\left(1 - \left(1 - \frac{\lambda}{W}\right)^{t}\right)^{-1}.$$

Since t and W are both of order $\Theta(\log n)$ that tend to infinity as n tends to infinity, we have that

$$c = \lambda\left(1 - \left(1 - \frac{\lambda}{W}\right)^{\frac{W}{\lambda}\cdot\frac{t\lambda}{W}}\right)^{-1} \to \lambda\left(1 - e^{-\frac{t\lambda}{W}}\right)^{-1},$$

when $n \to \infty$.

Since $t/W \to (1 + 2(c_q - c_s))/c_s$ when $n \to \infty$ this finally gives us

$$c = \frac{\lambda}{1 - e^{-\lambda(1+2(c_q-c_s))/c_s}}.$$

Then we obtain the following theorem.

Theorem 6.1. *Under Heuristics 1 and 2, the time and space complexity of the proposed algorithm is* $2^{(c+o(1))n}$*, where*

$$c = \frac{\lambda}{1 - e^{-\lambda(1+2(c_q-c_s))/c_s}},$$

and $\lambda = 0.292$ *for classic computers and* 0.265 *for quantum computers.*

Proof 6.1. *Since* $c > \lambda$*, there are exponential samples left for the distinguishing process. One can adjust the constants in* (7) *and* (8) *to ensure a success probability of hypothesis testing close to* 1. □

6.2 Asymptotics when Using Plain BKW Pre-Processing

In this section we show that Theorem 6.1 can be improved for certain LWE parameters. Suppose that we perform t_0 plain BKW steps and t_1 steps of coded-BKW with sieving, so $t = t_0 + t_1$. We first derive the following Lemma 6.1.

Lemma 6.1. *It is asymptotically beneficial to perform t_0 plain BKW steps, where t_0 is of order $(2(c_q - c_s) + 1 - c_s/\lambda \cdot \ln(c_q/c_s)) \log n$, if*

$$\frac{c_s}{\lambda} \ln \frac{c_q}{c_s} < 2(c_q - c_s) + 1.$$

Proof 6.2. *Suppose in each plain BKW step, we zero-out b positions. Therefore, we have that*

$$q^b = 2^{cn+o(n)},$$

and it follows that asymptotically

$$b = \frac{cn}{c_q \log n} + o\left(\frac{n}{\log n}\right). \tag{9}$$

Because the operated positions in each step will decrease using coded-BKW with sieving, it is beneficial to replace a step of coded-BKW with sieving by a pre-processing step of plain BKW, if the allowed number of steps is large. We compute t_1 such that for $t \geq i \geq t_1$, we have $n_i \leq b$. That is,

$$\frac{cn}{W}\left(1 - \frac{\lambda}{W}\right)^{t_1 - 1} = \frac{cn}{c_q \log n}.$$

Thus, we derive that t_1 is of order $c_s/\lambda \cdot \ln(c_q/c_s) \cdot \log n$. □

If we choose $t_0 = t - t_1$ plain BKW steps, where t_1 is of order $c_s/\lambda \cdot \ln(c_q/c_s) \cdot \log n$ as in Lemma 6.1, then

$$n - t_0 b = \sum_{i=1}^{t_1} n_i = \frac{cn}{\lambda}\left(1 - \left(1 - \frac{\lambda}{W}\right)^{t_1}\right).$$

Thus

$$1 - \frac{c}{c_q}\left(2(c_q - c_s) + 1 - \frac{c_s}{\lambda} \ln\left(\frac{c_q}{c_s}\right)\right) = \frac{c}{\lambda}\left(1 - \frac{c_s}{c_q}\right).$$

Finally, making the same heuristic assumptions as in Theorem 6.1, we have the following theorem for characterizing its asymptotic complexity.

Theorem 6.2. *Under Heuristics 1 and 2, if $c > \lambda$ and $\frac{c_s}{\lambda} \ln \frac{c_q}{c_s} < 2(c_q - c_s) + 1$, then the time and space complexity of the proposed algorithm with plain BKW pre-processing is $2^{(c+o(1))n}$, where*

$$c = \frac{\lambda c_q}{(1 + 2\lambda)(c_q - c_s) + \lambda - c_s \ln\left(\frac{c_q}{c_s}\right)},$$

and $\lambda = 0.292$ for classic computers and 0.265 for quantum computers.

Proof 6.3. *The proof is similar to that of Theorem 6.1.* □

Algorithm	Complexity exponent (c)
QS-BKW(w/ p)	0.8856
S-BKW(w/ p)	0.8951
S-BKW(w/o p)	0.9054
Coded-BKW [25, 31]	0.9299
DUAL-POLYSamples [27]	4.6720
DUAL-EXPSamples [27]	1.1680

Table 1: Asymptotic complexity for the Regev parameters

6.3 Case Study: Asymptotic Complexity of the Regev Parameters

In this part we present a case-study on the asymptotic complexity of Regev parameter sets, a family of LWE instances with significance in public-key cryptography.

Regev parameters: We pick parameters $q \approx n^2$ and $\sigma = n^{1.5}/(\sqrt{2\pi}\log_2^2 n)$ as suggested in [45].

The asymptotic complexity of Regev's LWE instances is shown in Table 1. For this parameter set, we have $c_q = 2$ and $c_s = 1.5$, and the previously best algorithms in the asymptotic sense are the coded-BKW variants [25, 31] (denoted Coded-BKW in this table) with time complexity $2^{0.9299n+o(n)}$. The item DUAL-POLYSamples represents the run time exponent of lattice reduction approaches using polynomial samples and exponential memory, while DUAL-EXPSamples represents the run time exponent of lattice reduction approaches using exponential samples and memory. Both values are computed according to formulas from [27], i.e., $2c_{BKZ} \cdot c_q/(c_q - c_s)^2$ and $2c_{BKZ} \cdot c_q/(c_q - c_s + 1/2)^2$, respectively. Here c_{BKZ} is chosen to be 0.292, the best constant that can be achieved heuristically [9].

We see from the table that the newly proposed algorithm coded-BKW with sieving outperforms the previous best algorithms asymptotically. For instance, the simplest strategy without plain BKW pre-processing, denoted S-BKW(w/o p), costs $2^{0.9054n+o(n)}$ operations, with pre-processing, the time complexity, denoted S-BKW(w/ p) is $2^{0.8951n+o(n)}$. Using quantum computers, the constant hidden in the exponent can be further reduced to 0.8856, shown in Table 1 as QS-BKW(w/ p). Note that the exponent of the lattice approach is much higher than that of the BKW variants for the Regev parameters.

6.4 A Comparison with the Asymptotic Complexity of Other Algorithms

A comparison between the asymptotic time complexity of coded-BKW with sieving and the previous best single-exponential algorithms is shown in Fig. 4, similar to the comparison made in [28]. The upper and lower picture show the state-of-the-art algorithms before and after coded-BKW with sieving was introduced. We use pre-processing with standard BKW steps (see Theorem 6.2), since that reduces the complexity of the coded-BKW with sieving algorithm for

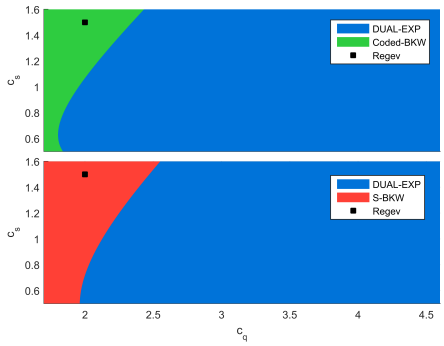


Figure 4: A comparison of the asymptotic behavior of the best single-exponential algorithms for solving the LWE problem for different values of c_q and c_s. The different areas show where in the parameter space the corresponding algorithm beats the other algorithms in that subplot.

the entire plotted area. Use of exponential space is assumed. Access to an exponential number of samples is also assumed.

First of all we notice that coded-BKW with sieving beats coded-BKW for all the parameters in the figure. It also outperforms the dual algorithm with an exponential number of samples on some areas where that algorithm used to be the best. It is also worth mentioning that the Regev instances are well within the area where coded-BKW with sieving performs best.

We have omitted the area where $c_s < 0.5$ in Fig. 4 and the subsequent figures. Here the Arora-Ge algorithm [8] is polynomial. This area is not particularly interesting in cryptographical terms, since Regev's reduction proof does not apply for $c_s < 0.5$.

7 Asymptotic Complexity of LWE with Sparser Secrets

In this part, we discuss the asymptotic solving complexity of an LWE variant whose secret symbols are sampled from a distribution with standard deviation $n^{c_{s_1}}$ and the error distribution is a discrete Gaussian with standard deviation $n^{c_{s_2}}$, where $0 < c_{s_1} < c_{s_2}$. We make the same heuristic assumptions as in Theorem 6.1 in the derivations in this section. One important application is the LWE problem with a polynomial number of samples, where c_{s_1} equals c_s, while c_{s_2} changes to

$$\begin{cases} c_s + \frac{1}{2} & \text{if we start with } \Theta(n \log n) \text{ samples,} \\ c_s + \frac{1}{2} + \frac{c_q}{c_m - 1} & \text{if we start with } \Theta(c_m n) \text{ samples,} \end{cases}$$

after the secret-noise transform and the sample amplification procedure (cf. [28]).

We assume that the best strategy is to choose $nB^2 n^{2c_{s_1}} \approx 2^t n^{2c_{s_2}}$. Therefore, we know that $B = C \cdot n^{c_q - c_{s_1}}$ and $t = \log_2 D + (2(c_q - c_{s_2}) + 1) \cdot \log_2 n$, for some constants C and D. We then derive similar formulas except that now $W = c_{s_1} \cdot \log n + o(\log n)$.

We have the following theorem.

Theorem 7.1. *Under Heuristics 1 and 2, the time and space complexity of the proposed algorithm for solving the* LWE *problem with sparse secrets is* $2^{(c+o(1))n}$, *where*

$$c = \frac{\lambda}{1 - e^{-\lambda\left(1+2\left(c_q - c_{s_2}\right)\right)/c_{s_1}}},$$

and $\lambda = 0.292$ *for classic computers and* 0.265 *for quantum computers.*

Lemma 7.1. *It is asymptotically beneficial to perform* t_0 *plain BKW steps, where* t_0 *is of order* $\left(2\left(c_q - c_{s_1}\right) + 1 - c_{s_1}/\lambda \cdot \ln\left(c_q/c_{s_1}\right)\right)\log n$, *if*

$$\frac{c_{s_1}}{\lambda} \ln \frac{c_q}{c_{s_1}} < 2\left(c_q - c_{s_1}\right) + 1.$$

If we choose $t_0 = t - t_1$ plain BKW steps, where t_1 is of order $c_{s_1}/\lambda \cdot \ln\left(c_q/c_{s_1}\right) \cdot \log n$ as in Lemma 7.1, then

$$n - t_0 b = \sum_{i=1}^{t_1} n_i = \frac{cn}{\lambda}\left(1 - \left(1 - \frac{\lambda}{W}\right)^{t_1}\right).$$

Thus

$$1 - \frac{c}{c_q}\left(2\left(c_q - c_{s_2}\right) + 1 - \frac{c_{s_1}}{\lambda} \ln\left(\frac{c_q}{c_{s_1}}\right)\right) = \frac{c}{\lambda}\left(1 - \frac{c_{s_1}}{c_q}\right).$$

Finally, we have the following theorem.

Theorem 7.2. *Under Heuristics 1 and 2, if* $c > \lambda$ *and* $\frac{c_{s_1}}{\lambda} \ln \frac{c_q}{c_{s_1}} < 2\left(c_q - c_{s_1}\right) + 1$, *then the time and space complexity of the proposed algorithm with plain BKW pre-processing for solving the* LWE *problem with sparse secrets is* $2^{(c+o(1))n}$, *where* c *is*

$$\frac{\lambda c_q}{\left(c_q - c_{s_1}\right) + \lambda\left(2\left(c_q - c_{s_2}\right) + 1\right) - c_{s_1} \ln\left(\frac{c_q}{c_{s_1}}\right)},$$

and $\lambda = 0.292$ *for classic computers and* 0.265 *for quantum computers.*

7.1 Asymptotic Complexity of LWE with a Polynomial Number of Samples

Algorithm	Complexity exponent (c)
QS-BKW	1.6364
S-BKW	1.6507
Coded-BKW [25, 31]	1.7380
DUAL-POLYSamples [27]	4.6720

Table 2: Asymptotic complexity for the Regev parameters with a polynomial number of samples

Applying Theorems 7.1 and 7.2 to the case where we limit the number of samples to $\Theta(n \log n)$ gives us the comparison of complexity exponents for the

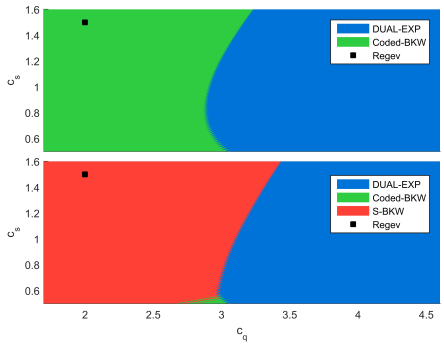


Figure 5: A comparison of the asymptotic behavior of the best single-exponential algorithms for solving the LWE problem for different values of c_q and c_s. The different areas show where in the parameter space the corresponding algorithm beats the other algorithms in that subplot. The number of samples is limited to $\Theta(n \log n)$.

Regev parameters in Table 2. Notice that, asymptotically speaking, the BKW algorithms perform much better compared to the lattice reduction counterparts in this scenario. Also, since pre-processing with plain BKW steps does not lower the complexity in the polynomial case, we just call the algorithms S-BKW and QS-BKW.

In Fig. 5 we compare the asymptotic behavior between the different algorithms for varying values of c_q and c_s, when the number of samples is limited to $\Theta(n \log n)$. The upper and lower picture show the state-of-the-art algorithms before and after coded-BKW with sieving was introduced. Notice here that the area where the BKW algorithms perform better is much larger than in the case with exponential number of samples. Coded-BKW is best only in a very small area.

8 New Variants of Coded-BKW with Sieving

We come back to the general LWE problem (without a limit on the number of samples). In this section, we present three novel variants, the first two showing unified views for the existing BKW algorithms, and the other improving the asymptotic complexity for solving many LWE instances including the important Regev ones, both classically and in a quantum setting. We make the same heuristic assumptions as in Theorem 6.1 in the derivations in this section.

The results can easily be extended to the solving of LWE problems with sparse secrets, by replacing c_{s_2} below by $c_s + 1/2$, like we did in Section 7. The improvements in the sparse case are similar to the ones we show below, so for ease of reading we omit this analysis.

To balance the noise levels for the best performance, we always perform $t = (2(c_q - c_s) + 1) \log_2 n + \mathcal{O}(1)$ reduction steps and make the noise in each position approximately equal to $B = \Theta(n^{c_q - c_s})$. For the t reduction steps, we have three different choices, i.e., plain BKW, coded-BKW, and coded-BKW with sieving. We assume that plain BKW steps (if performed) should be done

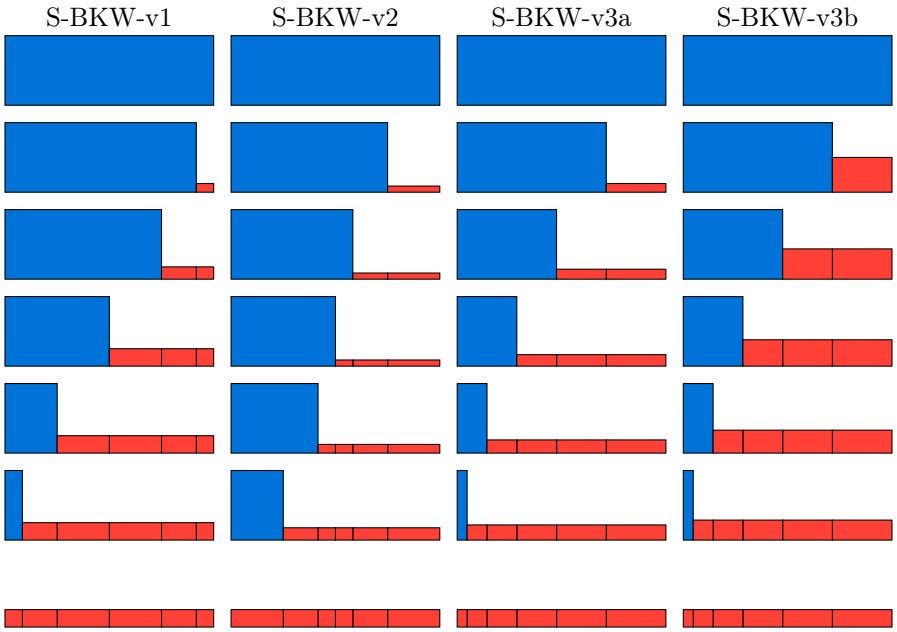


Figure 6: A high-level illustration of how the different new variants of coded-BKW with sieving work. The x-axis represents positions in the $\mathbf{a}$ vector, and the y-axis depicts the average absolute value of the corresponding position. The blue color corresponds to positions that have not been reduced yet and the red color corresponds to reduced positions. Notice that the two rightmost columns both correspond to the same version of the algorithm, but with $\gamma > 1$ and $\gamma < 1$ respectively.

before the other two options[7].

8.1 S-BKW-v1

We start with a simple procedure (named S-BKW-v1), i.e., first performing t_1 plain BKW steps, then t_2 coded-BKW steps, and finally t_3 steps of coded-BKW with sieving. Thus,

$$t_1 + t_2 + t_3 = t = (2(c_q - c_s) + 1) \log_2 n + \mathcal{O}(1),$$

and we denote that $t_3 = \alpha \log n + \mathcal{O}(1)$, $t_2 = \beta \log n + \mathcal{O}(1)$, and $t_1 = (2(c_q - c_s) + 1 - \alpha - \beta) \log n + \mathcal{O}(1)$. A straight-forward constraint is that

$$0 \leq \alpha, \beta \leq \alpha + \beta \leq 2(c_q - c_s) + 1.$$

This is a rather generic algorithm as all known BKW variants, i.e., plain BKW, coded-BKW, coded-BKW with sieving (with or without plain BKW pre-

[7] Assume that we apply coded-BKW or coded-BKW with sieving to the first steps and then plain BKW steps to the last steps. The noise corresponding to the first positions would then increase by a factor of $\sqrt{2}$ for each plain BKW step we take. Reversing the order, performing the plain BKW steps first and finishing with the coded-BKW/coded-BKW with sieving steps, we do not see the same increase in noise.

processing), can be treated as specific cases obtained by tweaking the parameters t_1, t_2 and t_3.

We want to make the noise variances for each position equal, so we set

$$B_i = \frac{B}{\sqrt{2^{t_3+i}}},$$

for $i = 1, \ldots, t_2$, where $2B_i$ is the reduced noise interval after $(t_2 - i + 1)$-th coded-BKW steps.

Let m_i be the length of the $(t_2 - i + 1)$-th coded-BKW step. We have that,

$$\left(\frac{q}{B_i}\right)^{m_i} \approx 2^{cn},$$

which simplifies to

$$cn = m_i(c_s \log n + \frac{t_3 + i}{2} + C_0).$$

Thus,

$$m_i = \frac{cn}{(c_s + \frac{\alpha}{2}) \log n + \frac{i}{2} + C_1}, \tag{10}$$

where C_1 is another constant. We know that

$$\sum_{i=1}^{t_2} m_i = 2cn \cdot \ln \frac{c_s + \frac{\alpha+\beta}{2}}{c_s + \frac{\alpha}{2}} + o(n). \tag{11}$$

Let n_i be the length of the i-th step of coded-BKW with sieving, for $1 \leq i \leq t_3$. We derive that

$$n_i \approx \frac{cn}{c_s \log n} \exp(-\frac{i}{c_s \log n}\lambda).$$

Therefore,

$$\sum_{i=1}^{t_3} n_i = \frac{cn}{\lambda}(1 - \exp(-\frac{\alpha}{c_s}\lambda)) + o(n). \tag{12}$$

We then have the following theorem.

Theorem 8.1. *Under Heuristics 1 and 2, one (c_q, c_s) LWE instance can be solved with time and memory complexity $2^{(c+o(1))n}$, where c is the solution to the following optimization problem*

$$\begin{aligned}
\underset{\alpha,\beta}{\textbf{\textit{minimize}}} \quad & c(\alpha, \beta) = (\frac{2(c_q - c_s) + 1 - \alpha - \beta}{c_q} \\
& + 2 \ln \frac{c_s + \frac{\alpha+\beta}{2}}{c_s + \frac{\alpha}{2}} + \lambda^{-1}(1 - \exp(-\frac{\alpha}{c_s}\lambda)))^{-1} \\
\textbf{\textit{subject to}} \quad & 0 \leq \alpha, \beta \leq 2(c_q - c_s) + 1, \\
& \alpha + \beta \leq 2(c_q - c_s) + 1.
\end{aligned}$$

Proof 8.1. *Since $n = t_1 b + \sum_{i=1}^{t_2} m_i + \sum_{i=1}^{t_3} n_i$, we have*

$$\begin{aligned}
1 = & c(\frac{2(c_q - c_s) + 1 - \alpha - \beta}{c_q} + 2 \ln \frac{c_s + \frac{\alpha+\beta}{2}}{c_s + \frac{\alpha}{2}} \\
& + \lambda^{-1}(1 - \exp(-\frac{\alpha}{c_s}\lambda))),
\end{aligned}$$

where b is obtained from (9).

Example 1. For the Regev parameters, i.e., $(c_q, c_s) = (2, 1.5)$, we derive that $\beta = 0$ for the best asymptotic complexity of S-BKW-v1. Thus, in this scenario, this generic procedure degenerates to coded-BKW with sieving using plain BKW processing discussed in Section 6.2, i.e., including no coded-BKW steps.

8.2 S-BKW-v2

Next, we present a variant (named S-BKW-v2) of coded-BKW with sieving by changing the order of the different BKW reduction types in S-BKW-v1. We first do t_1 plain BKW steps, then t_2 coded-BKW with sieving steps, and finally t_3 coded-BKW steps. Similarly, we let $t_3 = \alpha \log n + \mathcal{O}(1)$, $t_2 = \beta \log n + \mathcal{O}(1)$, and $t_1 = (2(c_q - c_s) + 1 - \alpha - \beta) \log n + \mathcal{O}(1)$. We also have the constraint

$$0 \leq \alpha, \beta \leq \alpha + \beta \leq 2(c_q - c_s) + 1.$$

This is also a generic framework including all known BKW variants as its special cases.

Let m_i represent the length of the $(t_3 - i + 1)$-th coded-BKW step, for $1 \leq i \leq t_3$, and n_j the length of the j-th step of coded-BKW step with sieving, for $1 \leq j \leq t_2$.

We derive that,

$$m_1 = \frac{cn}{c_s \log n} + o\left(\frac{n}{\log n}\right),$$

$$m_{t_3} = \frac{cn}{\left(c_s + \frac{\alpha}{2}\right) \log n} + o\left(\frac{n}{\log n}\right),$$

$$\sum_{i=1}^{t_3} m_i = 2cn \cdot \ln \frac{c_s + \frac{\alpha}{2}}{c_s} + o(n),$$

$$n_{t_2} = \frac{cn}{(c_s + \frac{\alpha}{2}) \log n} \exp\left(-\frac{\beta}{c_s}\lambda\right) + o\left(\frac{n}{\log n}\right),$$

$$\sum_{j=1}^{t_2} n_j = \frac{cc_s \cdot n}{\lambda(c_s + \frac{\alpha}{2})} \left(1 - \exp\left(-\frac{\beta\lambda}{c_s}\right)\right) + o(n).$$

We then have the following theorem.

Theorem 8.2. *Under Heuristics 1 and 2, one (c_q, c_s) LWE instance can be solved with time and memory complexity $2^{(c+o(1))n}$, where c is the solution to the following optimization problem*

$$\begin{aligned} \underset{\alpha,\beta}{\textbf{\textit{minimize}}} \quad & c(\alpha, \beta) = \left(\frac{c_s}{\lambda\left(c_s + \frac{\alpha}{2}\right)} \left(1 - \exp\left(-\frac{\beta\lambda}{c_s}\right)\right) + \right. \\ & \left. 2 \ln \frac{c_s + \frac{\alpha}{2}}{c_s} + \frac{1}{c_q} \left(2\left(c_q - c_s\right) + 1 - \alpha - \beta\right) \right)^{-1} \\ \textbf{\textit{subject to}} \quad & 0 \leq \alpha, \beta \leq 2(c_q - c_s) + 1, \\ & \alpha + \beta \leq 2(c_q - c_s) + 1. \end{aligned}$$

Proof 8.2. *The proof is similar to that of Theorem 8.1.*

Example 2. For Regev parameters, we derive that $\alpha = 0$ for the best asymptotic complexity of S-BKW-v2, so it also degenerates to coded-BKW with sieving using plain BKW processing discussed in Section 6.2.

8.3 S-BKW-v3

We propose a new variant (named S-BKW-v3) including a nearest neighbor searching algorithm after a coded-BKW step, which searches for a series of new vectors whose norm is smaller with a factor of γ, where $0 \leq \gamma \leq \sqrt{2}$, by adding or subtracting two vectors in a ball. This is a generalization of coded-BKW and coded-BKW with sieving from another perspective, since coded-BKW can be seen as S-BKW-v3 with reduction parameter $\gamma = \sqrt{2}$, and coded-BKW with sieving as S-BKW-v3 with reduction parameter $\gamma = 1$.

We denote the complexity exponent for the nearest neighbor searching algorithm λ, i.e., $2^{\lambda n + o(n)}$ time and space is required if the dimension is n. We can improve the asymptotic complexity for the Regev parameters further.

We start by performing t_1 plain BKW steps and then t_2 steps of coded-BKW with sieving using parameters (λ, γ). Let $t_2 = \alpha \log n + \mathcal{O}(1)$ and $t_1 = t - t_2 = (2(c_q - c_s) + 1 - \alpha) \log n + \mathcal{O}(1)$. We also have the constraint that $0 \leq \alpha \leq 2(c_q - c_s) + 1$.

Let n_1 be the length of the first step of coded-BKW with sieving. We have $B_1 = B/\gamma^{t_2}$, so

$$(t_2 \log \gamma + c_s \log n) \cdot n_1 = cn - \lambda n_1.$$

Thus,

$$\begin{aligned} n_1 &= \frac{cn}{(c_s + \alpha \log \gamma) \log n + C} \\ &= \frac{cn}{(c_s + \alpha \log \gamma) \log n} \cdot \left(1 + \Theta\left(\log^{-1} n\right)\right), \end{aligned}$$

where C is a constant.

For the i-th step of coded-BKW with sieving, we derive that

$$((t_2 - i + 1) \log \gamma + c_s \log n) \cdot n_i = cn - \lambda \sum_{j=1}^{i} n_j. \tag{13}$$

Thus,

$$n_i = \left(1 + \frac{\log \gamma - \lambda}{(t_2 - i + 1) \log \gamma + c_s \log n + \lambda}\right) \cdot n_{i-1}$$

and if $\gamma \neq 1$, we have that

$$\begin{aligned}
n_{t_2} &= \prod_{i=2}^{t_2} \left(1 + \frac{\log\gamma - \lambda}{(t_2 - i + 1)\log\gamma + c_s \log n + \lambda}\right) \cdot n_1 \\
&= n_1 \cdot \exp\left(\sum_{i=2}^{t_2} \ln\left(\frac{\log\gamma - \lambda}{(t_2 - i + 1)\log\gamma + c_s \log n + \lambda}\right.\right. \\
&\quad \left.\left. +1\right)\right) \\
&= n_1 \cdot \exp\left(\sum_{i=2}^{t_2} \left(\frac{\log\gamma - \lambda}{(t_2 - i + 1)\log\gamma + c_s \log n + \lambda}\right.\right. \\
&\quad \left.\left. +\Theta\left(\log^{-2} n\right)\right)\right) \\
&= n_1 \cdot \exp\left(\int_0^{\alpha} \frac{\log\gamma - \lambda}{t\log\gamma + c_s} dt + \Theta\left(\log^{-1} n\right)\right) \\
&= n_1 \cdot \exp\left(\frac{\log\gamma - \lambda}{\log\gamma} \cdot \ln\frac{c_s + \alpha\log\gamma}{c_s} + \Theta\left(\log^{-1} n\right)\right) \\
&= \frac{n}{\log n} \cdot \frac{c}{c_s + \alpha\log\gamma} \exp\left(\frac{\log\gamma - \lambda}{\log\gamma} \cdot \ln\frac{c_s + \alpha\log\gamma}{c_s}\right) \\
&\quad +o\left(\frac{n}{\log n}\right).
\end{aligned}$$

We also know that

$$N = \sum_{j=1}^{t_2} n_j = \lambda^{-1}\left(cn - (\log\gamma + c_s \log n) \cdot n_{t_2}\right)$$

Thus,

$$N = \lambda^{-1}\left(cn - \left(\frac{c_s}{\alpha\log\gamma + c_s}\right)^{\frac{\lambda}{\log\gamma}} \cdot cn + o(n)\right). \tag{14}$$

If $t_1 b + N = n$, then the following equation holds,

$$\begin{aligned}
n = &\quad (2(c_q - c_s) + 1 - \alpha)\frac{cn}{c_q} \\
+ &\quad \frac{cn}{\lambda}\left(1 - \left(\frac{c_s}{\alpha\log\gamma + c_s}\right)^{\frac{\lambda}{\log\gamma}}\right).
\end{aligned}$$

Thus, we derive the following formula to compute the constant c, i.e.

$$\left(2\left(1 - \frac{c_s}{c_q} + \frac{1-\alpha}{2c_q}\right) + \frac{1}{\lambda}\left(1 - \left(\frac{c_s}{\alpha\log\gamma + c_s}\right)^{\frac{\lambda}{\log\gamma}}\right)\right)^{-1}.$$

Theorem 8.3. *Under Heuristics 1 and 2*[8]*, if* $0 \leq \alpha_0 \leq 2(c_q - c_s) + 1$ *and* $\gamma \neq 1$*, then a* (c_q, c_s) *LWE instance can be solved with time and memory complexity*

[8] The heuristic here is slightly reformulated, we assume that after step i of coded-BKW with sieving, the vectors of $\mathcal{L}_\Delta$ are uniformly distributed on a sphere with radius $\sqrt{N_i} \cdot B/(\gamma^{t_2 - i})$

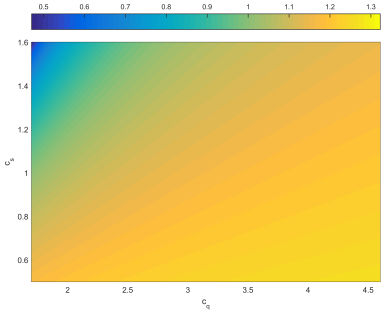


Figure 7: The optimal γ value for version D of the algorithm, as a function of (c_q, c_s).

$2^{(c+o(1))n}$, *where* c *is*

$$\left(\left(2\left(1-\frac{c_s}{c_q}\right)+\frac{1-\alpha_0}{c_q}\right)+\frac{1}{\lambda}\left(1-\left(\frac{c_s}{\alpha_0 \log \gamma + c_s}\right)^{\frac{\lambda}{\log \gamma}}\right)\right)^{-1}. \tag{15}$$

γ		0.78	0.80	0.82	0.84	0.86	0.88	0.90	0.92	0.94	0.96	0.98	1.00	1.02	1.04
λ	classic	0.610	0.577	0.544	0.512	0.482	0.452	0.423	0.395	0.368	0.342	0.317	0.292	0.269	0.246
	quantum	0.574	0.541	0.509	0.478	0.448	0.419	0.391	0.364	0.338	0.313	0.289	0.265	0.243	0.221
c	classic	0.8933	0.8930	0.8928	0.8927	**0.8927**	0.8928	0.8930	0.8932	0.8936	0.8940	0.8946	**0.8951**	0.8959	0.8967
	quantum	0.8796	**0.8795**	0.8795	0.8797	0.8800	0.8805	0.8810	0.8817	0.8825	0.8835	0.8845	**0.8856**	0.8870	0.8884

Table 3: The complexity Exponent c for various reduction factors when $(c_q, c_s) = (2, 1.5)$.

Example 3. The numerical results for the Regev parameters using various reduction factors are listed in Table 3, where the complexity exponent λ of the nearest neighbor searching is computed by the LSF approach [9]. Using S-BKW-v3, we can further decrease the complexity exponent c for solving the Regev LWE instance from 0.8951 to 0.8927 classically, and from 0.8856 to 0.8795 in a quantum setting. For these parameters, the choice of γ to achieve the best asymptotic complexity is 0.86 classically (or 0.80 using a quantum computer).

Optimal Choice of γ

The optimal choice of γ depends on the parameters c_q and c_s, illustrated in Fig. 7. The gap between c_q and c_s is important, since this optimal γ value increases with c_q for a fixed c_s, and decreases with c_s when c_q is determined.

8.4 An Asymptotic Comparison of the New Variants

We present a comparison describing the asymptotic behavior of the best single-exponential algorithms for solving the LWE problems with varying (c_q, c_s) in Fig. 8. The upper sub-plot includes all previous algorithms, which have been shown in Figure 4. S-BKW refers to coded-BKW with sieving with preprocessing, as defined in Section 6.2. We further add the new BKW variants from this section in the lower part.

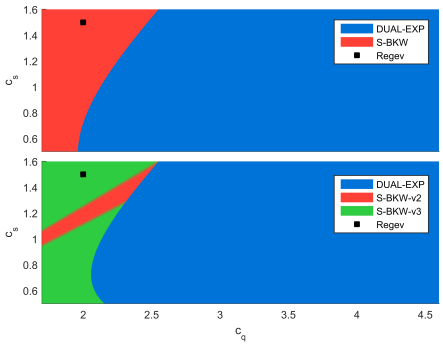


Figure 8: A comparison of the asymptotic behavior of the best single-exponential algorithms for solving the LWE problem for different values of c_q and c_s. The different areas show where in the parameter space the corresponding algorithm beats the other algorithms in that subplot.

Notice that all previous BKW variants are special cases of the three new algorithms, i.e., S-BKW-v1, S-BKW-v2, and S-BKW-v3, so in the lower sub-plot and for a particular pair of (c_q, c_s), the best algorithm is always among these three variants and DUAL-EXP. From this sub-plot, firstly, the area where DUAL-EXP wins becomes significantly smaller. Secondly, with respect to the area that the BKW variants win, S-BKW-v3 beats the other two in most pairs of parameters. S-BKW-v2 is the best algorithm in a thin strip, and by comparing with Fig. 7, we notice that this strip corresponds to an area where the optimal γ value is close to 1. Therefore, for the pairs of (c_q, c_s) in this area, optimizing for γ does not help that much, and S-BKW-v2 is superior. S-BKW-v1 never wins for parameters considered in this graph.

8.5 A High Level Description

A high level comparison showing how the different new versions of coded-BKW with sieving work, similar to Fig. 3, can be found in Fig. 6. In all versions, pre-processing with plain BKW steps is excluded from the description.

In S-BKW-v1, we first take coded-BKW steps. This means longer and longer steps, and gradually increasing noise. Then we switch to coded-BKW with sieving steps. Here the steps get shorter and shorter since we have to apply sieving to an increasing number of previous steps.

In S-BKW-v2, we begin with shorter and shorter coded-BKW with sieving steps, keeping the noise of the positions low. Then we finish off with longer and longer coded-BKW steps. Here we do not apply sieving to the previously sieved positions, thus the added noise of these positions grow.

For S-BKW-v3 we have two versions; S-BKW-v3a and S-BKW-v3b. These are identical except that they use different reduction factors γ. In S-BKW-v3a, we use regular coded-BKW with sieving steps with a reduction factor $\gamma > 1$. The steps get shorter and shorter because we need to sieve more and more positions. The added noise is small in the beginning, but in each sieved position it becomes larger and larger for each step. S-BKW-v3b is the same, except that we use $\gamma < 1$. This means that the noise is large in the beginning, but gets smaller and smaller for each step.

8.6 More Generalization

A straight-forward generalization of all the new variants described in Sections 8.1-8.3 is to allow different reduction parameter γ_i in different steps, after having pre-processed the samples with plain BKW steps. In addition, we can allow a sieving operation on positions in an interval $\mathcal{I}_i$ (or even more generally on any set of positions), using a flexible reduction factor γ_i. This optimization problem is complicated due to the numerous possible approaches, and generally, it is even difficult to write a closed formula for the objective function. We leave the problem of finding better asymptotic algorithms via extensive optimization efforts as an interesting scope for future research.

9 Conclusions and Future Work

In the paper we have presented a new BKW-type algorithm for solving the LWE problem. This algorithm, named coded-BKW with sieving, combines important ideas from two recent algorithmic improvements in lattice-based cryptography, i.e., coded-BKW and heuristic sieving for SVP, and outperforms the previously known approaches for important parameter sets in public-key cryptography.

For instance, considering Regev parameters, we have demonstrated an asymptotic improvement, reducing the time and space complexity from $2^{0.930n}$ to $2^{0.893n}$. Additionally, we showed a similar improvement, when restricting the number of available samples to be polynomial. Lastly, we obtained the first quantum acceleration for this parameter set, further reducing the complexity to $2^{0.880n}$ if quantum computers are provided.

In the conference version [24], this algorithm has proven significant non-asymptotic improvements for some concrete parameters, compared with the previously best BKW variants. But one should further investigate the analysis when heuristics like unnatural selection[9] are taken into consideration, in order to fully exploit its power on suggesting accurate security parameters for real cryptosystems. Moreover, the influence on the concrete complexity of using varying reduction factors is unclear. For this purpose, further analysis and extensive simulation results are needed, which can be a very interesting topic for future work.

Another stimulating problem is to search for new algorithms with better asymptotic complexity by solving the general optimization problem raised in Section 8.6, numerically or analytically.

Lastly, the newly proposed algorithm definitely also has importance in solving many LWE variants with specific structures, e.g., the Ring-LWE problem. An interesting research direction is to search for more applications, e.g., solving hard lattice problems, as in [31].

Acknowledgment

The authors would like to thank the anonymous reviewers from ASIACRYPT 2017 and the reviewers for IEEE Transactions on Information Theory for their

[9] Unnatural selection means creating more reduced vectors than needed and then choosing the best ones for the next step of the reduction. The idea is from [4].

invaluable comments that helped improve the quality of this paper.

References

[1] Ajtai, M., Kumar, R., Sivakumar, D.: A sieve algorithm for the shortest lattice vector problem. In: Proceedings of the thirty-third annual ACM symposium on Theory of computing. pp. 601–610. ACM (2001)

[2] Albrecht, M., Cid, C., Faugere, J.C., Robert, F., Perret, L.: Algebraic algorithms for LWE problems. Cryptology ePrint Archive, Report 2014/1018 (2014), http://eprint.iacr.org/2014/1018

[3] Albrecht, M.R., Cid, C., Faugere, J.C., Fitzpatrick, R., Perret, L.: On the complexity of the BKW algorithm on LWE. Designs, Codes and Cryptography 74(2), 325–354 (2015)

[4] Albrecht, M.R., Faugère, J.C., Fitzpatrick, R., Perret, L.: Lazy Modulus Switching for the BKW Algorithm on LWE. In: Krawczyk, H. (ed.) Public-Key Cryptography–PKC 2014, Lecture Notes in Computer Science, vol. 8383, pp. 429–445. Springer Berlin Heidelberg (2014)

[5] Albrecht, M.R., Fitzpatrick, R., Göpfert, F.: On the efficacy of solving LWE by reduction to unique-SVP. In: International Conference on Information Security and Cryptology. pp. 293–310. Springer (2013)

[6] Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology 9(3), 169–203 (2015)

[7] Applebaum, B., Cash, D., Peikert, C., Sahai, A.: Fast Cryptographic Primitives and Circular-Secure Encryption Based on Hard Learning Problems. In: Halevi, S. (ed.) Advances in Cryptology–CRYPTO 2009, Lecture Notes in Computer Science, vol. 5677, pp. 595–618. Springer Berlin Heidelberg (2009)

[8] Arora, S., Ge, R.: New algorithms for learning in presence of errors. In: Automata, Languages and Programming, pp. 403–415. Springer (2011)

[9] Becker, A., Ducas, L., Gama, N., Laarhoven, T.: New directions in nearest neighbor searching with applications to lattice sieving. In: Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 10–24. Society for Industrial and Applied Mathematics (2016)

[10] Becker, A., Gama, N., Joux, A.: A sieve algorithm based on overlattices. LMS Journal of Computation and Mathematics 17(A), 49–70 (2014)

[11] Bernstein, D.J., Lange, T.: Never trust a bunny. In: Radio Frequency Identification. Security and Privacy Issues, pp. 137–148. Springer (2013)

[12] Blum, A., Kalai, A., Wasserman, H.: Noise-tolerant Learning, the Parity Problem, and the Statistical Query Model. In: Proceedings of the Thirty-second Annual ACM Symposium on Theory of Computing–STOC 2000, pp. 435–440. ACM (2000)

[13] Blum, A., Kalai, A., Wasserman, H.: Noise-tolerant learning, the parity problem, and the statistical query model. J. ACM 50(4), 506–519 (2003)

[14] Bogos, S., Vaudenay, S.: Optimization of LPN solving algorithms. In: Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I 22. pp. 703–728. Springer (2016)

[15] Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. In: Advances in Cryptology–CRYPTO 2012, pp. 868–886. Springer (2012)

[16] Brakerski, Z., Vaikuntanathan, V.: Efficient Fully Homomorphic Encryption from (Standard) LWE. In: Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science. pp. 97–106. IEEE Computer Society (2011)

[17] Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-LWE and security for key dependent messages. In: Annual Cryptology Conference. pp. 505–524. Springer (2011)

[18] Chen, Y., Nguyen, P.Q.: BKZ 2.0: Better lattice security estimates. In: Advances in Cryptology–ASIACRYPT 2011, pp. 1–20. Springer (2011)

[19] Conway, J. H., Sloane, N. J. A.: Sphere packings, lattices and groups. In: (Vol. 290). Springer Science and Business Media (2013)

[20] Dubiner, M.: Bucketing coding and information theory for the statistical high-dimensional nearest-neighbor problem. IEEE Transactions on Information Theory 56(8), 4166–4179 (2010)

[21] Duc, A., Tramèr, F., Vaudenay, S.: Better Algorithms for LWE and LWR. In: Advances in Cryptology – EUROCRYPT 2015, pp. 173–202. Springer (2015)

[22] Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Advances in Cryptology–CRYPTO 2013, pp. 75–92. Springer (2013)

[23] Guo, Q., Johansson, T., Löndahl, C.: Solving LPN using covering codes. In: Advances in Cryptology–ASIACRYPT 2014, pp. 1–20. Springer (2014)

[24] Guo, Q., Johansson, T., Mårtensson, E., Stankovski, P.: Coded-BKW with Sieving. In: Advances in Cryptology–ASIACRYPT 2017, Part I, pp. 323–346. Springer (2017)

[25] Guo, Q., Johansson, T., Stankovski, P.: Coded-BKW: Solving LWE using lattice codes. In: Advances in Cryptology–CRYPTO 2015, pp. 23–42. Springer (2015)

[26] Hanrot, G., Pujol, X., Stehlé, D.: Algorithms for the shortest and closest lattice vector problems. In: Coding and Cryptology, pp. 159–190. Springer (2011)

[27] Herold, G., Kirshanova, E., May, A.: On the asymptotic complexity of solving LWE. IACR Cryptology ePrint Archive 2015, 1222 (2015), http://eprint.iacr.org/2015/1222

[28] Herold, G., Kirshanova, E., May, A.: On the asymptotic complexity of solving LWE. J. Designs, Codes and Cryptography, pp. 1–29 (2017)

[29] Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the thirtieth annual ACM symposium on Theory of computing. pp. 604–613. ACM (1998)

[30] Kirchner, P.: Improved generalized birthday attack. Cryptology ePrint Archive, Report 2011/377 (2011), http://eprint.iacr.org/2011/377

[31] Kirchner, P., Fouque, P.A.: An improved BKW algorithm for LWE with applications to cryptography and lattices. In: Advances in Cryptology–CRYPTO 2015, pp. 43–62. Springer (2015)

[32] Laarhoven, T.: Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In: Annual Cryptology Conference. pp. 3–22. Springer (2015)

[33] Laarhoven, T., Mosca, M., Van De Pol, J.: Finding shortest lattice vectors faster using quantum search. Designs, Codes and Cryptography 77(2-3), 375–400 (2015)

[34] Laarhoven, T., de Weger, B.: Faster sieving for shortest lattice vectors using spherical locality-sensitive hashing. In: International Conference on Cryptology and Information Security in Latin America. pp. 101–118. Springer (2015)

[35] Levieil, É., Fouque, P.A.: An improved LPN algorithm. In: Prisco, R.D., Yung, M. (eds.) SCN. Lecture Notes in Computer Science, vol. 4116, pp. 348–359. Springer-Verlag (2006)

[36] Lindner, R., Peikert, C.: Better Key Sizes (and Attacks) for LWE-Based Encryption. In: Kiayias, A. (ed.) Topics in Cryptology–CT-RSA 2011, Lecture Notes in Computer Science, vol. 6558, pp. 319–339. Springer Berlin Heidelberg (2011)

[37] Liu, M., Nguyen, P.Q.: Solving BDD by enumeration: An update. In: Topics in Cryptology–CT-RSA 2013, pp. 293–309. Springer (2013)

[38] May, A., Ozerov, I.: On computing nearest neighbors with applications to decoding of binary linear codes. In: Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I. pp. 203–228 (2015)

[39] Micciancio, D., Regev, O.: Worst-case to average-case reductions based on Gaussian measures. SIAM J. Comput., pp. 372-381 (2004)

[40] Micciancio, D., Regev, O.: Lattice-based Cryptography. In: Bernstein, D.J., Buchmann, J., Dahmen, E. (eds.) Post-Quantum Cryptography, pp. 147–191. Springer Berlin Heidelberg (2009)

[41] Micciancio, D., Voulgaris, P.: Faster exponential time algorithms for the shortest vector problem. In: Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms. pp. 1468–1480. SIAM (2010)

[42] Mulder, E.D., Hutter, M., Marson, M.E., Pearson, P.: Using Bleichenbacher's solution to the hidden number problem to attack nonce leaks in 384-bit ECDSA: extended version. J. Cryptographic Engineering 4(1), 33–45 (2014)

[43] Nguyen, P.Q., Vidick, T.: Sieve algorithms for the shortest vector problem are practical. J. Mathematical Cryptology 2(2), 181–207 (2008)

[44] Pujol, X., Stehlé, D.: Solving the shortest lattice vector problem in time $2^{2.465n}$. IACR Cryptology ePrint Archive 2009, 605 (2009), http://eprint.iacr.org/2009/605

[45] Regev, O.: On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. Journal of the ACM 56(6), 34:1–34:40 (Sep 2009)

[46] Schnorr, C.P., Euchner, M.: Lattice basis reduction: Improved practical algorithms and solving subset sum problems. Mathematical programming 66(1-3), 181–199 (1994)

[47] Wagner, D.: A generalized birthday problem. In: Advances in cryptology–CRYPTO 2002, pp. 288–304. Springer (2002)

[48] Wang, X., Liu, M., Tian, C., Bi, J.: Improved Nguyen-Vidick heuristic sieve algorithm for shortest vector problem. In: Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security. pp. 1–9. ACM (2011)

[49] Zhang, B., Jiao, L., Wang, M.: Faster algorithms for solving LPN. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 168–195. Springer (2016)

[50] Zhang, F., Pan, Y., Hu, G.: A three-level sieve algorithm for the shortest vector problem. In: International Conference on Selected Areas in Cryptography. pp. 29–47. Springer (2013)

Paper VI

The Asymptotic Complexity of Coded-BKW with Sieving Using Increasing Reduction Factors

The Learning with Errors problem (LWE) is one of the main candidates for post-quantum cryptography. At Asiacrypt 2017, coded-BKW with sieving, an algorithm combining the Blum-Kalai-Wasserman algorithm (BKW) with lattice sieving techniques, was proposed. In this paper, we improve that algorithm by using different reduction factors in different steps of the sieving part of the algorithm. In the Regev setting, where $q = n^2$ and $\sigma = n^{1.5}/(\sqrt{2\pi}\log_2^2 n)$, the asymptotic complexity is $2^{0.8917n}$, improving the previously best complexity of $2^{0.8927n}$. When a quantum computer is assumed or the number of samples is limited, we get a similar level of improvement.

Erik Mårtensson "The Asymptotic Complexity of Coded-BKW with Sieving Using Increasing Reduction Factors", in *2019 IEEE International Symposium on Information Theory (ISIT)*, pp. 2579-2583, 2019, Paris, France.

1 Introduction

Given access to large-scale quantum computers, Shor's algorithm solves both the integer factoring problem and the discrete logarithm problem in polynomial time. To remedy this, National Institute of Standards and Technology (NIST) has an ongoing competition to develop post-quantum cryptosystems [1]. One of the main underlying mathematical problems in the competition is the Learning with Errors problem (LWE).

The LWE problem was introduced by Regev in [2]. It has some really nice features, such as a reduction from average-case LWE instances to worst-case instances of hard lattice problems. An application of LWE is Fully Homomorphic Encryption (FHE) [3]. An important special case of LWE is the Learning Parity with Noise problem (LPN), essentially a binary version of LWE with Bernoulli distributed noise.

There are mainly three types of algorithms for solving the LWE problem. For surveys on the concrete and asymptotic complexity of these algorithms see [4] and [5] respectively.

The first type is the Arora-Ge algorithm, which was introduced in [6], and then improved in [7]. This type of algorithm is mostly applicable when the noise is too small for Regev's reduction proof to apply [2].

The second type of approach is lattice-based algorithms, where LWE is transformed into a lattice problem and then solved by methods like lattice-reduction, lattice sieving and enumeration. Lattice-based algorithms are currently the fastest algorithms in practice, and have the advantage of not needing an exponential amount of samples. For more details see [4] and the references therein.

The third type of approach is the Blum-Kalai-Wasserman (BKW) set of algorithms. These will be the focus of this paper.

The BKW algorithm was introduced in [8] as the first sub-exponential algorithm for solving the LPN problem. It was first used to solve the LWE problem in [9]. This was improved in [10] using Lazy Modulus Switching (LMS). Further improvements were made in [11, 12] by using a varying step size and a varying degree of reduction. In [13] coded-BKW with sieving was introduced, where lattice sieving techniques were used to improve the BKW algorithm. The full version in [14] improved the coded-BKW with sieving algorithm by finding the optimal reduction factor used for lattice sieving.

In this paper we further improve upon the coded-BKW with sieving algorithm by increasing the reduction factor for each step of the algorithm. We achieve a record low time complexity of $2^{0.8917n}$ in the Regev setting; that is, when $q = n^2$ and $\sigma = n^{1.5}/(\sqrt{2\pi}\log_2^2 n)$. The previous best result was $2^{0.8927n}$ from [14]. Also if a quantum computer is assumed or the number of samples is limited we get a similar level of improvement.

The remaining parts of the paper are organized the following way. We start off in Section 2 by introducing the LWE problem. In Section 3 we go over the previous versions of the BKW algorithm, when used for solving the LWE problem. In Section 4 we introduce the new algorithm and in Section 5 we cover the asymptotic complexity of it and other algorithms for solving LWE. We show our results in Section 6 and conclude the paper in Section 7.

2 Preliminaries

Let us define the LWE problem.

Definition 2.1 (LWE)**.** *Let n be a positive integer, q a prime. Let $\mathbf{s}$ be a uniformly random secret vector in $\mathbb{Z}_q^n$. Assume access to m noisy scalar products between $\mathbf{s}$ and known vectors $\mathbf{a_i}$, i.e.*

$$z_i = \langle \mathbf{a_i}, \mathbf{s} \rangle + e_i,$$

for $i = 1, \ldots, m$. The small error terms e_i are discrete Gaussian distributed with mean 0 and standard deviation σ. The (search) LWE problem is to find the secret vector $\mathbf{s}$.

In other words, when solving LWE you have access to a large set of pairs $(\mathbf{a_i}, z_i)$ and want to find the corresponding secret vector $\mathbf{s}$. In some versions there are restrictions on the number of samples you have access to.

3 BKW

BKW was introduced as the first sub-exponential algorithm for solving LPN (essentially LWE with $q = 2$) in [8]. It was first used for solving LWE in [9].

3.1 Plain BKW

The BKW algorithm consists of two steps, dimension reduction and guessing.

Reduction

Map all the samples into categories, such that the first b positions get canceled when adding/subtracting a pair of $\mathbf{a}$ vectors within the same category.

Given two samples $([\pm\mathbf{a_0}, \mathbf{a_1}], z_1)$ and $([\pm\mathbf{a_0}, \mathbf{a_2}], z_2)$ within the same category. By adding/subtracting the $\mathbf{a}$ vectors we get

$$\mathbf{a_{1,2}} = [\underbrace{0 \quad 0 \quad \cdots \quad 0}_{b \text{ symbols}} \quad * \quad * \quad \cdots \quad *].$$

By also calculating the corresponding z value we get $z_{1,2} = z_1 \pm z_2$. Now we have a new sample $(\mathbf{a_{1,2}}, z_{1,2})$. The corresponding noise variable is $e_{1,2} = e_1 \pm e_2$. Thus the variance of the new noise is $2\sigma^2$, where σ^2 is the variance of the originial noise. By going through all categories and calculating a suitable amount of new samples we have reduced the dimensionality of the problem by b, at the cost of increasing the noise. If we repeat the reduction process t_0 times we end up with a dimensionality of $n - t_0 b$, and a noise variance of $2^{t_0} \cdot \sigma^2$.

Guessing

The final positions of the secret vector $\mathbf{s}$ can be guessed and then each guess can be tested using a distinguisher. The guessing procedure does not affect the asymptotics, but is important for concrete complexity. The guessing procedure was improved in [15] using the Fast Fourier Transform (FFT).

3.2 Lazy Modulus Switching

The basic BKW algorithm was improved in [10] by Albrecht et al. The main idea there was to map samples that, almost but not completely, canceled each other, into the same category. This technique is called Lazy Modulus Switching (LMS).

By doing this an extra error term gets added in each step. The variance of this noise also doubles in each new reduction step. However, LMS allows us to use a larger step size, allowing us to solve larger LWE problems.

One problem with this version of the algorithm is that the extra added noise of the earlier steps grows in size much more than the noise of the later steps, leading to an uneven noise distribution among the positions of the final samples used for the guessing procedure.

3.3 Coded-BKW

The problem with the uneven noise distribution was adressed independently in [11,12]. The idea was to use a small step size and almost reduce the positions in the $\mathbf{a}$ vectors to 0 in the first step, and then gradually increase the step size n_i and use less strict reduction for each step.

In [11] different q-ary linear codes $\mathcal{C}_i$ with parameters $[n_i, b]$ were used, to vary the strictness of reduction. That version of BKW is called coded-BKW. For simplicity, consider the first reduction step. Pick two samples, such that the first n_1 positions of the $\mathbf{a}$ vectors map to the same codeword $\mathbf{c_0}$ in $\mathcal{C}_1$. In other words, we can write

$$\begin{aligned} z_1 &= \langle [\mathbf{c}_0 + \hat{\mathbf{e}}_1, \mathbf{a}_1],\ \mathbf{s} \rangle + e_1 \\ z_2 &= \langle [\mathbf{c}_0 + \hat{\mathbf{e}}_2, \mathbf{a}_2],\ \mathbf{s} \rangle + e_2, \end{aligned}$$

where $\hat{\mathbf{e}}_1$ and $\hat{\mathbf{e}}_2$ have small Euclidean norms. We can get a new sample by calculating

$$z_1 - z_2 = \langle [\hat{\mathbf{e}}_1 - \hat{\mathbf{e}}_2, \mathbf{a}_1 - \mathbf{a}_2],\ \mathbf{s} \rangle + e_1 - e_2.$$

Just like when using LMS, using this version of BKW adds an extra noise term, but allows us to use larger step sizes.

3.4 Coded-BKW with Sieving

In [13] an idea for combining BKW with lattice sieving techniques was introduced. Just like in [11,12] in step i, samples were mapped into categories based on the current n_i positions. Let $N_i = \sum_{j=1}^{i} n_j$. The new idea was to only add/subtract samples within a category such that also the previous N_{i-1} positions of the resulting $\mathbf{a}$ vector were equally small. This could have been done by looking at all possible pairs and picking only the ones with the smallest values in these positions. However, a more efficient way of doing this was to use lattice sieving techniques to find close pairs of vectors within a category faster.

A micro picture of coded-BKW with sieving can be found in Figure1. After step i, the average magnitude of the first N_i positions in the $\mathbf{a}$ vector is less than a constant B.

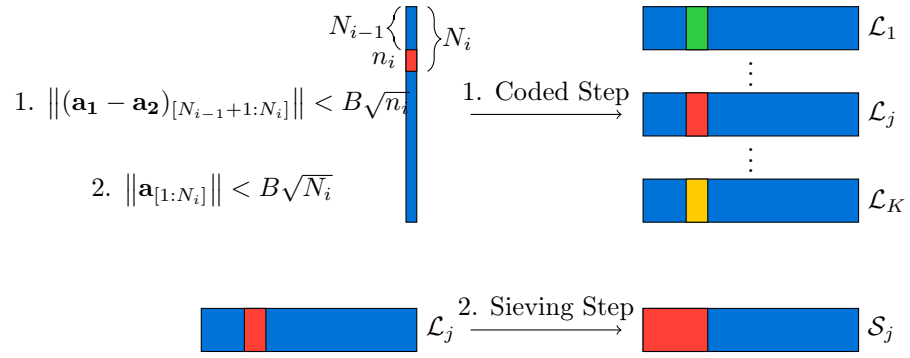


Figure 1: A micro picture of how one step of coded-BKW with sieving works. Slightly changed version of Figure 1 from [14]. Each sample gets mapped to one list $\mathcal{L}_j$ out of K lists. Sieving is then applied to each list to form new lists $\mathcal{S}_j$.

Using Different Reduction Factors

In [14] the idea of finding an optimal reduction factor was introduced. Instead of making sure the N_i positions currently considered are as small as the N_{i-1} positions in the previous step, they are made to be γ times as large. Depending on the parameter setting the optimal strategy is either to use $\gamma < 1$ or $\gamma > 1$, or in other words to gradually decrease or increase the values in the $\mathbf{a}$ vector. The final average magnitude is still less than the same constant B.

The original coded-BKW with sieving algorithm from [13] is the special case where $\gamma = 1$ and coded-BKW is the special case where $\gamma = \sqrt{2}$.

For an illustration of how the different BKW algorithms reduce the $\mathbf{a}$ vector, see Figure 2.

4 Coded-BKW with Sieving with Increasing Reduction Factors

The new idea in this paper is to use different reduction factors γ_i in different steps i. The idea is that in the earlier steps the sieving is cheap, and we can therefore use small values of γ_i. Gradually the sieving procedure gets more and more expensive, forcing us to increase the value of γ_i.

Assume that we take t_2 steps of coded-BKW with sieving in total and let $\gamma_1 = \gamma_s$ and$\gamma_{t_2} = \gamma_f$. We ended up choosing an arithmetic progression, that is we let

$$\gamma_i = \gamma_s + \frac{\gamma_f - \gamma_s}{t_2 - 1}(i - 1).$$

We also tried a geometric and logarithmic progression of the γ_i values, both leading to a slightly worse complexity. A power progression lead to an expression that had to be estimated numerically and resulted in almost exactly the same results as the arithmetic progression.

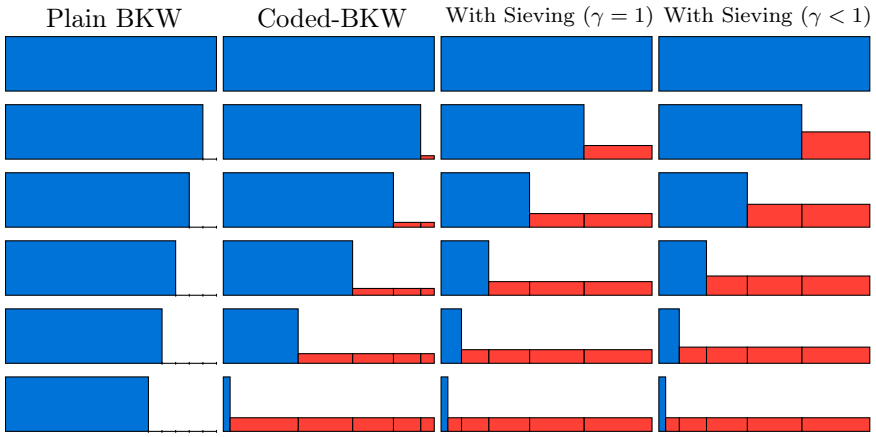


Figure 2: A high-level illustration of how the different versions of the BKW algorithm work. The x-axis represents positions in the $\mathbf{a}$ vector, and the y-axis depicts the average absolute value of the corresponding position. The blue color corresponds to positions that have not been reduced yet and the red color corresponds to reduced positions. The last few positions are used for guessing. The figure is a modified version of Figures 3 and 8 from [14].

5 Asymptotic Complexity

Asymptotically we let $q = n^{c_q}$ and $\sigma = n^{c_s}$, where c_q and c_s are constants. For most algorithms and settings the asymptotic complexity of solving LWE is $2^{cn+o(n)}$, where the exponent c depends on c_q and c_s. We leave settings such as a binary secret or a superpolynomial q for future research.

We will now quickly cover the asymptotic complexities of the Arora-Ge algorithm, lattice-based algorithms and all the previous versions of BKW, as a function of c_q and c_s. Initially, the assumed setting is one with a classical computer and an exponential amount of samples. Other settings will be discussed later.

Complexity Exponent for Lattice Sieving

The value $\lambda(\gamma)$ for lattice sieving using a reduction factor γ is the best available complexity exponent for doing lattice sieving. It is (currently) calculated by doing the optimization from the section about the total cost of sieving in [16], replacing the angle $\pi/3$ by $\theta = 2\arcsin(\gamma/2)$ and replacing $N = (4/3)^{n/2}$ by $(1/\sin(\theta))^n$. For $\gamma = 1$ we get $\lambda \approx 0.292$.

Quantum Setting

If having access to a quantum computer, Grover's algorithm can be used to speed up the lattice sieving, see [17], resulting in slightly improved complexity exponents. For $\gamma = 1$ we get $\lambda \approx 0.265$.

5.1 Arora-Ge and Lattice-based Methods

The Arora-Ge algorithm is polynomial when $c_s < 0.5$ and superexponential when $c_s > 0.5$, making it viable if and only if c_s is too small for Regev's reduction proof to apply [2]. Lattice-based algorithms can solve LWE with a time complexity exponent of $2\lambda c_q/(c_q - c_s + 1/2)^2$, using an exponential amount of memory [5].

5.2 Plain and Coded BKW

The time and space complexity for solving LWE using plain BKW is $c_q/(2(c_q - c_s) + 1)$ [9], and using Coded-BKW is $(1/c_q + 2\ln(c_q/c_s))^{-1}$ [12].

5.3 Coded-BKW with sieving

The time (and space) complexity of solving the LWE problem using coded-BKW with sieving gets calculated by solving increasingly difficult optimization problems. In both Theorem 5.1 and 5.2, the parameter α decides how large part of the samples should be pre-processed with plain BKW steps.

Theorem 5.1. *The time and space complexity of coded-BKW with sieving and a constant value of the reduction factor γ, is $2^{cn+o(n)}$, where c is the solution to the following optimization problem.*

$$\begin{aligned} \underset{\alpha,\gamma}{\textbf{\textit{minimize}}} \quad & c(\alpha,\gamma) = \left(\frac{2(c_q - c_s) + 1 - \alpha}{c_q} + \right. \\ & \left. \frac{1}{\lambda(\gamma)} \left(1 - \frac{c_s}{\alpha \log_2 \gamma + c_s} \cdot \exp(I(\alpha,\gamma)) \right) \right)^{-1} \\ \textbf{\textit{subject to}} \quad & 0 \leq \alpha \leq 2(c_q - c_s) + 1, \\ & 0 < \gamma \leq \sqrt{2}. \end{aligned}$$

Here, we have

$$I(\alpha,\gamma) = \int_0^\alpha \frac{\log_2 \gamma - \lambda(\gamma)}{t \log_2 \gamma + c_s} dt.$$

By setting $\gamma = 1$ we get (a restatement of) the complexity of the original coded-BKW with sieving algorithm [13].

Proof 5.1. *The theorem is a slight restatement of Theorem 7 from [14], to make it more similar to Theorem 5.2 of this paper. Theorem 7 from [14] includes both the proof and the underlying heuristic assumptions the proof is based on.*

5.4 Coded-BKW with Sieving with Increasing Reduction Factors

The complexity of the new algorithm is covered in the following theorem.

Theorem 5.2. *The time and space complexity of coded-BKW with sieving and an arithmetic progression of the γ_i values, is $2^{cn+o(n)}$, where c is the solution to the following optimization problem.*

$$\begin{aligned}
\underset{\alpha,\gamma_s,\gamma_f}{\textbf{\textit{minimize}}} \quad & c(\alpha,\gamma_s,\gamma_f) = \left(\frac{2(c_q-c_s)+1-\alpha}{c_q} + \right. \\
& \left. \int_0^{\alpha} \frac{c_s}{\left(t\cdot\ell(t)+c_s\right)^2}\cdot\exp(I(t;\alpha,\gamma_s,\gamma_f))dt\right)^{-1} \\
\textbf{\textit{subject to}} \quad & 0 \le \alpha \le 2(c_q-c_s)+1, \\
& 0 < \gamma_s < \gamma_f \le \sqrt{2}.
\end{aligned}$$

Here, we have

$$I(t;\alpha,\gamma_s,\gamma_f) = \int_0^t \frac{\log_2\gamma(s)-\lambda(\gamma(s))}{s\cdot\ell(s)+c_s}ds,$$

and

$$\begin{aligned}
\gamma(s) &= \gamma_s + \frac{\alpha-s}{\alpha}(\gamma_f-\gamma_s), \\
\ell(s) &= \left(\frac{\gamma_f\ln(\gamma_f)-\gamma(t)\ln(\gamma(t))}{\gamma_f-\gamma_s}\frac{\alpha}{s}-1\right)/\ln(2).
\end{aligned}$$

It should be mentioned that the objetive function of the optimization problem here would change slightly if another method for the progression of the γ_i values was chosen.

Proof 5.2. *A proof of Theorem 5.2 can be found in the appendix.*

5.5 Polynomial Number of Samples

With access to only a polynomial number of samples the complexity exponent of lattice-based algorithms changes to $2\lambda c_q/(c_q-c_s)^2$ [5].

When using BKW with access to only a polynomial number of samples, amplification is used to increase the number of samples, at the cost of an increased noise. For plain and coded BKW the complexity exponents change to $c_q/(2(c_q-c_s))$ and $(1/c_q+2\ln(c_q/c_s))^{-1}$ [5].

In the optimization problems in Theorem 5.1 and 5.2 the upper limit of α changes to $2(c_q-c_s)$. For each theorem the numerator in the first term of the objective function changes to $2(c_q-c_s)-\alpha$.

6 Results

Let us use the Regev instances as a case study, in other words, let $c_q = 2$ and $c_s = 1.5$ [2]. Table 1 shows the complexity exponent for coded-BKW with sieving, with $\gamma = 1$, an optimized constant γ and an arithmetic progression of the γ values, for four different scenarios. Either we use classical or quantum computers and either we have access to a polynomial or an exponential number of samples. Notice how using increasing reduction factors improves the complexity exponent in all the scenarios.

In all the scenarios in the Regev setting BKW algorithms beat lattice-based algorithms. For a picture comparing the asymptotic complexity of the different

BKW versions with lattice-based algorithms in other settings, see Figure 5 and 7 in [14]. The new version constitutes an improvement compared to the constant γ algorithm for all parameter pairs (c_q, c_s), as can be seen in Figure 3.

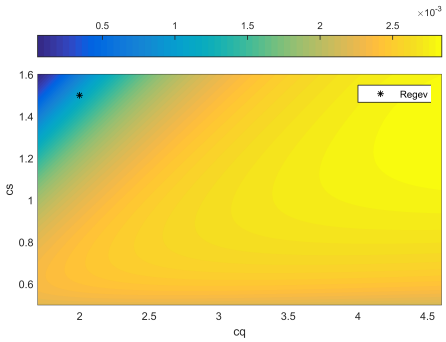


Figure 3: The improvement in the complexity exponent when going from a constant γ to an arithmetic progression of the γ values

The source code used for calculating all the complexity exponents in the different scenarios can be found on GitHub[1].

	Classical Setting	Quantum Setting
Exponential samples	0.8951 ($\gamma = 1$) 0.8927 (γ constant) **0.8917** (γ arithmetic)	0.8856 ($\gamma = 1$) 0.8795 (γ constant) **0.8782** (γ arithmetic)
Polynomial samples	1.6507 ($\gamma = 1$) 1.6417 (γ constant) **1.6399** (γ arithmetic)	1.6364 ($\gamma = 1$) 1.6211 (γ constant) **1.6168** (γ arithmetic)

Table 1: The asymptotic complexity exponent for the different versions of BKW in the Regev setting.

7 Conclusions

We have developed a new version of coded-BKW with sieving, achieving a further improved asymptotic complexity. We have also improved the complexity when having access to a quantum computer or only having access to a polynomial number of samples. All BKW algorithms solve a modified version of the 2-list problem, where we also have a modular reduction and have a limited number of total steps. Generalizing the optimization of the BKW algorithm from this perspective is an interesting new research idea. Another possible

[1] https://github.com/ErikMaartensson/BKWIncreasingReductionFactors

further research direction is looking at the complexity of different versions of BKW when only having access to a limited amount of memory, like was briefly started in [18]. Finally, it is interesting to investigate the concrete complexity of coded-BKW with sieving and implement it to see when BKW starts to beat lattice-type algorithms in practise.

Acknowledgment

The author would like to acknowledge Qian Guo, whose idea of using a reduction factor $\gamma \neq 1$ this paper generalizes. The author would also like to thank Thomas Johansson and Paul Stankovski Wagner for fruitful discussions during the writing of this paper.

References

[1] National Institute of Standards and Technology, "The post-quantum cryptography standardization process," 2019, https://csrc.nist.gov/projects/post-quantum-cryptography.

[2] O. Regev, "On Lattices, Learning with Errors, Random Linear Codes, and Cryptography," in *STOC*. ACM Press, 2005, pp. 84–93.

[3] C. Gentry, "Fully Homomorphic Encryption Using Ideal Lattices," in *STOC*. ACM Press, 2009, pp. 169–178.

[4] M. R. Albrecht, R. Player, and S. Scott, "On The Concrete Hardness Of Learning With Errors," *J. Mathematical Cryptology*, vol. 9, no. 3, pp. 169–203, 2015.

[5] G. Herold, E. Kirshanova, and A. May, "On the asymptotic complexity of solving LWE," *Designs, Codes and Cryptography*, vol. 86, no. 1, pp. 55–83, 2018.

[6] S. Arora and R. Ge, "New Algorithms for Learning in Presence of Errors," in *Automata, Languages and Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 403–415.

[7] M. R. Albrecht, C. Cid, J.-C. Faugère, and L. Perret, "Algebraic algorithms for LWE," Cryptology ePrint Archive, Report 2014/1018, 2014, http://eprint.iacr.org/2014/1018.

[8] A. Blum, A. Kalai, and H. Wasserman, "Noise-Tolerant Learning, the Parity Problem, and the Statistical Query Model," in *STOC*. ACM Press, 2000, pp. 435–440.

[9] M. R. Albrecht, C. Cid, J.-C. Faugère, R. Fitzpatrick, and L. Perret, "On the complexity of the BKW algorithm on LWE," *Designs, Codes and Cryptography*, vol. 74, no. 2, pp. 325–354, 2015.

[10] M. R. Albrecht, J.-C. Faugère, R. Fitzpatrick, and L. Perret, "Lazy Modulus Switching for the BKW Algorithm on LWE," in *PKC*, ser. LNCS, vol. 8383. Springer, Heidelberg, Germany, 2014, pp. 429–445.

[11] Q. Guo, T. Johansson, and P. Stankovski, "Coded-BKW: Solving LWE Using Lattice Codes," in *CRYPTO*, ser. LNCS, vol. 9215. Springer, Heidelberg, Germany, 2015, pp. 23–42.

[12] P. Kirchner and P.-A. Fouque, "An Improved BKW Algorithm for LWE with Applications to Cryptography and Lattices," in *CRYPTO*, ser. LNCS, vol. 9215. Springer, Heidelberg, Germany, 2015, pp. 43–62.

[13] Q. Guo, T. Johansson, E. Mårtensson, and P. Stankovski, "Coded-BKW with Sieving," in *ASIACRYPT*, ser. LNCS, vol. 10624. Springer, Heidelberg, Germany, 2017, pp. 323–346.

[14] Q. Guo, T. Johansson, E. Mårtensson, and P. Stankovski Wagner, "On the Asymptotics of Solving the LWE Problem Using Coded-BKW with Sieving," *IEEE Transactions on Information Theory*, 2019.

[15] A. Duc, F. Tramèr, and S. Vaudenay, "Better Algorithms for LWE and LWR," in *EUROCRYPT*, ser. LNCS, vol. 9056. Springer, Heidelberg, Germany, 2015, pp. 173–202.

[16] A. Becker, L. Ducas, N. Gama, and T. Laarhoven, "New Directions in Nearest Neighbor Searching with Applications to Lattice Sieving," in *SODA*. ACM-SIAM, 2016, pp. 10–24.

[17] T. Laarhoven, M. Mosca, and J. van de Pol, "Finding shortest lattice vectors faster using quantum search," *Designs, Codes and Cryptography*, vol. 77, no. 2, pp. 375–400, 2015.

[18] A. Esser, F. Heuer, R. Kübler, A. May, and C. Sohler, "Dissection-BKW," in *CRYPTO*, ser. LNCS, vol. 10992. Springer, Heidelberg, Germany, 2018, pp. 638–666.

A Proof of Theorem 5.2

Let us prove Theorem 5.2.

Proof A.1. *This proof is generalization of the proof of Theorem 5.1, as proven in [14]. The proof structure is similar, but the proof is also much longer. To avoid making it longer than necessary, some notation is borrowed from the proof in [14].*

Instead of using a constant reduction factor γ, we use a low reduction factor in the first steps when the sieving is cheap, and then gradually increase the reduction factor. Let γ_i denote the reduction factor in step i. We have the constraints

$$0 < \gamma_1 < \gamma_2 < \ldots < \gamma_{t_2} \leq \sqrt{2}.$$

Let $\lambda_i = \lambda(\gamma_i)$ denote the corresponding complexity exponent for nearest neighbor searching using the LSF algorithm. By $\log$ we denote $\log_2$.

Later we will let γ_i increase arithmetically. Other progressions also work, and as long as possible we will use a general progression of the γ_i values.

We start by performing t_1 plain BKW steps and then t_2 coded-BKW with sieving steps with parameters (λ_i, γ_i) in the i^{th} of the latter steps. Let $t_2 = \alpha \log n + \mathcal{O}(1)$ and $t_1 = \beta \log n = (2(c_q - c_s) + 1 - \alpha) \log n + \mathcal{O}(1)$. We have the constraint that $0 \leq \alpha \leq 2(c_q - c_s) + 1$. Like previously the step size of the plain BKW steps is

$$b = \frac{cn}{c_q \log n}.$$

Let n_1 be the length of the first coded-BKW with sieving step. Let B_i denote the magnitude of the values at step i and B denote the final magnitude of the values. Then we get

$$B_1 = \frac{B}{\Pi_{i=1}^{t_2} \gamma_i}.$$

Therefore,

$$\left(\log(\Pi_{i=1}^{t_2} \gamma_i) + c_s \log(n)\right) \cdot n_1 = cn - \lambda_1 \ \cdot n_1.$$

Thus,

$$\begin{aligned} n_1 &= \frac{cn}{c_s \log n + \log(\Pi_{i=1}^{t_2} \gamma_i) + \lambda_1} \\ &= \frac{cn}{c_s \log n + \log(\Pi_{i=1}^{t_2} \gamma_i)} \cdot (1 + \Theta(\log^{-1} n)). \end{aligned} \tag{1}$$

Analogously with step 1, for step i we get

$$\left(\log(\Pi_{j=i}^{t_2} \gamma_j) + c_s \log(n)\right) \cdot n_i = cn - \lambda_i \sum_{j=1}^{i} n_j. \tag{2}$$

Use $\star_i$ to denote the expression within the outer parantheses in the left hand-side of (2), for step i. Multiply (2) for step i by λ_{i-1}, for step $i-1$ by λ_i and subtract the expressions to get

$$\lambda_{i-1} \star_i n_i - \lambda_i \star_{i-1} n_{i-1} = cn(\lambda_{i-1} - \lambda_i) - \lambda_{i-1}\lambda_i n_i. \tag{3}$$

Solving (3) *for* n_i *gives*

$$n_i = \frac{\lambda_i \star_{i-1} n_{i-1} + cn(\lambda_{i-1} - \lambda_i)}{\lambda_{i-1} \star_i + \lambda_{i-1}\lambda_i}. \tag{4}$$

Calculating n_{t_2} *from* (4) *gives*

$$n_{t_2} = n_1 \prod_{i=2}^{t_2} \frac{\lambda_i \star_{i-1}}{\lambda_{i-1}(\star_i + \lambda_i)} \tag{5}$$

$$+ \quad cn \sum_{i=2}^{t_2} \frac{\lambda_{i-1} - \lambda_i}{\lambda_{i-1}(\star_i + \lambda_i)} \prod_{j=i+1}^{t_2} \left(\frac{\lambda_j \star_{j-1}}{\lambda_{j-1}(\star_j + \lambda_j)} \right). \tag{6}$$

Let us next look at the term (6). *First we rewrite the product as*

$$\begin{aligned} \prod_{j=i+1}^{t_2} \left(\frac{\lambda_j \star_{j-1}}{\lambda_{j-1}(\star_j + \lambda_j)} \right) &= \prod_{j=i+1}^{t_2} \frac{\lambda_j}{\lambda_{j-1}} \left(\frac{\star_{j-1}}{\star_j + \lambda_j} \right) \\ &= \frac{\lambda_{t_2}}{\lambda_i} \prod_{j=i+1}^{t_2} \left(\frac{\star_j + \log(\gamma_{j-1}) + \lambda_j - \lambda_j}{\star_j + \lambda_j} \right) \\ &= \frac{\lambda_{t_2}}{\lambda_i} \prod_{j=i+1}^{t_2} \left(1 + \frac{\log(\gamma_{j-1}) - \lambda_j}{\star_j + \lambda_j} \right). \end{aligned}$$

Use Π_i *to denote the product and* a_i *to denote* $\star_i + \lambda_i$. *We can then write the term* (6) *as*

$$\begin{aligned} & cn \sum_{i=2}^{t_2} \frac{\lambda_{i-1} - \lambda_i}{\lambda_{i-1} a_i} \frac{\lambda_{t_2}}{\lambda_i} \Pi_i \\ &= cn \sum_{i=2}^{t_2} \frac{1}{a_i} \frac{\lambda_{t_2}}{\lambda_i} \Pi_i - cn \sum_{i=2}^{t_2} \frac{1}{a_i} \frac{\lambda_{t_2}}{\lambda_{i-1}} \Pi_i \\ &= cn \left(\frac{1}{a_{t_2}} - \frac{1}{a_2} \frac{\lambda_{t_2}}{\lambda_1} \Pi_2 + \sum_{i=2}^{t_2-1} \frac{\lambda_{t_2}}{\lambda_i} \left(\frac{\Pi_i}{a_i} - \frac{\Pi_{i+1}}{a_{i+1}} \right) \right). \end{aligned} \tag{7}$$

Next, we write the expression within paranthesis in the sum in (7) *as*

$$\begin{aligned}
&\frac{\Pi_i a_{i+1} - \Pi_{i+1} a_i}{a_i a_{i+1}} \\
&= \frac{\Pi_{i+1}\left(\left(1 + \frac{\log(\gamma_i) - \lambda_{i+1}}{\log(\Pi_{j=i+1}^{t_2}\gamma_j) + c_s \log(n) + \lambda_{i+1}}\right) a_{i+1} - a_i\right)}{a_i a_{i+1}} \\
&= \frac{\Pi_{i+1}\left(a_{i+1} + \log(\gamma_i) - \lambda_{i+1} - a_i\right)}{a_i a_{i+1}} \\
&= \frac{\Pi_{i+1}\left(\lambda_{i+1} + \log(\gamma_i) - \lambda_{i+1} - (\log(\gamma_i) + \lambda_i)\right)}{a_i a_{i+1}} \\
&= -\frac{\lambda_i}{a_i a_{i+1}} \Pi_{i+1}.
\end{aligned}$$

The product (5) *can be written as*

$$\frac{cn}{a_1} \frac{\lambda_{t_2}}{\lambda_1} \Pi_1.$$

Thus, adding the two parts (5) *and* (6) *gives*

$$cn \left(\frac{1}{a_{t_2}} - \lambda_{t_2} \sum_{i=1}^{t_2 - 1} \frac{\Pi_{i+1}}{a_i a_{i+1}} \right). \tag{8}$$

Next, we want to evaluate Π_{i+1}*, which can be rewritten as*

$$\begin{aligned}
&\prod_{j=i+2}^{t_2} 1 + \frac{\log(\gamma_{j-1}) - \lambda_j}{\log(\Pi_{k=j}^{t_2}\gamma_k) + c_s \log(n) + \lambda_j} \\
&= \exp\left(\ln\left(\prod_{j=i+2}^{t_2} 1 + \frac{\log(\gamma_{j-1}) - \lambda_j}{\log(\Pi_{k=j}^{t_2}\gamma_k) + c_s \log(n) + \lambda_j} \right) \right) \\
&= \exp\left(\sum_{j=i+2}^{t_2} \frac{\log(\gamma_{j-1}) - \lambda_j}{\log(\Pi_{k=j}^{t_2}\gamma_k) + c_s \log(n) + \lambda_j} + \Theta\left(\log^{-2} n\right) \right).
\end{aligned}$$

Now the progression of the γ_i *values needs to be specified. Use an arithmetic progression from* $\gamma_1 = \gamma_s$ *up to* $\gamma_{t_2} = \gamma_f$*, where* $0 < \gamma_s < \gamma_f \leq \sqrt{2}$*. That is, let*

$$\gamma_i = \gamma_s + d(i-1) = \gamma_s + \frac{\gamma_f - \gamma_s}{t_2 - 1}(i-1).$$

The idea now is to let n *go towards infinity and let the sum in* (8) *approach an integral. If we let* n *go towards infinity and make a change of variables we get*

$$\left[\begin{array}{l} t = (t_2 - j + 1)/\log(n) \\ j = t_2 - t\log(n) + 1 \\ dt = -\dfrac{1}{\log(n)} dj \\ j = 1 \Rightarrow t = \dfrac{t_2}{\log(n)} = \dfrac{\alpha \log(n)}{\log(n)} = \alpha \\ j = t_2 \Rightarrow t = 1/\log(n) \to 0, \ \textit{as } n \to \infty \\ \gamma_j = \gamma_{t_2 - t\log(n)+1} = \gamma_s + \dfrac{\gamma_f - \gamma_s}{t_2 - 1}(t_2 - t\log(n)) \\ \quad \to \gamma_s + \dfrac{\alpha - t}{\alpha}(\gamma_f - \gamma_s), \ \textit{as } n \to \infty \\ \lambda_j = \lambda(\gamma_j) \to \lambda\left(\gamma_s \dfrac{\alpha - t}{\alpha}(\gamma_f - \gamma_s)\right), \ \textit{as } n \to \infty \end{array}\right].$$

Let us denote $\gamma(t) = \gamma_s + \frac{\alpha - t}{\alpha}(\gamma_f - \gamma_s)$. *We also want to evaluate* $\log(\prod_{k=j}^{t_2}(\gamma_k))$. *First of all we have*

$$\begin{aligned} \prod_{k=j}^{t_2} \gamma_k &= d \cdot \frac{\gamma_j}{d} \cdot d \cdot \left(\frac{\gamma_j}{d} + 1\right) \cdots d \cdot \left(\frac{\gamma_j}{d} + t_2 - j\right) \\ &= d^{t_2 - j + 1} \frac{\Gamma\left(\frac{\gamma_j}{d} + t_2 - j + 1\right)}{\Gamma\left(\frac{\gamma_j}{d}\right)}. \end{aligned} \tag{9}$$

Let us denote $t' = t\log(n)$. *Since* $\gamma_j/d = (\gamma_s + (j-1)d)/d = \gamma_s/d + j - 1$ *we can rewrite* (9) *as*

$$d^{t'} \frac{\Gamma\left(\frac{\gamma_s}{d} + t_2\right)}{\Gamma\left(\frac{\gamma_s}{d} + t_2 - t'\right)} = d^{t'} \frac{\Gamma\left(\frac{\gamma_f}{d}\right)}{\Gamma\left(\frac{\gamma(t)}{d}\right)}. \tag{10}$$

The natural logarithm of the gamma function is equal to

$$\ln\left(\Gamma(z)\right) = z(\ln(z) - 1) + \mathcal{O}(\log(z)).$$

Thus, the dominant part of (10) *can be written as*

$$\begin{aligned} &\log\left(d^{t'} \frac{\Gamma\left(\frac{\gamma_f}{d}\right)}{\Gamma\left(\frac{\gamma(t)}{d}\right)}\right) \\ &= \left(t' \ln(d) + \frac{\gamma_f}{d}\left(\ln\left(\frac{\gamma_f}{d}\right) - 1\right)\right. \\ &\quad \left. - \frac{\gamma(t)}{d}\left(\ln\left(\frac{\gamma(t)}{d}\right) - 1\right)\right) / \ln(2) \\ &= \left(\frac{\gamma_f \ln(\gamma_f) - \gamma(t)\ln(\gamma(t))}{d} - t'\right) / \ln(2) \\ &= \left(\frac{\gamma_f \ln(\gamma_f) - \gamma(t)\ln(\gamma(t))}{\gamma_f - \gamma_s} \frac{\alpha}{t} - 1\right) t\log(n) / \ln(2). \end{aligned} \tag{11}$$

Now, the sum in (8) *approaches the following double integral as* n *approaches infinity.*

$$\int_0^\alpha \frac{1}{(t \cdot \ell(t) + c_s)^2} \exp(I(t; \alpha, \gamma_s, \gamma_f))dt,$$

where

$$I(t; \alpha, \gamma_s, \gamma_f) = \int_0^t \frac{\log(\gamma(s)) - \lambda(\gamma(s))}{s\ell(s) + c_s} ds,$$

and

$$\ell(s) = \left(\frac{\gamma_f \ln(\gamma_f) - \gamma(t) \ln(\gamma(t))}{\gamma_f - \gamma_s} \frac{\alpha}{s} - 1 \right) / \ln(2).$$

Now, let $i = t_2$ *in* (2) *to get*

$$N = \sum_{j=1}^{t_2} n_j = \frac{cn - (\log(\gamma_f) + c_s \log(n)) n_{t_2}}{\lambda_f}. \tag{12}$$

The dominant part of this expression can be written as

$$cn \int_0^\alpha \frac{c_s}{(t \cdot \ell(t) + c_s)^2} \exp(I(t; \alpha, \gamma_s, \gamma_f))dt.$$

Like in previous derivations t_1 *steps of plain-BKW with step-size* b *is in total equal to*

$$t_1 \cdot b = (2(c_q - c_s) + 1 - \alpha) \frac{cn}{c_q}.$$

We have $n = N + t_1 \cdot b$. *Using the expression for* N *from* (12) *and solving for* c *finally gives us*

$$\left(\frac{2(c_q - c_s) + 1 - \alpha}{c_q} + \int_0^\alpha \frac{c_s}{(t \cdot \ell(t) + c_s)^2} \exp(I(t; \alpha, \gamma_s, \gamma_f))dt \right)^{-1}.$$

Popular Scientific Summary in Swedish

Populärvetenskaplig sammanfattning

Att kunna skicka krypterade meddelanden är inte längre bara angeläget för militärer och underrättelsetjänster. Varje gång du betalar räkningar via din internetbank eller handlar varor över internet vill du kryptera meddelanden du sänder så att inte obehöriga kan ta del av känslig information.

Symmetrisk kryptering

Tänk dig situationen att Alice vill skicka ett hemligt meddelande till Bob. Inom det som kallas symmetrisk kryptering använder Alice då ett hemligt ord (nyckel) för att göra meddelandet oläsligt (kryptera). Bob använder sen samma nyckel för att göra det krypterade meddelandet läsligt igen (dekryptera). Med symmetrisk kryptering kan stora mängder data skickas snabbt och säkert. Ett problem är dock hur Alice och Bob ska komma överens om en gemensam nyckel. Ett sätt detta kan göras på är med asymmetrisk kryptering.

Asymmetrisk kryptering

Inom asymmetrisk kryptering har Bob två nycklar, en publik nyckel som vem som helst kan se och en privat nyckel som bara Bob kan se. Alice skickar nu ett krypterat meddelande med hjälp av Bobs publika nyckel. Bob dekrypterar sen meddelandet med hjälp av sin privata nyckel.

Kvantdatorer

För att kunna dekryptera meddelandet utan tillgång till den privata nyckeln måste en attackerare lösa ett matematiskt problem. Kryptering idag är baserat på att det är svårt att faktorisera väldigt stora tal eller att hitta diskreta logaritmer i ändliga grupper. Trots decennier av intensiv forskning har ingen lyckats hitta effektiva algoritmer för att lösa dessa problem med en klassisk dator.

En klassisk dator arbetar med bitar, som kan anta värdet 0 eller 1. Ett register med n bitar kan i sin tur vara i något av 2^n tillstånd. En kvantdator är en dator baserad på kvantbitar, där n kvantbitar befinner sig i superposition mellan dessa 2^n tillstånd. När man mäter tillståndet på kvantbitarna kollapsar

dessa till ett av de 2^n tillstånden. Kvantalgoritmer applicerar unitära transformationer på kvantbitarna på ett sätt så att sannolikheten är hög att dessa kollapsar till rätt tillstånd när man väl gör en mätning.

Med hjälp av Shors algoritm kan kvantdatorer både faktorisera tal och hitta diskreta logaritmer på ett effektivt sätt. Idag kan kvantdatorerna bara hantera ett fåtal kvantbitar. Men får vi storskaliga kvantdatorer i framtiden visar det sig att det är lätt att faktorisera stora tal och då behöver vi basera kryptering på andra matematiska problem.

Postkvantkryptering

Forskningsområdet där man studerar ersättare till dagens system för asymmetrisk kryptering kallas postkvantkryptering. Två av huvudspåren inom postkvantkryptering är gitterbaserad kryptering och kodbaserad kryptering.

Gitterbaserad kryptering

Ett gitter i tre dimensioner beskriver den diskreta placeringen av atomer i en kristallstruktur. Matematiskt kan man generalisera denna struktur till godtyckligt antal dimensioner. I högre dimensioner visar det sig vara ett svårt problem att, givet en godtycklig punkt, hitta den närmaste gitterpunkten. Gitterbaserad kryptering är baserad på att detta problem är svårt. I artikel 3 studerade vi hur algoritmer för att lösa detta problem kan förbättras med hjälp av en kvantdator.

LWE

Att lösa linjära ekvationssystem med tusentals ekvationer och obekanta är enkelt för en modern dator. Learning with Errors (LWE) är en variant av detta problem, men med små brustermer tillförda varje ekvation. Ett sätt att lösa LWE på är att översätta problemet till att leta efter den närmaste punkten i ett gitter.

En annan typ av algoritm kallas Blum-Kalai-Wasserman (BKW) och innebär en generalisering av Gausselimination, som används för att lösa brusfria, linjära ekvationssystem. Artikel 1, 2, 5 och 6 handlar om olika aspekter av denna typ av algoritm, från olika tekniker för att förbättra algoritmen till implementering av algoritmen för att lösa konkreta instanser av LWE.

Kodbaserad kryptering

Kodbaserad kryptering använder tekniker från felkorrigerande koder för att kryptera meddelanden. Alice transformerar sitt meddelande till ett kodord i en linjär kod. Sen lägger hon medvetet på nog med brus till meddelandet så att det blir oläsligt för attackeraren. Bob har hemlig tillgång till den linjära kodens struktur och kan med hjälp av denna korrigera bort bruset och på så sätt läsa meddelandet. Normalt används binärt brus (0:or och 1:or). I artikel 4 studerade vi vad som händer om man istället använder normalfördelat brus och visade att detta leder till allvarliga sårbarheter. Algoritmen vi utvecklade för att lösa detta problem visade sig ha applikationer även inom andra områden av kryptering och inom kodningsteknik.